TU Delft

Delft University of Technology

# Modelling, Uncertainty, and Data Analysis for Engineers (MUDE) Textbook

Lanzafame, R.C.; van Woudenberg, T.R.; Verhagen, S.; Cabboi, A.; Mares Nasarre, P.; Lanzafame, R.C.; Verhagen, S.; Arriaga Garcia, J.; Pittman, J.C.; van der Meer, F.P.

**DOI**
10.5281/zenodo.16223061

**Publication date**
2024

**Document Version**
Other version

**Citation (APA)**
Lanzafame, R. C. (Ed.), van Woudenberg, T. R. (Ed.), Verhagen, S. (Ed.), Cabboi, A., Mares Nasarre, P., Lanzafame, R. C., Verhagen, S., Arriaga Garcia, J., Pittman, J. C., van der Meer, F. P., Tiberius, C. C. J. M., Amiri Simkooei, A., Homem de Almeida Correia, G., Nogal Macho, M., Gao, J., Ahmadi, B., Rocha, I. B. C. M., Poot, A., Storm, J., & Riccius, L. F. (2024). *Modelling, Uncertainty, and Data Analysis for Engineers (MUDE) Textbook*. Delft University of Technology. https://doi.org/10.5281/zenodo.16223061

**Important note**
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

**Copyright**
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

**Takedown policy**
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.*
*For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.*

This publication is best viewed as a website:

https://mude.citg.tudelft.nl/book/2024

This PDF-version is created for TU Delft attribution purposes

# Welcome to the MUDE Textbook

## Contents


- Spot a mistake?
- Have a question / answer?
- Confused?
- Credits and Additional Information


Welcome to the MUDE textbook for the 2024-25 academic year. This is where assigned reading is located, along with interactive exercises to practice and study the module material.

**Interactive Pages—Use Python in your Browser!**

This online textbook has a number of pages that are set up to be used interactively. You can use the “Live Code” button under the Rocket Ship icon in the top right to activate the interactive features and use Python interactively!

Sometimes the interactivity will involve completing an exercise, whereas on other pages it might simply provide the opportunity to edit the contents of code cells and execute it to explore the page contents interactively. Other pages may provide interactive figures (e.g., widgets).

This feature is supported by TeachBooks.

## Spot a mistake?

If you spot a mistake, you can click on 💡, login with your NetID and report your issue. It’ll be solved soon!

## Have a question / answer?

If you’ve a question regarding any content, look at https://answers.citg.tudelft.nl/ to see if it’s already there. If not, ask a new question! Your teachers and TAs will look into it soon! If you’ve an answer to on of the question posted there, you’re welcome to answer questions yourself! More info on the answers platform can be found here.

## Confused?

That can happen, let us know how we can make it work for you! Send an email to MUDE-CEG@tudelft.nl

## Credits and Additional Information

For more information about the Editors and Authors, how this book is made and how to reuse the material yourself, see the credits and license page.

# Modelling concepts

> **Attribution**
>
> This chapter was written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

In this chapter, you will be introduced to the basic concepts related to modelling. The chapter is split into four sections, with the aim of each to enable you to discuss and answer the following questions, respectively:

- What is a model and which types of models can be built?
- How do I choose the appropriate model?
- How do I know that a model is accurate?
- How can I assess model performance?

These concepts are generally useful to reflect on all topics with MUDE and many branches of engineering.

# Model classification

## Contents

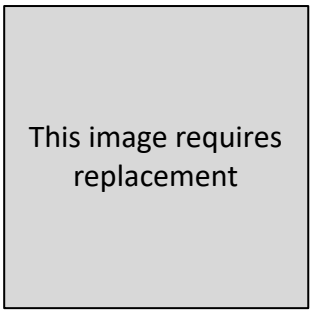

- What is a Model?
- Fit-for-purpose: Modelling Assumptions
- Trade-offs
- Classifying Models


## What is a Model?

> **Note**
>
> **In short:** A model is a purpose built abstraction of physical reality.

In any engineering discipline, **we interact with a system** (Earth, ecosystem, water network, structure, …). Usually, we have questions about how the system behaves and how our intervention will affect it (e.g.: how a water body reacts when the cooling water from a nuclear power plant is discharged), or how the system will impact a proposed intervention (e.g.: a river which may flood close to our road).

To answer that, we build an abstraction of the system which can mimic those aspects we are interested in. For instance, if we focus on the example from the cooling water and the water body, we can model the heat distribution in the water body to know the final temperature. Here we are only modelling some aspects of the system we are studying. What about the response from the species living there? Well, **if we do not need to know about this, and this doesn't affect our results in a meaningful way, our model is good enough to answer our question!**




…

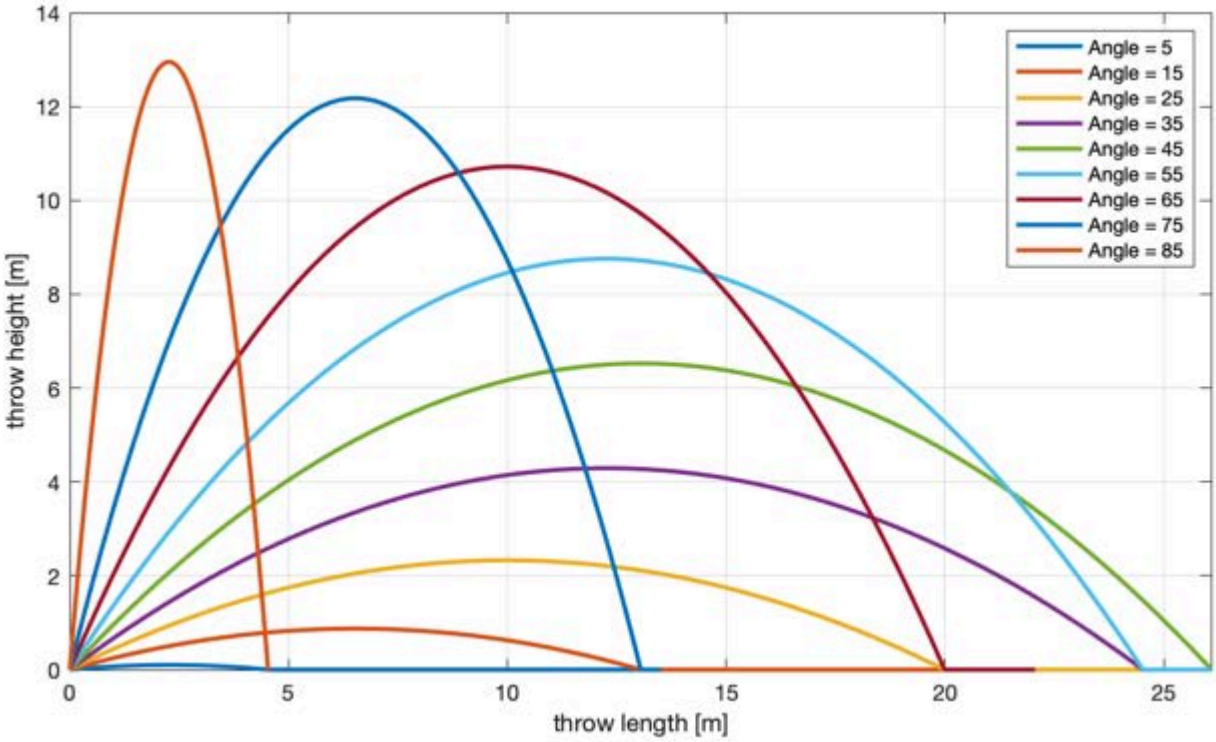
Once we are sure that the model can reproduce the behaviors we are interested in through **mathematization** ,i.e, the implementation of the abstraction, we can further use it to make **computations**. Those computations need to be **interpreted** to get the answer to our actual question. We can also check if our model realistically represents the system we are modelling through **verification** - more about that later.

---

Using a short practical example to illustrate this: imagine you are throwing a projectile, let us say it is a basket ball, and you want to **model** its trajectory. To do so, you would write the following equations of motion of the ball in the $x$ and $y$ direction, and obtain the following the kinematic expressions that entirely define the trajectory of the ball::

$$\begin{cases} x = v_0 t \cos\theta \\ y = v_0 t \sin\theta - \frac{1}{2} g t^2 \end{cases}$$

where $x$ and $y$ are the horizontal and vertical components of the position of the ball at time $t$ , $v_0$ is the initial velocity, $\theta$ is the initial angle and $g$ is the gravity acceleration. The trajectory of the projectile depends on the velocity and angle we shoot the ball. In the figure below, different trajectories depending on $\theta$ and $v_0$ are shown. These are our predicted trajectories based on our very basic model.



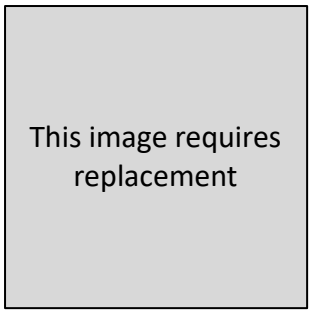

We will discuss other outside factors that may affect the validity of the model in the following sections. For now, can you think of how you may modify this model to reflect the real-life behaviour more accurately?

# Fit-for-purpose: Modelling Assumptions

Our model will be based on a series of assumptions depending on the problem we are solving. It is important to keep in mind that **different purpose** $\Longrightarrow$ **different model**, and that a model needs to be designed in a way that can give us relevant information back. We will now look to different examples that illustrate this.

## Example 1: Car or train?

Imagine you want to predict what would be the choice made by a rational individual to travel from Delft to Paris (i.e. given origin and destination). Would he choose to go by car or by train?




We can consider the simplest model: the traveller will choose the shortest travelling time. In this case, according to our model, the traveller will choose to go by car. However, does this model accurately asess the travellers preference? **No.** Modelling people choices is way more complicated, since they also depend on other factors such as travel cost, comfort, sustainability, etc.

# Example 2: modelling car emissions

Let us now consider a different scenario: we want to predict how much $CO_2$ a conventional internal combustion vehicle will produce on a trip. We can consider, again, the simplest possible models to represent the car dynamics and car emissions.

For the first, let us take a car dynamics model where the velocity over time is constant.

$$v(t) = \bar{v}$$

For the latter, let us take the simplest model to predict the car emissions: the faster you drive, the more $CO_2$ you emit", which can be formulated as:

$$e(t) = av(t) + b$$

where

- $e(t)$ is the emitted amount of $CO_2$.

Then, we can combine the two models obtaining:

$$e(t) = a\bar{v} + b$$

But is this good enough to answer our question and predict the emissions of the vechicle? **No.** In fact, there are lots of factors that gain a stronger importance in this scenario that the model does not take into account. These include:

- Type of the fuel used
- Traffic and driving style

- Air resistance and car profile
- Wind speed and angle
- Many more!

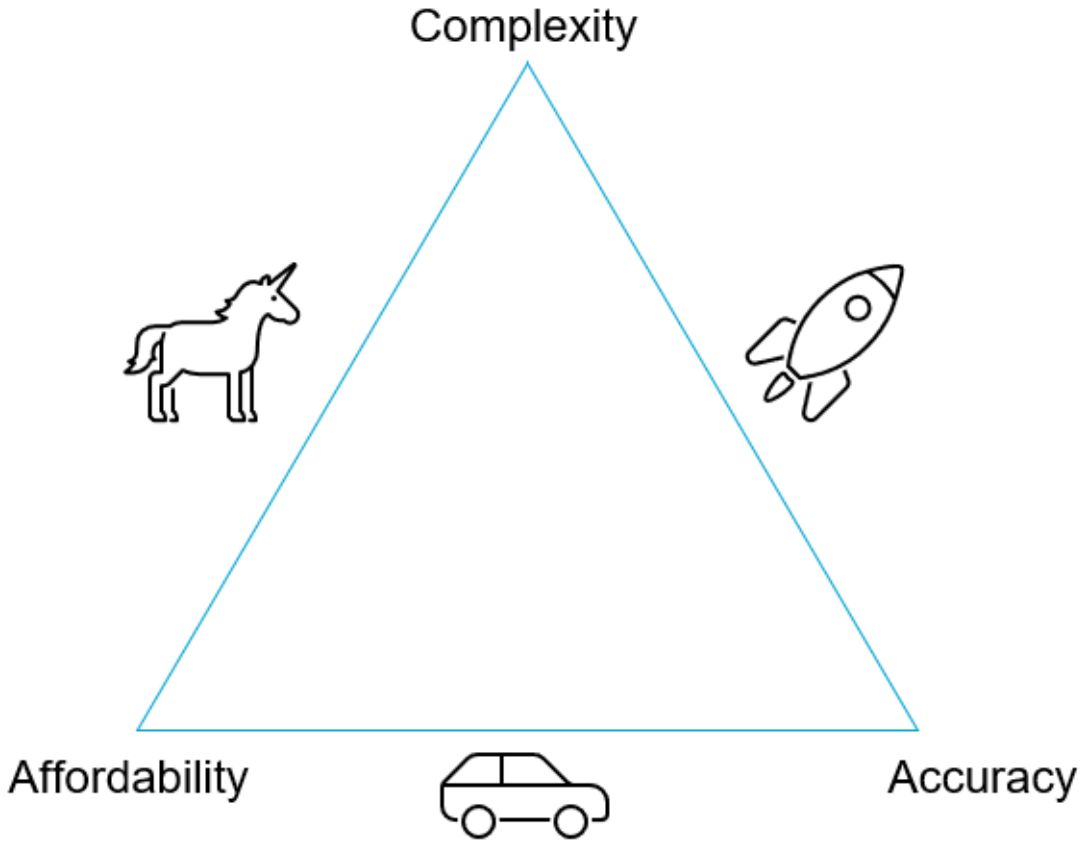
## Conclusions

In the case of both these examples, it is advisable to build the model in a different way. We will need to go back and adjust our assumptions, so that our model can be updated accordingly. This way, we can better capture the complexities of the problem, and include the factors that have the greatest importance in the given scenario.

# Trade-offs

When designing a model, we need the model to be good enough to give reasonable answers to our questions, but we do not want it to be that complex that it is computationally expensive or hinders the interpretation of the results. Therefore, the assumptions made in modelling will take our model to a certain position in this trade-off triangle.

- **Affordability and accuracy:** cheap and functional models, but with a limited scope. i.e., potentially lacking in complexity
- **Accuracy and complexity:** very realistic models, but prohibitively expensive. i.e., not affordable
- **Complexity and affordability:** the unicorn icon might speak for itself - difficult to achieve! i.e., its almost impossible to get an amazing model on a budget, that also accurately shows us what we need!




# Classifying Models

Models can be classified in different ways. Here, we are going to introduce the classification according to their nature. There are four categories:

- **Conceptual models**: representations of how reality looks at the highest level of abstraction. This should be always the first step of any modelling approach, and abstracting a real-case scenario is not always straightforward. In our case, we will generally provide such abstractions directly, so we will not go much more in-depth into this topic here.
- **Mechanistic models**: these models make use of first principle laws from physics, chemistry or biology to describe the behavior of the constitutive elements of the system. They are then systems of mechanistic models which can be very complex.
- **Phenomenological models**: these models rely on the mathematical consistency between quantities based on observations. This is, you take measurements during experiments, you have a look at them under the view of the existing first principles of physics and infer how they are related to each other. Therefore, they do not only rely on data but also they have to be mathematically sound and should obey the basic laws of mechanics and thermodynamics to the extent possible. Phenomenological models are

often only valid for a very specific scale (e.g. micro-scale or macro-scale), depending on the type of experimental data used to derive them, and usually cannot be upscaled or downscaled.
- **Data-driven models**: these models, as the name indicated, make only use of data. Thus, you perform experiments, measure several variables and perform some sort of data analysis, such as regression, to figure out the relationship between input and outputs.

Let's see the last three with some examples.

# Mechanistic Models

As said, these models make use of first principle laws from physics, chemistry or biology to describe the behavior of the constitutive elements of the system. They embodies the idea that a complex system is the summation of many basic phenomena, so a complex system can be modelled as a system of systems.

**Examples:**

- Newton's laws for describing rigid body motion
- Elasticity theory to predict stresses

# Phenomenological Models

They rely on the mathematical consistency between quantities build using, for example, dimensional analysis, and combine it with observations. Thus, the model is constructed by setting up appropriate experiments, measuring observations and fitting hypothetical structure with data (e.g. linear regressions). This allows to develop new theories based on observations of a complex system, rather than relying solely on data, as is done in data-driven models.

**Example - Drag Forces:**

The drag force is the force that appears in an object which moves with respect to the surrounding fluid. We typically calculate it as

$$F_D = \frac{1}{2}\rho v^2 C_D A$$

where

- $\rho$ is the density of the fluid
- $v$ is the velocity of the object with respect to the fluid
- $A$ is the cross-sectional area
- $C_D$ is the drag coefficient, a dimensionless number that depends on the shape of the object and conditions of the surrounding flow. It is typically calculated through relationships with the Reynolds' number - using then a combination of well-known physics (fluid mechanics here) and data.

# Data-driven models

They purely use data to create empirical and probabilistic relations between the desired output and input.

**Example - Forces on a wall:**

Imagine we want to predict the force of waves acting on a wall. To do so, we start by **going to the lab** and performing some experiments and measurements there. We perform them under different conditions, generating waves of different heights in a wave flume and measuring the forces on a model wall that is put inside. With the collected data, we fit a linear regression between the wave height and the measured forces.

This is purely empirical and, thus, a data-driven model! We have simply used data to fit a relationship **input -> output**. These models can be very useful when trying to model very complex systems when there is lots of data available.

> **Attribution**
>
> This chapter was written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Model decisions

## Contents


- Dynamic vs. Static
- Linear vs Nonlinear
- Time-invariant vs Time-variant
- Deterministic vs Stochastic


In this section, we are going to focus on the decisions we make when setting up a model. We will start seeing models as part of the creative and decision making process!

When building a model, we need to identify which simplifications we can make to keep our model as simple as possible, while modelling the system we are interested in with the appropriate accuracy. In real-life applications, simpler models also equates to lower costs, so it is often one of the main considerations in model design. In short, the simpler, the better, as long as it answers our questions. We are going to talk about the following characteristics of a model:

- Dynamic vs. static models
- Linear vs. non-linear models
- Time-invariant vs. Time-variant models
- Deterministic vs. Stochastic models

## Dynamic vs. Static

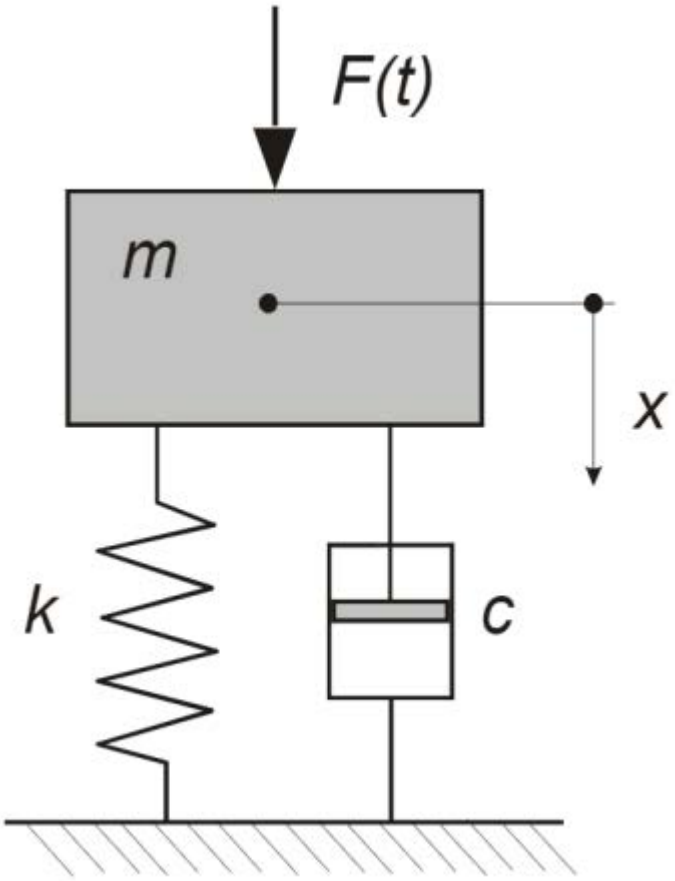
The decision between dynamic and static models is based on whether a model needs to account for previous or future states. In other words, the response of a dynamic model will have a dependence on the past and future states of the system. On the contrary, in a static model, we consider the response of the system only depending on present values and, thus, the model does not account for previous time steps.

**Let’s see an example.**

In classical mechanics, as you know, motion is governed by Newton's laws - the second law in particular - that allow us to write the famous main equation of motion that we have know for years:

$$\underbrace{\sum_{i}^{n} F_i}_{\text{sum of all external forces}} = \overbrace{m\mathbf{a}}^{\text{inertial terms}}$$

Let us consider the mass-spring system as a first example.




For this system, Newton's equations of motion sketch as:

$$F - c\frac{dx}{dt} - kx = m\frac{d^2x}{dt^2} \Leftrightarrow m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F$$

In this equation, a dependence exists between the present and past states of the system. Thus, it is a dynamic model. This equation can be tricky to solve: it is both dynamic and nonlinear (see below)… However, it is also possible to simplify it!

If the terms with the derivatives are small and negligible when compared to $x$, then we find the following equation of motion:

$$\frac{d^2x}{dt^2}, \frac{dx}{dt} << x \implies kx = F$$

**Under which circumstances does this occur?**

- *Slow loading rate:* constant $F$ or $F\cos(\omega t)$ if $\omega < \omega_0$, where $\omega_0$ is the natural frequency of the system
- Small mass and small values of the damping constant, $c$

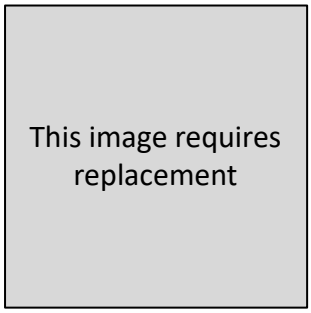
In those circumstances, we can make the assumption that the model is static (simplification) and still get accurate answers to our questions using the simplified model.

# Linear vs Nonlinear

In linear models, the relationships between variables can be modelled using linear predictor functions. The advantage of these models is that the superposition principle applies and the long term behavior does not depend on initial conditions. On the contrary, in non-linear models, the superposition principle does not apply anymore and long and short term behavior is highly dependent on the initial conditions.

**Let's see an example.**

Let us consider one of the most typical Physics examples: the simple pendulum.




The moment equation around the pivot point corresponds to a nonlinear dynamic model:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\sin\theta = 0$$

However… it can be linearized around a stable position, where we use $\sin\theta \approx \theta$. This assumption is reasonable when the motion of the pendulum is small compared to the length of the pendulum. Thus, the equation of motion will become:

$$\frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0$$

# Time-invariant vs Time-variant

In time-invariant models, the model coefficients or model parameters are constant. This is, the response or properties of the system do not change over time. In time-variant models, the model parameter do change over time, so the behavior of the system evolves.

**And how do I choose between a time-variant and time-invariant model?**

Mainly based on the knowledge I have about the system! As an engineer, **I will have to assess whether or not such changes are relevent for my problem.** For instance, nowadays, we consider climate change when designing a breakwater. It has been proved that waves and storms are becoming more extreme over time. I will have to account for this to assess the performance of the breakwater. However, if I am designing a temporary platform to assist in some construction works in the sea, I will not take it into account. Climate change is a long-time scale process. It will impact on the life cycle of a breakwater which is meant to be there for tens or even hundreds of years. On the other hand, the influence of climate change on a platform that will only be there for a few weeks is negligible.

# Deterministic vs Stochastic

Deterministic models are those which for some given inputs, always provide the same output. For instance, a equation which gives the average concentration of $CO_2$ in a city as function of the traffic. For a certain value of traffic, the model will always provide the same concentration of $CO_2$. In these models, there is no uncertainty (other than the uncertainty of the input data). Stochastic models **embrace** the uncertainty. This is, stochastic models will produce different outputs for a given input. In fact, the inputs and outputs of stochastic models are probabilistic distributions (you will learn more about this later!), which relate the values of the variable with the probability of observing it.

**And how do I choose between a deterministic and stochastic model?**

All systems in reality are stochastic, since we never truly know the actual properties and inputs. However, under certain circumstances, this *stochasticity* can be neglected. Let us take a look to some examples of deterministic and stochastic systems:

**Cases of stochastic systems:**

Structures or systems with known or unknown properties subject to unknown or known loads, respectively:

- Soil systems: we can extract samples of soil but we cannot see or measure the properties of every point in the system.
- Traffic systems: inputs are too complicated, from weather to human decisions.

Note that there are also uncertainties regarding the measurement of the observations (you will also learn about this later!).

**Cases of deterministic systems:**

Known structures subject to known static or dynamic loads. For instance, when we decide to model the response of a building (deformations) under a specific (known) load. Note, however, that in practice loads are almost always variable, or impossible to predict with certainty; thus we will often use deterministic models to understand the behavior of the structure of interest, while also using a stochastic models to check whether the structure will perform well under the range of loads that can be expected.

> **Attribution**
>
> This chapter was written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Verification, Calibration and Validation

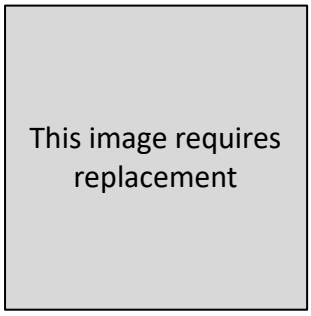
## Contents


- How do we know a model fits the purpose?


> **Note**
>
> *All models are wrong, but some are useful.*
>
> George Box (1913-2013)

We have already discussed that, when building a model, we have to make some decisions: depending on the system I am studying and the questions I have, I will have to make some simplifications. That leads to the previous statement (*'all models are wrong'*), because they only model the part of the system we are interested in with the required accuracy to answer our questions (*'but some are useful'*). So then… **how can they even be considered valid?**

Let us consider two different models, the mass-spring system (model 1) and a tower (described by a complex numerical model, making using of the Finite Element Method).




Both models do **not exactly represent** the real structure, but they can be valid models in which we can trust to answer to certain research questions. For example:

- **Model 1:** What is the fundamental frequency of the structure?

Knowing the spring constant, $k$, the damping coefficient, $c$, and the mass we have on top, then we can exactly determine the natural (fundamental) frequency of the structure.

- **Model 2:** How does the structure deform at its fundamental frequency?

We need to use a more complex model to answer to this more complex question. In thism case, the Finite Element Method (More on that in Q2) would be an appropriate one.

**Conclusion:** Our models are valid if they fit the purpose and answer our research questions!

> **But… what do we mean with research questions?**
>
> Well, let us enumerate a few examples so you can have a clearer idea!
>
> - What is the maximum water depth during the flooding of city?
> - What is the concetration of a contaminant in a given location of my reservoir?
> - How does the dynamic behavior of a structure change if the interaction with the soil is taken into account?

# How do we know a model fits the purpose?

Essentially, **models need to be verified and validated to fit the purpose**. It is important to keep in mind that verification $\neq$ validation!

## Verification

Verification is the process of checking whether the model is correctly implemented with respect to the original conceptual model. It should answer the question: *Have we built the model right?*

For example, we could check if:

- **It matches the assumptions**

If we assumed a linear behavior, for example, then the model's responses should obey the linear superposition principle.

- **All the main required aspects/actions are implemented**

We can do this by comparing our model with a flow diagram that includes each logical action and check whether the model performs such actions.

We can also do a structured walk-through/one-step analysis. i.e., explain your steps to a non-expert person (or a rubber duckie!). This helps you to go throught the modelling steps again.

- **The results look reasonable for known input values (this can also be validation)**

We can the model's results against analytical results (if possible) or benchmark solutions. This could also be done for parts of the model.

- **No numerical or math errors exist**

We can try unning the model for extreme cases. Checking asymptotic convergence of simulations or verifying a good representation of floating-point numbers are a few examples of performing this step.

# Validation

Validation is the process of testing the ability of the model in answering the research questions as best as possible. It should answer the question: *Have we built the right model?*

## How do we validate a model?

First, we need to define features, goodness of fit metrics, and acceptable error ranges (limits and tolerances).

- **Feature**: The quantity that is chosen to be predicted. For example, the peak stress of a specific structural component or the peak concentration of a contaminant.
- **Goodness of fit metric**: Quantitative measures of the distance or error between the predicted feature by the model and the reference value of the chosen feature (you will learn more about this in the next section).
- **Limits and tolerances**: How much deviation from the reference value that is acceptable. As we introduced, models are not perfect and, thus, there will be some error in the predictions. An appropriate model is that whose error is negligible to answer the research question.

Once we have our meanurable parameters defined, the key phase of the validation process takes place: the comparison of the results of our model with the reference value. As a result,

we will be in two possible scenarios:

1. We find a significant/unexpected mismatch. Then, we need to go back to our original assumptions,review them and make changes and modify the model. We will probably need to make our model more complex. Once the results are within our tolerances, then we have **validated** the model.
2. We find a small/expected mismatch, then we can **calibrate** the model to make it more accurate. i.e., adjust/tune the model parameters to improve the match.

## How is the model comparison performed?

There are many ways to do this and will depend on the nature of the model and the data available. Here is an example from the machine learning community:

To compare the model with either the experiment or the validated models, we assume we have a certain number of data points, obtained from experiments or from another validated model.

We split this data points into two different sets: (1) dataset 1 for first comparison and calibration, and (2) dataset 2 for validation. There is not strict rule about the size of each dataset and it may depend on the field. As a first suggestion, it can be divided in 40%-60% for datasets 1 and 2, respectively.

Once calibrated, we can compare the model (using the same features and metrics) against:

- The remaining set of data from experiments or simulated results
- Future collected data

**It is important to note that we do not test the model against the data it was calibrated on!**

> **Tip**
>
> In some applications, you might want to calibrate a model beforehand using some existing data to help build the model (i.e., training a model). In this course, we do not concern ourselves with this type of calibration, only the type applied after the model has been validated.

# Final notes on model validation

Be very systematic in verifying and validating models:

- Have a plan for experiments and model simulations to perform;
- Justify the chosen features and metrics;
- Mark out which changes have been made to the model;
- Make a track list of the corrected mistakes;
- Keep note of what changed in the results, for each modification performed in the model;
- Clearly highlight for which parameter space the model is valid.

> **Attribution**
>
> This chapter was written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Goodness of Fit

## Contents


- How does it look?
- Square differences
- Bias

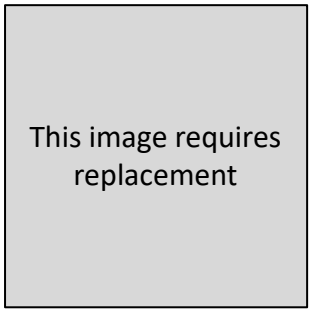

In the previous sections we have introduced the concept of model and the need of selecting an appropriate model *ad hoc* for our problem. Also, you have been introduced to the concepts of verification, validation and calibration. In the validation process, we compared the results of our model with known reference values to assess the performance of the model. **But how good is the performance?** If there are different verified and validated models I can use, **which one is better?**

We can make use of **goodness of fit (GOF) techniques** to support our answers to those questions. GOF techniques are objective and (mainly) quantitative metrics which provide insight into the performance of models. It is important to note that GOF techniques are not a ground truth, but an objective way of comparing models. Different techniques may lead to different results and it is the engineer who has to balance those outputs and select the best model. Thus, it is recommended to use more than one GOF technique in the decision-making process. In the subsequent sections, some commonly used GOF techniques are presented.

In order to illustrate these techniques, the following example will be used. Waves suffer several physical processes when travelling from offshore to the coast (namely, refraction, diffraction, shoaling, reflection and breaking). We typically have wave measurements offshore and need to propagate them using a propagation model to the coast to assess the loads on, for instance, a water intake or a structure. In this example, we have measurements of waves both offshore and near the shore obtained through physical model tests (small-scale tests). We are going to assess how accurate the performance of the propagation model is for two wave features: the significant wave height ($H_{m0}$) and the mean wave period ($T_m$) [1]. GOF techniques will be applied to quantify this performance.

# How does it look?

The first step when addressing a problem is to visualize it. Thus, the first GOF technique recommended here (note that it is not a metric) is visual inspection. It is common practice to plot the reference values against the predictions given by our model, as shown below. The perfect fit would correspond to all the points aligned with the 45-degrees line.




Just using the above plot, we can see that our model tends to underpredict $T_m$. Also, we can see that the performance to estimate $H_{m0}$ is better than for $T_m$, since the points are way closer to the 45-degrees line. But **how much better?**

# Square differences

In many fields, it is common practice to use GOF metrics based on the squared errors between the measurements and the predictions ($SE$).

$$SE_i = (y_i - \hat{y}_i)^2$$

where $y_i$ are the observations and $\hat{y}_i$ are the predictions.

Note that it does not matter the sign of the error ($y_i > \hat{y}_i$ or $y_i < \hat{y}_i$), $SE$ will always be positive. This is computationally very convenient in optimization processes, such as when calibrating the coefficients of a model minimizing the $SE$. Convergence is faster when the objective function ($SE$) does not change from one sign to another between iterations but only changes in absolute value (you will learn more about this later in the course).

Within this family of metrics, we will focus here on two: the Root Mean Square Error ( $RMSE$) and the coefficient of determination ($R^2$).

# Root Mean Square Error ($RMSE$)

The Root Mean Square Error ($RMSE$) is defined as

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N}} = \sqrt{MSE}$$

where $N$ is the number of observations and $MSE$ is the mean squared error computed as

$$MSE = \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{N} = \frac{\sum_{i=1}^{N}(SE_i)^2}{N}$$

$RMSE$ is a very intuitive metric since it represents the mean error between the observations and the predictions. Therefore, the lower the $RMSE$, the better. Moreover, it has the same units as the studied variable, so we can put into the context of our design process the obtained $RMSE$ and assess whether it is acceptable. For instance, in our example, $RMSE$ will be in centimeters and seconds for $H_{m0}$ and $T_m$, respectively. Computing this metric in our example, we obtain $RMSE(H_{m0}) = 0.65cm$ and $RMSE(T_m) = 0.016s$. If we see the magnitude of $RMSE$ with respect to the magnitude of the variables ($H_{m0} \in [5, 20]cm$ and $H = T_m \in [0.8, 1.8]s$), the error seems reasonable.

But **which variable is better predicted by the model?** We would need an standardized GOF metric to this end.

# Coefficient of determination ($R^2$)

The coefficient of determination ($R^2$) is an standardized version of the $RMSE$, which the $MSE$ dimensionless using the variance of the observations as

$$R^2 = 1 - \frac{MSE}{Var(y)} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \overline{y})^2}$$

where $\overline{y}_i$ is the average of the observations.

Therefore, $0 \leq R^2 \leq 1$ represents the percentage of variance explained by the model and the higher, the better the performance of the model. The main advantage is that the value of the metric does not depend on the units of the variable easing the comparison process.

Moreover, if the natural variability of the process is known, the $R^2$ can be interpreted into that context. For instance, if the natural variability of a phenomenon is around 5%, $R^2 \approx 95\%$ would represent a very accurate model. If the studied phenomenon is way more uncertain with variabilities up to 30%, $R^2 \approx 70\%$ would be a satisfactory model.

If we compute this metric in our example, we obtain $R^2(H_{m0}) = 0.94$ and $R^2(T_m) = 0.68$. Using the $RMSE$, it was complicated to choose which variable was better modelled. With $R^2$, we can conclude that $H_{m0}$ is better modelled than $T_m$. Moreover, considering that the natural variability of waves is low, the performance for $H_{m0}$ is satisfactory while the performance or $T_m$ is reasonable.

# Bias

So far, the GOF metrics introduced were focused on quantifying the scatter of the predictions around the perfect fit. Another interesting aspect to quantify in our model is its bias. The bias is the systematic tendency of our model to under- or over-prediction. This is, how skewed the predictions of our model are and towards which side.

Between the existing bias metrics, we are going to introduce the relative bias ($rbias$).

# Relative bias ($rbias$)

The relative bias ($rbias$) is a standardized dimensionless measure of bias which can be computed as

$$rbias = \frac{1}{N}\sum_{i=1}^{N}\frac{\hat{y}_i - y_i}{|y_i|}$$

$rbias$ quantifies the mean ratio between the error and the absolute value of the variable. Therefore, $-1 \leq rbias \leq 1$ and the closer to 0, the better. If $rbias < 0$, the predictions provided by the model are in average lower than the reference values, and viceversa. Therefore, $rbias$ gives us information about how much my model deviates from the reference values (magnitude of $rbias$) and the tendency of the model to under- or over-predict the studied variable (sign of $rbias$).

If we compute this metric for our example, we obtain $rbias(H_{m0}) = 0.05$ and $rbias(T_m) = -0.09$. These results tell us that the model tends to slightly overpredict $H_{m0}$ and underpredict $T_m$.

> **Attribution**
>
> This chapter was written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

Data extracted from Mares-Nasarre et al. (2022). Hydraulic stability of cube-armored mound breakwaters in depth-limited breaking wave conditions, *Ocean Engineering* 259, 111845. https://doi.org/10.1016/j.oceaneng.2022.111845

# Propagation of Uncertainty

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

In engineering and sciences we often work with functions of random variables, since when estimating or modelling something, the output is a function of the random input variables, see Fig. 2.1

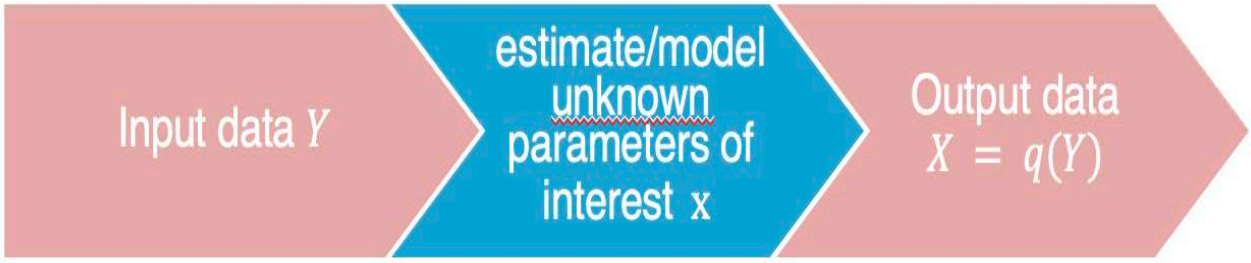



*Fig. 2.1* Output $X$ of a model is function of random input $Y$.

Some simple examples are:

- conversion of temperature measured in degrees Celsius to temperature in degrees Fahrenheit: $T_f = q(T_c) = \frac{9}{5}T_c + 32$
- taking the mean of $m$ repeated measurements $Y_i$: $\hat{X} = q(Y_1, \ldots, Y_m) = \frac{1}{m}\sum_{i=1}^{m} Y_i$
- subsurface temperature $T_z$ as a function of depth $Z$ and surface temperature $T_0$ and known $a$: $T_z = T_0 + aZ$
- wind loading $F$ on a building as function of area of building face $A$, wind pressure $P$, drag coefficient $C$: $F = A \cdot P \cdot C$
- Evaporation $Q$ using Bowen Ratio Energy Balance as function of the net radiation $R$, ground heat flux $G$, bowen ratio $B$: $Q = q(R, G, B) = \frac{R-G}{1-B}$

Fig. 2.2 shows an example of the distribution of the average July temperature in a city, both in degrees Celsius and degrees Fahrenheit. Due to the change of units, the PDF is transformed, the mean is shifted and the variance changed.

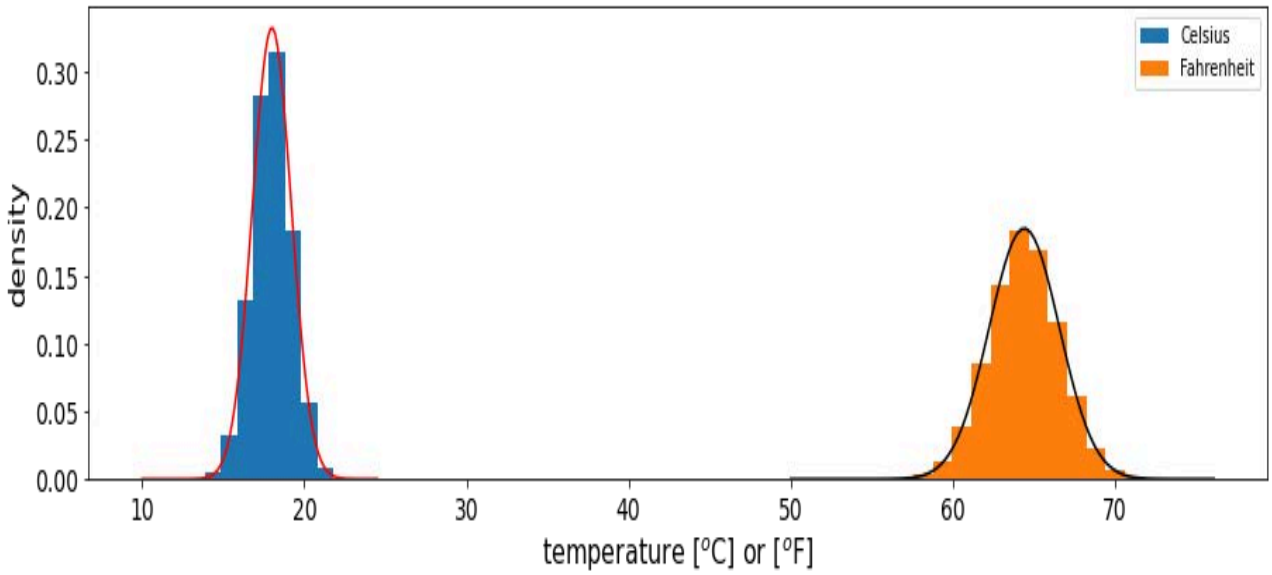



*Fig. 2.2* Distribution of temperature in degrees Celsius and degrees Fahrenheit.

The question we are interested in is: **how does the statistical uncertainty in input data propagate in the output variables?**

In this part, we will first introduce random vectors, covariance, correlation and covariance matrices, as well as the multivariate normal distribution, uncertainty classifications, and then look at the propagation laws that we need.

# Random vectors, covariance and correlation

## Contents


- Covariance and correlation
- Covariance matrix


When dealing with multiple random variables we need to consider that these may not be independent. Instead of considering the individual random variables, we will then need to work with a random vector $X = [X_1 \quad X_2 \quad \dots \quad X_m]^T$, which has a *multivariate* (or: joint) distribution. We will first introduce the covariance and correlation coefficient of two random variables, and then introduce the multivariate normal distribution.

## Covariance and correlation

The covariance $Cov(X_1, X_2)$ is a measure of the joint variability of the two random variables $X_1$ and $X_2$. It gives us information about whether and how the two variables are correlated and it can be either positive or negative (hence, $Cov(X_1, X_2) \lessgtr 0$).

By definition

$$Cov(X_1, X_2) = \mathbb{E}([X_1 - \mathbb{E}(X_1)][X_2 - \mathbb{E}(X_2)])$$

where $\mathbb{E}(X_1)$ is the expected value of the first random variable, for instance.

Recall: the expected value of a random variable $X$ is

$$\mathbb{E}(X) = \mu_X = \int_{-\infty}^{\infty} x f_X(x) dx$$

where $f_X(x)$ is the probability density function of $X$.

If $Cov(X_1, X_2) > 0$ it means that high (low) values of $X_1$ occur together with high (low) values of $X_2$, therefore the covariance is defined as *POSITIVE*.

On the other hand, if $Cov(X_1, X_2) < 0$ it means that high (low) values of $X_1$ occur together with low (high) values of $X_2$. In this case the covariance is defined as *NEGATIVE*.

Note that:

$$Cov(X_i, X_i) = \mathbb{E}([X_i - \mathbb{E}(X_i)]^2) = \sigma_i^2$$

In order to better understand the relation between the two random variables we can also compute the Pearson correlation coefficient

$$\rho(X_1, X_2) = \frac{Cov(X_1, X_2)}{\sigma_{X_1}\sigma_{X_2}}$$

which in fact is a measure of the strength of the *linear relationship* among the variables.

The correlation coefficient by definition takes a value between -1 and 1. If $\rho_{ij} = 0$ the random variable are uncorrelated, this is the case if the random variables are *independent*. If $\rho_{ij} = \pm 1$ the variables are fully correlated: knowing the value of one variable, means that the value of the other variable is also known, and the two variables have a linear relation. A positive correlation coefficient means that if one variable increases, the other one tends to increase as well; conversely a negative correlation means that an increase of one variable is accompanied by a tendency of the other variable to decrease.

Examples are shown in Fig. 2.3, for an example with a large number of repeated measurements; for this example $\sigma_{X_1} = 1$ and $\sigma_{X_2} = 2$ (the mean values are indicated as $\mu_1$ and $\mu_2$). The larger standard deviation of the second measurement results in a larger spread in the vertical direction. Obviously, the measurements fluctuate around the means.

Generally it is quite difficult to find cases in which $\rho = \pm 1$ or even $\rho = 0$. In fact it is more likely to have values of $\rho$ that range between $-1$ and $1$ and, based on this value, we can determine the strength of the linear dependence.

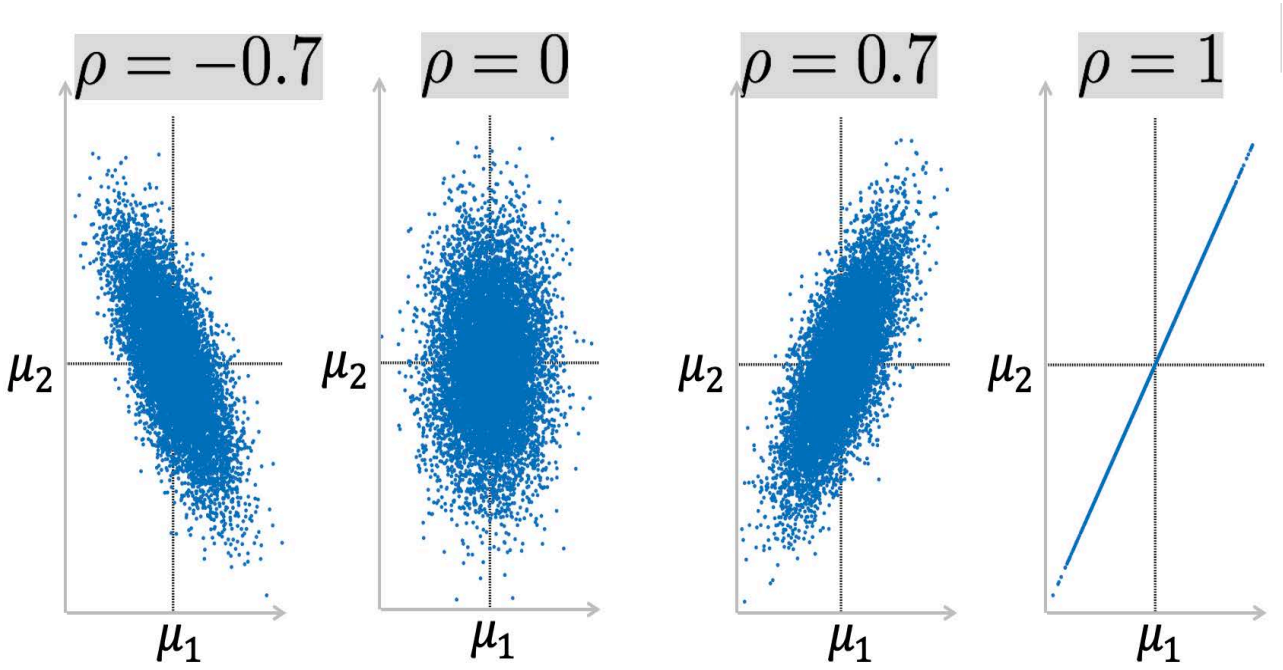



*Fig. 2.3* Scatterplots of outcomes of $(X_1, X_2)$ with different correlation coefficients.

The interactive element below allows you to play around with the correlation value yourself. Observe how the distribution's density contours, or the scattered data, changes when you adapt the correlation value.

**Attribution**

This interactive figure is created by Max Ramgraber. Find out more here.

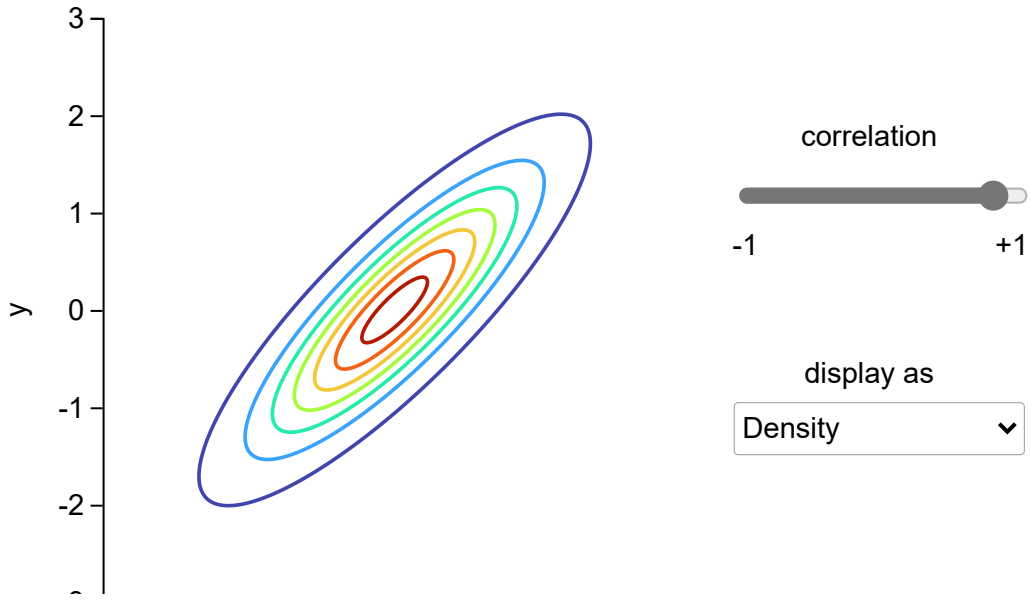


*Fig. 2.4* Interactively change the correlation coefficient to visualize the effect on density contours or samples of the bivariate Gaussian distribution.

# Covariance matrix

When considering a random vector $X = [X_1 \quad X_2 \quad \dots \quad X_m]^T$, we can 'collect' all covariances in the so-called *covariance matrix*:

$$\Sigma_X = \begin{bmatrix} \sigma_1^2 & Cov(X_1, X_2) & \dots & Cov(X_1, X_m) \\ Cov(X_1, X_2) & \sigma_2^2 & \dots & Cov(X_2, X_m) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_1, X_m) & Cov(X_2, X_m) & \dots & \sigma_m^2 \end{bmatrix}$$

Note that the covariance matrix is symmetric, since $Cov(X_i, X_j) = Cov(X_j, X_i)$.

If all measurements are independent, all covariances will be equal to zero, and the covariance matrix becomes a diagonal matrix with the variances on the diagonal.

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Multivariate normal distribution

## Contents


- Play with bivariate normal PDF


If all $X_i$ in the random vector $X = [X_1 \quad X_2 \quad \dots \quad X_m]^T$ are normally distributed, then $X$ will have the *multivariate normal distribution*, for which the PDF is given as:

$$f_X(x) = \frac{1}{\sqrt{\det(2\pi\Sigma_X)}} \exp(-\frac{1}{2}(x-\mu_X)^T\Sigma_X^{-1}(x-\mu_X))$$

which is determined by the expectation $\mu_X$ and covariance matrix $\Sigma_X$.

Notation: $X \sim N(\mu_X, \Sigma_X)$

An example of the bivariate normal PDF is shown in Fig. 2.5; in this example $\sigma_{X_1} = 1$ and $\sigma_{X_2} = 2$ and $\rho(X_1, X_2) = 0.7$.

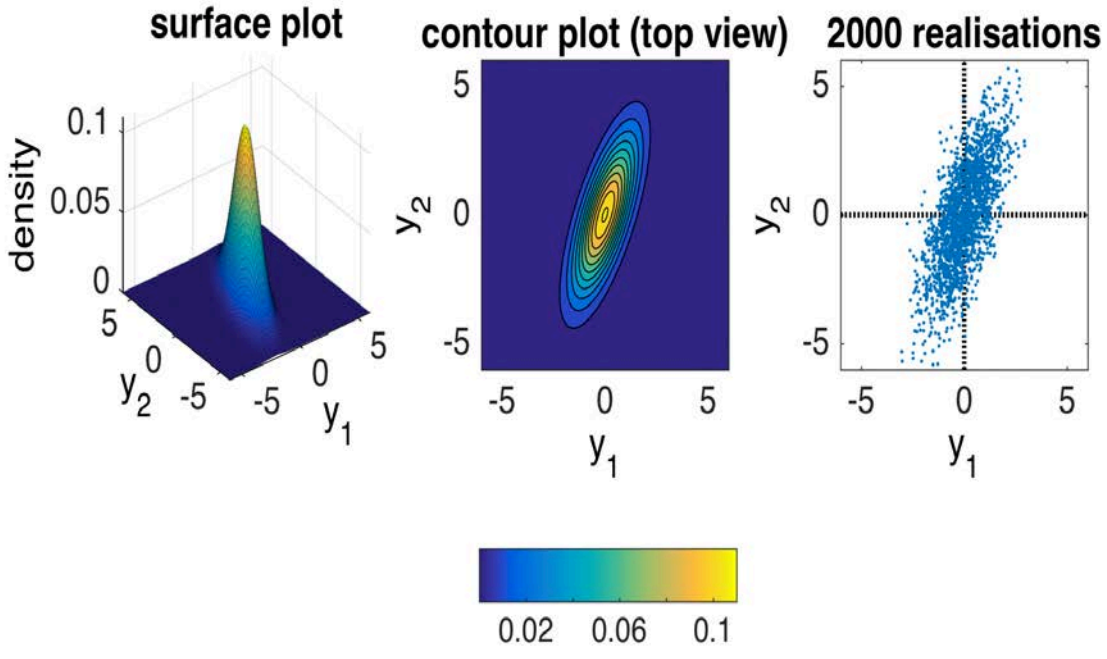



*Fig. 2.5* Example of bivariate normal PDF and corresponding scatterplot with 2000 realizations.

# Play with bivariate normal PDF

In this small demo you can create plots of the bivariate normal PDF, including a scatter plot of generated realizations, where you can play with the values of the mean, standard deviations and correlation coefficient. Adjust the values by dragging the handles for the mean (red), standard deviations (blue and yellow), and correlation (green) in the left subplot. Observe how the distribution changes, and how each parameter affects the covariance matrix.

> **Attribution**
>
> This interactive figure is created by Max Ramgraber. Find out more here.

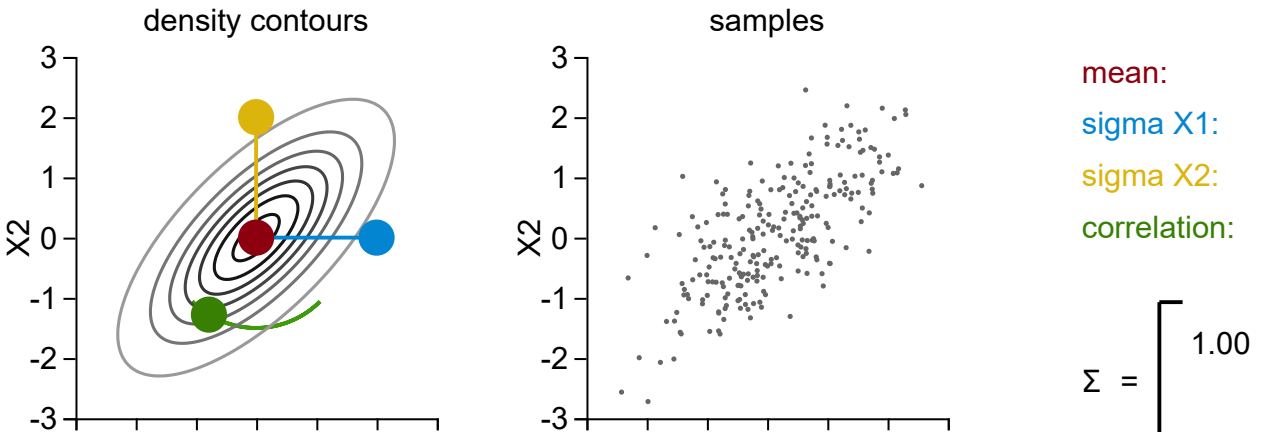



*Fig. 2.6* Interactively change the means, standard deviations and correlation coefficient for the bivariate Gaussian PDF and visualize how the density contours, samples and covariance matrix change.

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Uncertainty Classification

## Contents


- Aleatoric
- Epistemic
- Error
- A note about modelling uncertainty


All of us have an idea of what is uncertainty, but we may use different names for it: unpredictability, randomness, ambiguity, variability… Thus, we can say that something is uncertain if we don't know for sure (*not a big surprise, right?*).

In the context of modelling and enginering, it can be useful to classify the sources of uncertainty since it allows us to identify, quantify and, if possible, reduce it. In general, we can distinguish three main types of uncertainty: aleatoric, epistemic and error.

## Aleatoric

The word *aleatoric* derives from the Latin *alea*, which means the rolling of dice. Therefore, when we talk about the aleatoric uncertainty, we refer to the one which is an intrinsic phenomenon and is typically associated with variations that occur in nature. For instance, if we want to predict river discharge next year for Amazon River, it is not possible to use models and data to reduce the uncertainties, since they occur in the future and a range of values is possible.

## Epistemic

The word epistemic derives from the Greek *episteme*, which means knowledge. Therefore, an epistemic uncertainty is the one caused by the lack of knowledge. For instance, if I have only measured the width of a river 5 times over a length of a few km, I can only guess at the true width in between those measurement points. Since there are no other sources of information, there is not much I can say about those widths, therefore they are uncertain. If I invest time and resources making more measurements of the river width, the uncertainty will

decrease. Thus, epistemic uncertainty is sometimes a matter of time and resources; at some point it is better to use a probability distribution to take the uncertainty into account rather than to collect more information to reduce it.

# Error

Error stands for any identifiable deficiency in any stage of modelling and simulation that is not due to lack of knowledge. For instance, the gauge will have some error when measuring the discharge. Sometimes this is also called model error, and sometimes it is also grouped together with epistemic uncertainty.

# A note about modelling uncertainty

In MUDE we will typically use probability distributions to model uncertainty because it is a relatively simple and straightforward approach. In fact, we focus mostly on continuous parametric distributions. Since we often use the same distributions to model all sources of uncertainty, the ability to classify it into different categories can be useful for interpreting the results of our stochastic models.

Note that there is no universal uncertainty classification. You can read more about this topic in [1] or [2].

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.


Der Kiureghian and Ditlevsen (2009). Aleatory or epistemic? Does it matter? *Structural safety* 31, 2, 105-112. https://doi.org/10.1016/j.strusafe.2008.06.020

Van Gelder (2000). Statistical methods for the risk-based design of civil structures. PhD thesis. TUDelft. http://resolver.tudelft.nl/uuid:6a62d6fa-cbcc-4c38-af8a-027c3d191a9d

# Mean and variance propagation laws

## Contents


- Video
- What are the mean and variance of $X$?
- Function of one random variable
- Function of two random variables
- Functions of $n$ random variables
- Video


Here, will only consider the propagation of the mean (= expectation) and (co-)variances, and not the transformation of the full PDF or CDF.

The general form of our problem will be given as follows. Consider the single function of $m$ random variables

$$X = q(Y) = q(Y_1, \ldots, Y_m)$$

with the mean and covariance matrix of $Y$ known:

$$\mathbb{E}(Y) = \mu_Y, \quad \mathbb{D}(Y) = \Sigma_Y$$

## Video

In this video the first part of this chapter is explained. You can also first read this section, and only watch the video if you like someone to explain it.

# What are the mean and variance of $X$?

Let's start with the mean for the case that the function is linear:

$$q(Y) = a_1 Y_1 + a_2 Y_2 + \cdots + a_m Y_m + c$$

with the $a_i$ and $c$ deterministic constants. Since the expectation operator is a linear operator, we have:

$$\mathbb{E}(q(Y)) = \mathbb{E}(a_1 Y_1 + a_2 Y_2 + \cdots a_m Y_m + c) = a_1 \mathbb{E}(Y_1) + \cdots + a_m \mathbb{E}(Y_m) + c$$

But what if the function is non-linear? Then we can use the Taylor series approximation of $q(Y)$.

# Function of one random variable

We will first look at the simplest case, where we have a function of a single random variable, $X = q(Y)$, with the Taylor approximation:

$$q(Y) \approx q(\mu_Y) + \left(\frac{\partial q}{\partial Y}\right)_0 (Y - \mu_Y) + \frac{1}{2!}\left(\frac{\partial^2 q}{\partial Y^2}\right)_0 (Y - \mu_Y)^2 + \text{H.O.T}$$

where we take the mean $y_0 = \mu_Y$ as the logical initial guess of the random vector $Y$. (H.O.T. stands for higher-order terms). The subscript $_0$ indicates that the partial derivatives are evaluated at $y_0 = \mu_Y$.

Due to the linearity of the expectation we then find as a second-order approximation of $\mathbb{E}(q(Y))$, known as *mean propagation law*:

$$\begin{aligned}\mathbb{E}(X) = \mathbb{E}(q(Y)) &\approx \mathbb{E}(q(\mu_Y) + \left(\frac{\partial q}{\partial Y}\right)_0 (Y-\mu_Y) + \frac{1}{2!}\left(\frac{\partial^2 q}{\partial Y^2}\right)_0 (Y-\mu_Y)^2) \\ &= \mathbb{E}(q(\mu_Y)) + \left(\frac{\partial q}{\partial Y}\right)_0 \mathbb{E}\left((Y-\mu_Y)\right) + \frac{1}{2}\left(\frac{\partial^2 q}{\partial Y^2}\right)_0 \mathbb{E}\left((Y-\mu_Y)^2\right. \\ &= q(\mu_Y) + \frac{1}{2}\left(\frac{\partial^2 q}{\partial Y^2}\right)_0 \sigma_Y^2\end{aligned}$$

where in order to arrive at the last equation we should recognize that:

- $\mathbb{E}(q(\mu_Y)) = q(\mu_Y)$ (since $\mu_Y$ is deterministic and known)
- $\mathbb{E}(Y-\mu_Y) = \mathbb{E}(Y) - \mu_Y = 0$
- $\mathbb{E}\left((Y-\mu_Y)^2\right) = \sigma_Y^2$.

For the variance of $X = q(Y)$ it generally suffices to use a first-order approximation. The result follows as:

$$\sigma_X^2 = \left(\frac{\partial q}{\partial Y}\right)_0^2 \sigma_Y^2$$

and is referred to as the *variance propagation law*.

**Example $X = Y^2$**

$$\begin{aligned}\mathbb{E}(X) &\approx \mu_Y^2 + \frac{1}{2}\cdot 2 \cdot \sigma_Y^2 = \mu_Y^2 + \sigma_Y^2 \\ \sigma_X^2 &\approx (2\mu_Y)^2 \sigma_Y^2 = 4\mu_Y^2 \sigma_Y^2\end{aligned}$$

**Example gravitational force**

The gravitational force is given by \(F =g \cdot M\). The mass \( M\) is a random variable with mean \(\mu_M= 100 [kg]\) and standard deviation \( \sigma_M=0.2 [kg]\). What is the variance of \(F\) if \(g = 9.81 [m/s^2]\)? *Give your answer rounded to 2 decimal places.*

\(\sigma_F^2 \approx\)

Check

# Function of two random variables

Let's consider the case that we have one function of two random variables, $Y = [Y_1 \;\; Y_2]^T$ with known mean and covariance matrix:

$$\mathbb{E}(Y) = \mu_Y = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \quad \Sigma_Y = \begin{bmatrix} \sigma_1^2 & Cov(Y_1, Y_2) \\ Cov(Y_1, Y_2) & \sigma_2^2 \end{bmatrix}$$

The Taylor series approximations of $X = q(Y_1, Y_2)$ follow as:

$$\mathbb{E}(X) \approx q(\mu_Y) + \frac{1}{2}\left(\frac{\partial^2 q}{\partial Y_1^2}\right)_0 \sigma_1^2 + \frac{1}{2}\left(\frac{\partial^2 q}{\partial Y_2^2}\right)_0 \sigma_2^2 + \left(\frac{\partial^2 q}{\partial Y_1 \partial Y_2}\right)_0 Cov(Y_1, Y_2)$$

$$\sigma_X^2 \approx \left(\frac{\partial q}{\partial Y_1}\right)_0^2 \sigma_1^2 + \left(\frac{\partial q}{\partial Y_2}\right)_0^2 \sigma_2^2 + 2\left(\frac{\partial q}{\partial Y_1}\right)_0 \left(\frac{\partial q}{\partial Y_2}\right)_0 Cov(Y_1, Y_2)$$

These are thus the mean and variance propagation laws for a function of two random variables. Pay attention to the $\approx$-sign.

If $Y_1$ and $Y_2$ are independent, we have that $Cov(Y_1, Y_2) = 0$, such that the last term in both equations disappears.

**Exercise mathematical pendulum**

We will measure the period on one oscillation $T$ of a pendulum, and also the length $L$ of the pendulum. Both measurements are affected by random errors, and therefore the 'calculated' gravitational acceleration $G$ is a function of two random variables:

$G = q(L, T) = 4\pi^2 \frac{L}{T^2}$

Apply the mean and variance propagation laws to approximate the mean and variance of $G$ given that the expectations $\mathbb{E}(L) = \mu_L$ and $\mathbb{E}(T) = \mu_T$, as well as the variances

$\sigma_L^2$ and $\sigma_T^2$ are known, and the covariance $Cov(L,T) = 0$.

**Solution**

First determine the first- and second-order partial derivatives of the function $q(L,T)$ to obtain:

$$\mathbb{E}(X) \approx 4\pi^2 \frac{\mu_L}{\mu_T^2} + 12\pi^2 \frac{\mu_L}{\mu_T^4}\sigma_T^2$$

$$\sigma_G^2 \approx 16\pi^4 \frac{1}{\mu_T^4}\sigma_L^2 + 64\pi^4 \frac{\mu_L^2}{\mu_T^6}\sigma_T^2$$

**Exercise moment of inertia**

A moment of inertia of a rectangle is given by \(I = \frac{1}{12}WH^3\) with \(W\) the width and \(H\) the height.
We have that \(\mu_W = 30 [cm]\), \(\mu_H = 2 [cm]\), \(\sigma_W^2=0.5 [cm^2]\), \(\sigma_H^2=0.2 [cm^2]\) and \(Cov(H,W)=0.2[cm^2]\).
*Give your answers with 1 decimal.*

\(\mu_I \approx \) and \(\sigma^2_I \approx \)

Check

# Functions of $n$ random variables

The propagation laws for functions of $n$ random variables are as follows:

$$\mathbb{E}(X) \approx q(\mu_Y) + \frac{1}{2}\sum_{i=1}^{n}\left(\frac{\partial^2 q}{\partial Y_i^2}\right)_0 \sigma_i^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\left(\frac{\partial^2 q}{\partial Y_i \partial Y_j}\right)_0 Cov(Y_i,Y_j)$$

$$\sigma_X^2 \approx \sum_{i=1}^{n}\left(\frac{\partial q}{\partial Y_i}\right)_0^2 \sigma_i^2 + \sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\left(\frac{\partial q}{\partial Y_i}\right)_0\left(\frac{\partial q}{\partial Y_j}\right)_0 Cov(Y_i,Y_j)$$

### Exercises wind load

We have the following equation, corresponding to the wind loading equation $F$,

$$F = A \cdot P \cdot C_d$$

Assume $A$, $P$, and $C_d$ are independent random variables.

What is the covariance matrix of the random vector $Y = [A, P, C_d]^T$?

**Solution**

$$\Sigma_Y = \begin{bmatrix} \sigma_A^2 & 0 & 0 \\ 0 & \sigma_P^2 & 0 \\ 0 & 0 & \sigma_{C_d}^2 \end{bmatrix}$$

What is the approximation of the mean $\mu_F$?

**Solution**

$$\mu_F \approx q(\mu_A, \mu_P, \mu_{C_d}) = \mu_A \mu_P \mu_{C_d}$$

(second-order partial derivatives are all 0, and also the covariances are all 0)

What is the approximation of $\sigma_F^2$?

**Solution**

$$\sigma_F^2 \approx (\mu_P \cdot \mu_{C_d} \cdot \sigma_A)^2 + (\mu_A \cdot \mu_{C_d} \cdot \sigma_P)^2 + (\mu_A \cdot \mu_P \cdot \sigma_{C_d})^2$$

### Exercise evaporation

The evaporation $Q$ is given as function of net radiation $R_n$, ground heat flux $G$ and Bowen ratio $B$:

$$Q = \frac{R_n - G}{1 - B}$$

Assume that the random variables $R_n$, $G$, $B$ are independent.

What is the approximation of $\sigma_Q^2$:

**Solution**

$$\sigma_Q^2 \approx \frac{1}{1 - 2\mu_B + \mu_B^2} \cdot (\sigma_{R_n}^2 + \sigma_G^2) + \left( \frac{\mu_{R_n} - \mu_G}{(1 - \mu_B)^2} \right)^2 \sigma_B^2$$

# Video

In this video the functions of $n$ random variables is discussed, as well as the linear propagation laws is explained.

1_3_3b_Video_ErrorPropagation_2

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Linear propagation laws of mean and covariance

## Contents


- Linear function of two random variables
- Linear functions of $n$ random variables


## Linear function of two random variables

Consider a linear function of two random variables

$$X = q(Y) = a_1 Y_1 + a_2 Y_2 + c$$

We can now show that $\mathbb{E}(q(Y)) = a_1 \mathbb{E}(Y_1) + a_2 \mathbb{E}(Y_2) + c$ using our Taylor approximations. The first-order partial derivatives namely follow as

$$\frac{\partial q}{\partial Y_1} = a_1, \ \frac{\partial q}{\partial Y_2} = a_2$$

and all the higher-order derivatives are zero, and consequently all higher-order terms in the Taylor series will be zero. The expectation of $q(Y)$ follows therefore as

$$\mathbb{E}(q(Y)) = q(\mu_1, \mu_2) = a_1 \mu_1 + a_2 \mu_2 + c$$

which is exact (i.e., not an approximation anymore).

**Exercise**

In a similar fashion derive the variance of $X$, which is also an exact result.

**Solution**

First determine the first- and second-order partial derivatives of the function $q(L, T)$ to obtain:

$$\sigma_X^2 = a_1^2\sigma_1^2 + a_2^2\sigma_2^2 + 2a_1a_2Cov(Y_1, Y_2)$$

Note that it does not depend on the deterministic constant $c$.

# Linear functions of $n$ random variables

Note that the linear function of two random variables can also be written as $q(Y) = \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} + c$. We will now generalize to the case where we have $m$ linear functions of $n$ variables, which can be written as a linear system of equations:

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} + \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \mathrm{A}Y + \mathrm{c}$$

with known $\mathbb{E}(Y)$ and covariance matrix $\Sigma_Y$, and $\mathrm{c}$ a vector with deterministic variables.

The linear propagation laws of the mean and covariance matrix are given by

$$\mathbb{E}(X) = \mathrm{A}\mathbb{E}(Y) + \mathrm{c}$$

$$\Sigma_X = \mathrm{A}\Sigma_Y\mathrm{A}^T$$

These are exact results, since for linear functions the higher-order terms of the Taylor approximation become zero and thus the approximation error is zero.

**Exercise**

Consider the linear system of equations

$$X = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$$

with

$$\mu_Y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \ \Sigma_Y = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix}$$

Apply the linear propagation laws to find $\mathbb{E}(X) = \mu_X$ and $\Sigma_X$.

**Solution**

$$\mu_X = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\Sigma_X = \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} 6 & -3 \\ -3 & 15 \end{bmatrix}$$

**Exercise buckets of concrete**

You want to measure out 1.5$l$ of concrete in a bucket, but only have a bucket of with lines indicating 5$l$, 2$l$, and 0.5$l$. These buckets are named $a$, $b$, and $c$, respectively. To achieve this you take the 5$l$ bucket, take out 2$l$ using bucket $b$, and then three times 0.5 $l$ using bucket $c$: $Y_1 = V_a - V_b - 3V_c$ (with $V_i$ is the volume of bucket $i$). However, you can't read the lines on the buckets perfectly, and the variance of your pouring skills is 1/100th of the volume of the bucket. Assume the volumes to be independent.

You do something similar to achieve 4.5$l$ ($Y_2 = V_a - V_c$) and 1$l$ ($Y_3 = V_a - 2V_b$). Compute the covariance matrix of $[Y_1 \ Y_2 \ Y_3]^T$.

Hint: first find the $\mathrm{A}$ matrix of the linear system $Y = \mathrm{A} \cdot V$.

**Solution**

$$\mathrm{A} = \begin{bmatrix} 1 & -1 & -3 \\ 1 & 0 & -1 \\ 1 & -2 & 0 \end{bmatrix}, \Sigma_V = \begin{bmatrix} \frac{5}{100} & 0 & 0 \\ 0 & \frac{2}{100} & 0 \\ 0 & 0 & \frac{0.5}{100} \end{bmatrix}$$

$$\begin{aligned} \Sigma_Y &= \mathrm{A}\Sigma_V\mathrm{A}^T \\ &= \begin{bmatrix} 1 & -1 & -3 \\ 1 & 0 & -1 \\ 1 & -2 & 0 \end{bmatrix} \begin{bmatrix} \frac{5}{100} & 0 & 0 \\ 0 & \frac{2}{100} & 0 \\ 0 & 0 & \frac{0.5}{100} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \\ -1 & 0 & -2 \\ -3 & -1 & 0 \end{bmatrix} = \frac{1}{100} \begin{bmatrix} 11.5 \\ 6.5 \\ 9 \end{bmatrix} \end{aligned}$$

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Observation theory

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

As in other physical sciences, empirical data are used in Civil and Environmental Engineering and Applied Earth Sciences to make inferences so as to describe physical reality. Many such problems involve the determination of unknown model parameters that bear a functional relationship to the set of observed data or measurements. This determination or parameter estimation can be based on different estimation principles.

# Introduction

## Contents


- Functional model: examples
- Redundancy
- Estimation and linear regression


From experience we know that various uncertain phenomena can be modeled as a random variable (or a random vector), say $Y$. In this part the random variables are measurements (e.g., sensor readings), which are uncertain due to random errors. We will refer to these random variables (our input data) $Y$ as the *observables*.

The unknown model parameters will be denoted as $\mathrm{x}$. The ingredients for this part on sensing and observation theory can then be summarized as follows:

- a model to describe the relation between observables $Y$ and parameters of interest $\mathrm{x}$
- estimation method(s) to estimate the parameters as function of $Y$: $\hat{X} = q(Y)$ (i.e., the estimation method defines the function $q$)
- uncertainty propagation to assess the precision of the estimated parameters $\hat{X}$
- hypothesis testing to account for errors in data and/or to choose best model from different candidates

**Definition**

The *functional model* describes the functional relationship between the observed variables (*observables*) and the unknown parameters.

In case the functional relationship is linear, the functional model will have the form:

$$\mathbb{E}(Y) = \mathrm{A}\mathrm{x} \quad \text{or} \quad Y = \mathrm{A}\mathrm{x} + \epsilon$$

where $\mathrm{x}$ is a vector with the unknown model parameters, and $\mathrm{A}$ is referred to as the *design matrix*. $\mathbb{E}(.)$ is the expectation operator. The random errors are given by $\epsilon$ and are assumed to have zero mean: $\mathbb{E}(\epsilon) = 0$.

A functional model can be mechanistic, phenomological or even data-driven, see the Model classification section.

**Exercise**

Show that the two expressions for the functional model are indeed equivalent.

**Solution**

$$\mathbb{E}(Y) = \mathbb{E}(\mathbf{A}\mathbf{x} + \epsilon) = \mathbf{A}\mathbf{x} + \mathbb{E}(\epsilon) = \mathbf{A}\mathbf{x}$$

Recall that the unknown parameters are deterministic.

Next, we will develop the principles and underlying theory of (1) Least-squares estimation, (2) Best linear unbiased estimation, and (3) Maximum likelihood estimation.

Special attention is then given to how the uncertainty in the measurements propagates into parameter estimation results, such that we can assess the precision of the estimated parameters. For that purpose we need the stochastic model.

**Definition**

The *stochastic model* describes the uncertainty of the observables in the form of the covariance matrix $\Sigma_Y = \Sigma_\epsilon$.

This covariance matrix is assumed to be known here. In practice, it can be determined based on a calibration campaign: taking repeated measurements and calculate the empirical (co-)variances. Note that the uncertainty in the observations is fully attributed to the random errors, therefore we have that the covariance matrix of the observables is equal to that of the random errors.

**Quiz question**

Which of the following statements is/are true?

- [ ] There is a 95% chance that random errors are within a range of $\pm 1\sigma_\epsilon$
- [ ] $\sigma_\epsilon$ is a measure for the accuracy
- [ ] $\sigma_y = \sigma_\epsilon$
- [ ] Precision is not affected by non-random systematic biases and outliers

Check

Parameter estimation requires specification of the underlying functional and stochastic models. It may happen, however, that some parts of the models are misspecified, thereby invalidating the results of estimation. Some measurements, for instance, may be corrupted by blunders (which are not random!), or the chosen model may fail to give an adequate description of physical reality. Testing for such misspecifications is the topic of the last sections of this chapter.

# Functional model: examples

## Linear trend model

The unknown parameters are the intercept $x_1$ and rate of change (velocity) $x_2$. The observation equation of a single observable $Y_i$ is:

$$\mathbb{E}(Y_i) = x_1 + x_2 t_i$$

where the observation times, or epochs, $t_i$ are assumed to be known and therefore deterministic.

The linear functional model for $m$ observables becomes:

$$\mathbb{E}\left(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}\right) = \underbrace{\begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathrm{x}}$$

Examples where as a first guess the linear trend model could be applied (but might be too simplistic): subsidence due to gas extraction, uplift due to water injection, location on a line of a target with constant velocity, sea level rise of last 10 years.

**Exercise**

What is the functional model if the observation equations are given by the quadratic function $\mathbb{E}(Y_i) = x_1 + x_2 t_i + x_3 t_i^2$. Is this a linear model?

**Solution**

$$\mathbb{E}\left(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}\right) = \underbrace{\begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \\ 1 & t_m & t_m^2 \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{\mathrm{x}}$$

This is model is linear in $\mathrm{x}$, therefore we refer to it as a linear model.

# Step function

Consider a process with unknown parameter $x_1$ assumed to be constant up till time $t_{i-1}$, and a sudden change (step) at time $t_i$, after which the parameter remains constant at $x_2$. See Fig. 3.1.

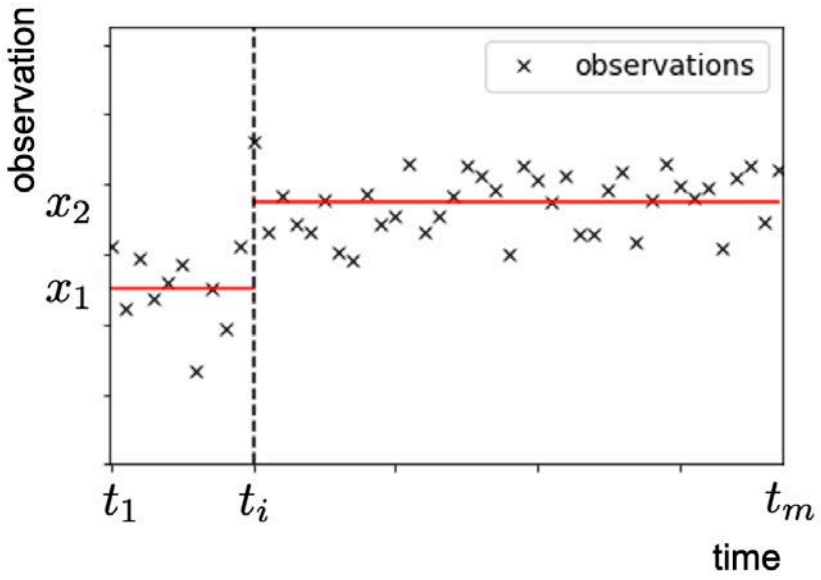



*Fig. 3.1* Observations with fitted step function.

The functional model can be written as:

$$\mathbb{E}\left(\begin{bmatrix} Y_1 \\ \vdots \\ Y_{i-1} \\ Y_i \\ \vdots \\ Y_m \end{bmatrix}\right) = \underbrace{\begin{bmatrix} 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 0 & 1 \\ \vdots & \vdots \\ 0 & 1 \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathrm{x}}$$

Such model may apply when considering deformation events in the subsurface, but can also be applicable in case of sensor replacement where one sensor has an offset with respect to the other (which would have to be calibrated).

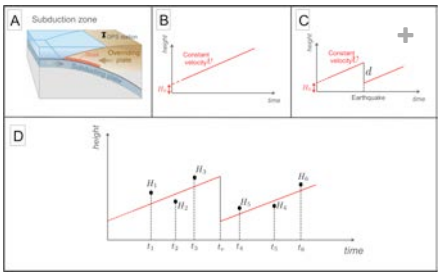
**Exercise**

The functional model of the step function can also be parameterized in terms of the initial value $x_1$ plus the step size $s = x_2 - x_1$. Give the corresponding functional model.

**Solution**

$$\mathbb{E}\left(\begin{bmatrix} Y_1 \\ \vdots \\ Y_{i-1} \\ Y_i \\ \vdots \\ Y_m \end{bmatrix}\right) = \underbrace{\begin{bmatrix} 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & 1 \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} x_1 \\ s \end{bmatrix}}_{\text{x}}$$

**Exercise**




**Subduction zones: what are the unknown parameters?**

In subduction zones, two tectonic plates collide, and the denser plate is forced down under (or subducts) the less dense one, as can be seen in Figure A. This subduction results in the uplift, usually of constant speed, of points on the overriding plate. Figure B shows the height change of one point.

Every now and then, the accumulated stress between plates is released, resulting in an earthquake and a relatively rapid deformation of the surface. Note that here the point is assumed to move with constant velocity, denoted by $v$, before and after the earthquake due to the subduction mechanism, and the earthquake event results in a jump in the time series, denoted by $d$ (Figure C). Using GPS we are monitoring the height of the GPS station in order to determine how that point moves, which is illustrated in Figure D (where $t_e$ is the time at which the earthquake takes place). What are the unknown parameters?

- [ ] GPS heights $H_i$
- [ ] Deformation $d$
- [ ] Various times $t_i$
- [ ] Velocity $v$
- [ ] Initial height $H_0$

Check

# Positioning model

As a final example we will consider a non-linear functional model for estimating the unknown position $\mathrm{x} = [x, y, z]^T$ of a satellite. The observables are distance measured with laser ranging from $m$ ground stations at known positions $[x_i, y_i, z_i]^T$.

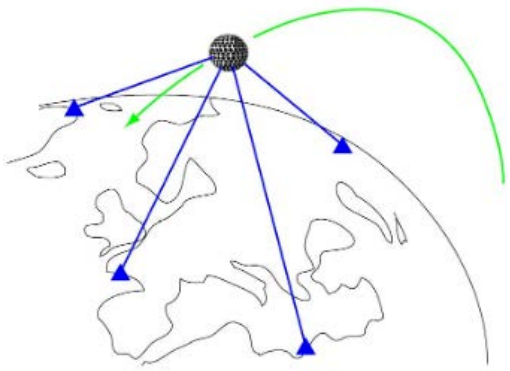


*Fig. 3.2* Positioning of satellite in orbit using laser ranging from multiple ground stations on knonw locations.

The functional model comprises $m$ non-linear functions of the unknown parameter vector $\mathrm{x}$:

$$\mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \begin{bmatrix} \sqrt{(x_1 - x)^2 + (y_1 - x)^2 + (z_1 - x)^2} \\ \sqrt{(x_2 - x)^2 + (y_2 - x)^2 + (z_2 - x)^2} \\ \vdots \\ \sqrt{(x_m - x)^2 + (y_m - x)^2 + (z_m - x)^2} \end{bmatrix} = \begin{bmatrix} q_1(\mathrm{x}) \\ q_2(\mathrm{x}) \\ \vdots \\ q_m(\mathrm{x}) \end{bmatrix}$$

# Redundancy

Later we will see that the *redundancy* of our model plays an important role regarding the precision of our estimated parameters. For a model with $m$ observables and $n$ unknown parameters, the redundancy is given by:

$$\text{redundancy} = m - n$$

this is true since we assume to work with $\mathrm{A}$-matrices (size $m \times n$) that have full column rank: $rank(\mathrm{A}) = n$.

# Estimation and linear regression

The goal of estimation is the estimate *model parameters* from a set of observations. In Civil Engineering, Applied Earth Sciences and Environmental Engineering this is needed in many monitoring and sensing applications, such as:

- Sea level rise

- Land subsidence / uplift
- Air quality
- Settlement of soils
- Tunnel deformation
- Bridge motions
- Traffic flow rates
- Water vapor content for numerical weather prediction
- Ground water level

In this part, we will introduce different estimation principles, starting with (weighted) least-squares. Next, Best Linear Unbiased estimation and Maximum Likelihood estimation will be introduced, providing the probabilistic framework for estimating *model* parameters from *uncertain data*.

*Linear regression* aims to estimate the relationship between so-called *dependent* and *independent* variables, the latter also referred to as *predictors*, since once the relationship is known, these can be used to predict the behaviour of the dependent variables. Examples of bivariate regression model applications:

- predict *river run-off* (dependent variable) based on *amount of rainfall* (independent variable), see Fig. 3.3
- predict *house prices* (dependent variable) based on their *energy label* (independent variable)
- predict *geothermal power production* (dependent variable) based on *brine temperature* (independent variable)
- predict *structural deformation* (dependent variable) based on *wind load* (independent variable)

Note that regression analysis is not restricted to linear nor to bivariate cases.

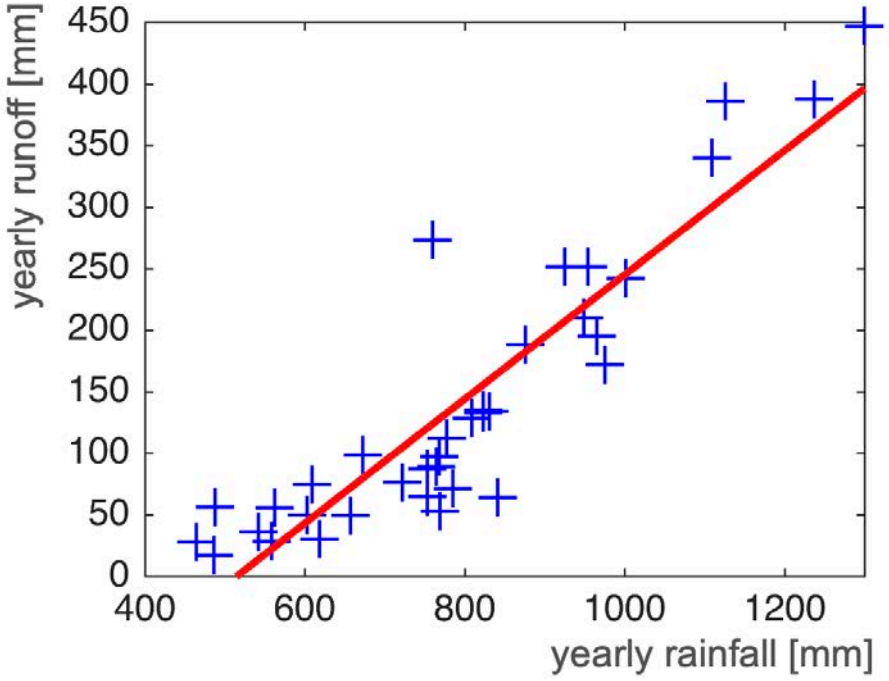



*Fig. 3.3* Linear regression example for amount of rainfall (independent variable) and river runoff (dependent variable).

The estimation principles discussed in this part are needed for estimating these relationships.

Supervised machine learning (MUDE topic in Q2) is all about finding relationships between target (dependent) variables and certain features (predictors), and therefore regression analysis.

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Least-squares estimation

## Contents


- Functional model: dealing with inconsistency
- Least-squares principle
- Least-squares solution
- Summary


## Functional model: dealing with inconsistency

Given a set of observations which contain noise, and a model that is assumed to explain these data, the goal is to estimate the unknown parameters of that model. The least-squares principle can be used to solve this task.

Mathematically, we use the 'system of observation equations', with the $m$-vector of observations $\mathrm{y}$, the $n$-vector of unknowns $\mathrm{x}$, and the $m \times n$ design matrix $\mathrm{A}$. In a simple form, we can write this system of observation equations as

$$\mathrm{y} = \mathrm{A}\mathrm{x}.$$

If all the observations would fit perfectly to this model, this system is called consistent. This is only possible if the number of observations $m$ is equal to (or smaller than) the number of unknowns $n$, i.e. if the system is determined.

If the number of observations is greater than the number of unknowns (and the design matrix $\mathrm{A}$ is of full column rank), it is very unlikely that the system is consistent. Physical reality would need to be perfectly described by the conceptual mathematical model. It is evident that in real life this is never the case, since (i) our observations are always contaminated by some form of noise, and (ii) physical reality is often more complex than can be described with a simplified mathematical model.

Thus, in the case in which there are more observations than unknowns (and design matrix $\mathrm{A}$ is of full column rank) the $\mathrm{y} = \mathrm{Ax}$ system of equations has no solution, this is referred to as an overdetermined system. In other words; every 'solution' would be wrong, since the model would not perfectly 'fit' the data.

The *redundancy* of the system is equal to $m - n$, i.e., the 'additional' number of observations compared to the minimum required to solve the system of equations.

# Example: linear trend model

Assume we have $m$ observations and we try to fit the linear trend model:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}}_{\mathrm{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{\mathrm{x}}$$

Fig. 3.4 shows the observations, together with a fitted linear trend line. Obviously, it is impossible to fit a line that perfectly matches all observations due the small measurement errors, which are assumed to be random.

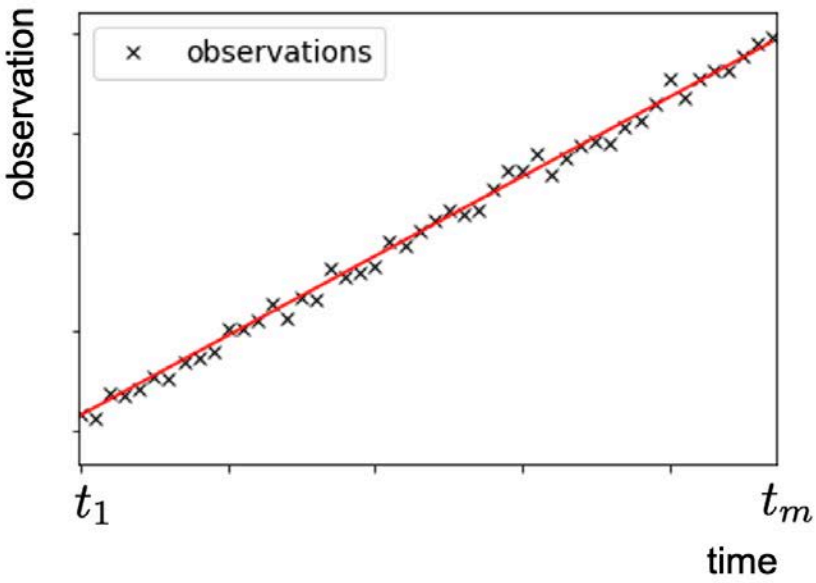



*Fig. 3.4* Linear trend line fitted to a set of $m$ observations affected by random errors.

We solve the problem of having an 'unsolvable' equation by considering the random errors for each observation:

$$\mathrm{y} = \mathrm{A}\mathrm{x} + \epsilon$$

The length of the error vector (or vector of residuals) $\epsilon$ is equal to the length of the vector of observations $\mathrm{y}$.

# Least-squares principle

We are looking for a solution for $\mathrm{x}$; this solution will be denoted by $\hat{\mathrm{x}}$. Based on this solution, the 'adjusted observations' would be $\hat{\mathrm{y}} = \mathrm{A}\hat{\mathrm{x}}$ (solution of the forward model).

To find a solution for an inconsistent system of equations, we prefer the one for which the observations are 'as close as possible' to the model. This is a 'minimum distance' objective. In mathematical terms, we look at the vector of residuals $\hat{\epsilon}$: the difference between the observations $\mathrm{y}$ and the model realization $\hat{\mathrm{y}}$:

$$\hat{\epsilon} = \mathrm{y} - \hat{\mathrm{y}} = \mathrm{y} - \mathrm{A}\hat{\mathrm{x}}$$

We want this vector of residuals to be as small as possible (i.e., the minimum distance objective). The 'least-squares' principle achieves this by minimizing ('least') the sum of the squared residuals ('squares').

If we take the square-root of this sum, we find the length of the vector, also known as the *norm* of the vector. Thus, the various possible error vectors have a norm defined as:

$$|| \epsilon || = \sqrt{\epsilon_1^2 + \epsilon_2^2 + \ldots + \epsilon_m^2} = \sqrt{\epsilon^T \epsilon}$$

Finding the minimum of (a) the sum of the squared differences, or (b) the square-root of the sum of the squared differences, yields the same result, since both terms are always positive or zero.

If we find this minimum, the corresponding solution $\hat{\mathrm{x}}$ is the least-squares solution of the system of observation equations. In mathematical terms this is written as

$$\hat{\mathrm{x}} = \arg\min_{\mathrm{x}} || \epsilon ||^2 = \arg\min_{x} \left(\epsilon^T \epsilon\right)$$

From the system observation equations $\mathrm{y} = \mathrm{A}\mathrm{x} + \epsilon$, it follows directly that $\epsilon = \mathrm{y} - \mathrm{A}\mathrm{x}$ and therefore the least-squares solution follows as:

$$\hat{\mathrm{x}} = \arg\min_{\mathrm{x}} (\mathrm{y} - \mathrm{Ax})^{\mathrm{T}} (\mathrm{y} - \mathrm{Ax}).$$

In other words, $\hat{\mathrm{x}}$ is the solution for $\mathrm{x}$ which provides the minimum of $(\mathrm{y} - \mathrm{Ax})^{\mathrm{T}}(\mathrm{y} - \mathrm{Ax}) = \sum_{i=1}^{m} \epsilon_i^2$, hence the smallest sum of squared residuals.

# Least-squares solution

We can find the minimum of a function by taking the first derivative with respect to the 'free variable'. Since the observation vector is not free, and also the design matrix $\mathrm{A}$ is fixed, the only variable which we can vary is $\mathrm{x}$. The first derivative of our objective function should be equal to zero to reach a minimum (see Gradient, and Vectors and matrices):

$$\begin{aligned}
\partial_x (\mathrm{y} - \mathrm{Ax})^{\mathrm{T}} (\mathrm{y} - \mathrm{Ax}) &= 0 \\
\partial_x \left( \mathrm{y}^{\mathrm{T}}\mathrm{y} - (\mathrm{Ax})^{\mathrm{T}}\mathrm{y} - \mathrm{y}^{\mathrm{T}}\mathrm{Ax} + (\mathrm{Ax})^{\mathrm{T}}(\mathrm{Ax}) \right) &= 0 \\
\partial_x (\mathrm{y}^{\mathrm{T}}\mathrm{y} - \mathrm{x}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{y} - \mathrm{y}^{\mathrm{T}}\mathrm{Ax} + \mathrm{x}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{Ax}) &= 0 \\
\partial_x (\mathrm{y}^{\mathrm{T}}\mathrm{y} - 2\mathrm{x}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{y} + \mathrm{x}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}\mathrm{Ax}) &= 0 \\
-2\mathrm{A}^{\mathrm{T}}\mathrm{y} + 2\mathrm{A}^{\mathrm{T}}\mathrm{Ax} &= 0 \\
\mathrm{A}^{\mathrm{T}}\mathrm{Ax} &= \mathrm{A}^{\mathrm{T}}\mathrm{y}
\end{aligned}$$

This last equation is known as the normal equation, and the matrix $\mathrm{N} = \mathrm{A}^{\mathrm{T}}\mathrm{A}$ is known as the normal matrix.

**MUDE exam information**

You will not be asked to derive the least-squares solution as above.

If it is possible to compute the inverse of the normal matrix, the normal equation can be written to express the estimate of $\mathrm{x}$, denoted by $\hat{\mathrm{x}}$, as a linear function of the observations:

$$\hat{\mathrm{x}} = (\mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{y}$$

# Summary

In summary, the least-squares estimates of $\mathrm{x}$, $\mathrm{y}$ and $\epsilon$ are given by:

$$\hat{\mathrm{x}} = (\mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{y}$$

$$\hat{\mathrm{y}} = \mathrm{A}\hat{\mathrm{x}}$$

$$\hat{\epsilon} = \mathrm{y} - \mathrm{A}\hat{\mathrm{x}} = \mathrm{y} - \hat{\mathrm{y}}$$

Fig. 3.5 visualizes the least-squares solution based on a linear trend model.

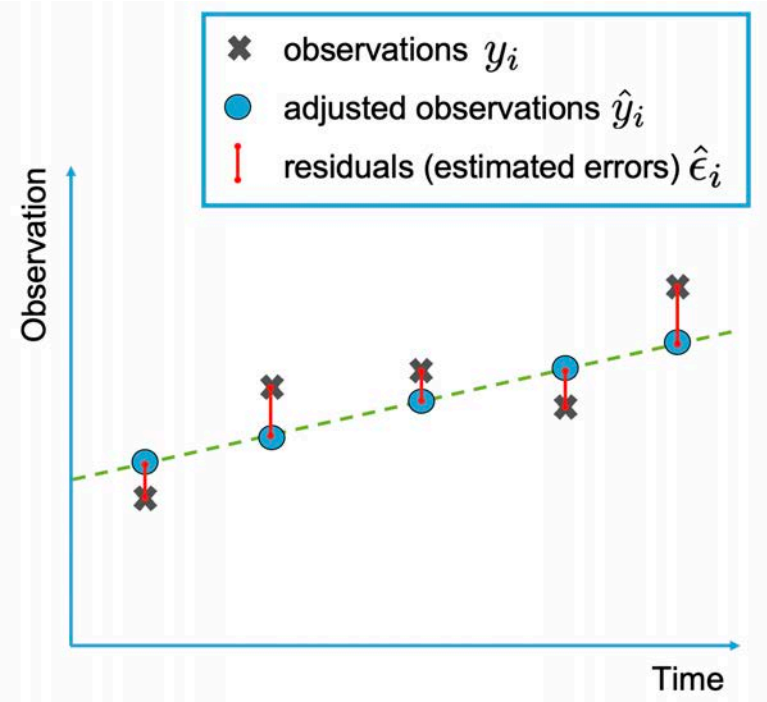



*Fig. 3.5* Least-squares fit to a set of 5 observations affected by random errors. Index $i$ refers to the $i$th observation.

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Notebook exercise: fitting different models

In this interactive notebook you will fit several models to a time series of height observations of a point on a glacier, to assess whether it is melting.

Click 🚀 –> **Live Code** on the top right corner of this screen and then wait until all cells are executed.

**Learning objectives:**

- set-up an observation model
- apply least-squares estimation
- assess the estimation results considering redundancy, squared norm of residuals, under- and overfitting

You have `m` = 12 monthly measurements of the height of a point on a glacier. The measurements are obtained from a satellite laser altimeter.

- Time [months]: `t` $= [0, 1, 2, \ldots, 11]$
- Observed heights [meters]: `y` $= [102.4, 98.2, 97.5, 97.9, 99.7, 100.7, 98.3, 94.2, 90.9, 86.1, 81.2, 76.9]$

`t` and `y` and `m` are already defined, so you can directly use these variables in your code.

We will consider three different models, with the following observation equations:

- Model 1: constant velocity, $\mathbb{E}\left(Y_i\right) = x_0 + x_1 t_i$
- Model 2: 8th order polynomial, $\mathbb{E}\left(Y_i\right) = x_0 + x_1 t_i + x_2 t_i^2 + \ldots + x_8 t_i^8 = \sum_{p=0}^{8} x_p t_i^p$
- Model 3: constant velocity + annual signal, $\mathbb{E}\left(Y_i\right) = x_0 + x_1 t_i + x_2 \cos\left(\frac{2\pi t_i}{12}\right)$

**Exercise 1**

With `column_stack` you can combine arrays (column vectors) to create a matrix. The design matrices for the first two models are already given. Add the matrix for the third model yourself, and check your answer below.

```
A_1 = np.column_stack((np.ones(m), t))
A_2 = np.column_stack((A_1, t**2, t**3, t**4, t**5, t**6, t**7, t**8))
A_3 = ?
```

### Exercise 2

You can use the NumPy function `np.linalg.inv` to compute the inverse of a matrix. Recall that for a matrix product $\mathrm{A} \cdot \mathrm{B}$ you can use `A @ B`. Complete the function below. The transpose $\mathrm{A}^T$ is obtained with `A.T`.

First complete the function `lsqe`, and check whether it is correct below.

```
def lsqe(y,A):
    '''Apply least-squares estimation
    Input:
    y : vector with observations
    A : design matrix
    '''
    # estimated parameters
    xhat = ?
    # adjusted observations
    yhat = A @ xhat
    # residuals
    ehat = ?
    # squared norm of residuals
    eTe = ehat.T @ ehat

    return xhat, yhat, ehat, eTe

xhat_1, yhat_1, ehat_1, eTe_1 = lsqe(y,A_1)
xhat_2, yhat_2, ehat_2, eTe_2 = lsqe(y,A_2)
xhat_3, yhat_3, ehat_3, eTe_3 = lsqe(y,A_3)

print(f'Redundancy of linear trend model: '
      f'{m - np.shape(A_1)[1]}')
print(f'Redundancy of 8th order polynomial model: '
      f'{m - np.shape(A_2)[1]}')
print(f'Redundancy of linear trend + annual signal model: '
      f'{m - np.shape(A_3)[1]}')
```

**Check whether `xhat_1` is correct (if not, you need to correct your `lsqe` function).**

**Check whether `eTe_1` is correct (if not, you need to correct your `lsqe` function).**

**Solution**

In case your code did not work, here's the correct solution:

```
xhat = np.linalg.inv(A.T @ A) @ A.T @ y

ehat = y-yhat
```

**Running the next cell will plot the observations and the three fitted models in one figure, and the residuals for each model in another figure.** Note that the code with the function `plot_results` is hidden.

```
plot_results()
```

### Wrap-up exercises

One of the models is *underfitting*. Try to reason what is meant by this and which model this applies to.

**Solution**

Model 1 obviously does not capture the annual signal, so it is does not accurately describe the relation between observations and height change. This is referred to as underfitting.

One of the models is *overfitting*. Try to reason what is meant by this and which model this applies to.

**Solution**

Model 2 is overfitting - it results in very small residuals since we include a lot of parameters, but what is the physical interpretation? Including higher orders will result in an even better fit (ultimately the perfect fit, see below). A big problem is that most likely this model will not be able to predict the height change for the future. This is illustrated in the figure below, where we predict the height for the coming 2 months, and see how the polynomial model predicts a very unrealistic increase in height.

What would happen if you would fit a 11th order polynomial?

**Solution**

This would result in a perfect fit: $\mathrm{A}$ becomes a full-rank square matrix, such that $\hat{\mathrm{x}} = \mathrm{A}^{-1}\mathrm{y}$ and the residuals become 0.

Do the residuals (see plot) behave as expected for each of the models.

**Solution**

No, you would expect them to fluctuate randomly around zero, but especially for model 1 you can see a clear residual signal, which is an indication that the model is too simplistic.

Discuss why the squared norm of residuals is quite different for the 3 models.

**Solution**

The squared norm for model 1 is very large, due to the annual signal that is not accounted for. The squared norm for model 2 is very small, so it results in a very good fit (due to the high order of the polynomial). Model 3 results in a somewhat larger squared norm than model 2 - this is due to the additional parameters in model 2, which will result in a better fit (i.e., it allows to better capture the actual height variation over time).

How could we quantitatvely assess whether model 3 is a good choice (this is a topic we will discuss later, but try to think for yourself how it could be done).

**Solution**

You would somehow have to compare the size of the residuals with the precision as described by $\Sigma_\epsilon$.

**Running the next cell will plot the fitted models including predictions for the next two months.** Note that the code with the function `plot_prediction` is hidden.

```
plot_prediction()
```

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Weighted least-squares estimation

## Contents


- Video
- Discussion on the weight matrix
- Weighted least-squares estimator: properties


In ordinary least-squares estimation, we assume that all observations are equally important. In many cases this is not realistic, as observations may be obtained by different measurement systems, or under different circumstances. We want our methodology for least-squares estimation to be able to take this into account.

We achieve this goal by introducing a weight matrix in the minimization problem:

$$\min_{\mathrm{x}} (\mathrm{y} - \mathrm{Ax})^{\mathrm{T}} \mathrm{W} (\mathrm{y} - \mathrm{Ax})$$

In the unweighted least-squares approach, we arrive at the normal equation by pre-multiplying both sides of $\mathrm{y} = \mathrm{Ax}$ with the transpose of the design matrix $\mathrm{A}^{\mathrm{T}}$:

$$\mathrm{y} = \mathrm{Ax} \;\rightarrow\; \mathrm{A}^{\mathrm{T}}\,\mathrm{y} = \mathrm{A}^{\mathrm{T}}\,\mathrm{Ax}$$

In the weighted least-squares approach, we now need to add weight matrix $\mathrm{W}$ to this pre-multiplication factor, i.e., $\mathrm{A}^{\mathrm{T}}\mathrm{W}$, to obtain the normal equation:

$$\mathrm{y} = \mathrm{Ax} \;\rightarrow\; \mathrm{A}^{\mathrm{T}}\mathrm{W}\,\mathrm{y} = \mathrm{A}^{\mathrm{T}}\mathrm{W}\,\mathrm{Ax}$$

The normal matrix is now defined as $\mathrm{N} = \mathrm{A}^{\mathrm{T}}\mathrm{WA}$. From this, assuming that the normal matrix $\mathrm{N}$ is invertible (non-singular) we find the weighted least-squares estimate $\hat{\mathrm{x}}$,

$$\begin{aligned} \hat{\mathrm{x}} &= (\mathrm{A}^{\mathrm{T}}\mathrm{WA})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{Wy} \\ &= \arg\min_{\mathrm{x}} (\mathrm{y} - \mathrm{Ax})^{\mathrm{T}} \mathrm{W} (\mathrm{y} - \mathrm{Ax}) \end{aligned}$$

To see the derivation, watch the video, though you will not be asked to do this on the exam.

We also find the derived estimate $\hat{\mathrm{y}}$ and $\hat{\epsilon}$:

$$\hat{\mathrm{y}} = \mathrm{A}\hat{\mathrm{x}} = \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{y}$$

$$\hat{\epsilon} = \mathrm{y} - \hat{\mathrm{y}} = \mathrm{y} - \mathrm{A}\hat{\mathrm{x}} = \mathrm{y} - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{y} = (\mathrm{I} - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W})\mathrm{y}$$

**Exercises**

Select the correct statement(s).

- [ ] Ordinary least-squares estimation is also a form of weighted least-squares.
- [ ] If we apply weighted least-squares, we 'expect' the residuals of observations with smaller weight to be larger.
- [ ] The weighted least-squares solution provides the smallest possible \ (\hat{\epsilon}^T\hat{\epsilon}\).

Check

You and your fellow students are asked to measure the width of a canal. You all decide to take a different approach. Alice is asked to come up with an estimate of the canal width using all \(m\) observations. Based on her own 'ranking', Alice decides to give different weights to the different observations. Which of the following is the equation that Alice will use to estimate the canal width? The individual weights are called \(w_i\), \( (i=1,..., m) \)

- ○ \( \hat{x} = \frac{1}{\sum_{i=1}^{m}w_i}\sum_{i=1}^{m}(w_i y_i) \)
- ○ \( \hat{x} = \sum_{i=1}^{m} \frac{y_i}{w_i} \)
- ○ \( \hat{x} = \frac{1}{m}\sum_{i=1}^{m}y_i \)
- ○ \( \hat{x} = \frac{1}{\sum_{i=1}^{m}w_i}\sum_{i=1}^{m}y_i \)

Check

# Video

**MUDE exam information**

You will not be asked to derive the weighted least-squares solution as shown in the video.

# Discussion on the weight matrix

The weight matrix $\mathrm{W}$ expresses the (relative) weights between the observations. It is always a symmetric square matrix. The size of the weight matrix depends on the number of observations, $m$. The size of the weight matrix is $m \times m$.

If it is a unit matrix ($\mathrm{W} = \mathrm{I}$), this implies that all observations have equal weight. Note that in this case the equations are equal to the ordinary least-squares solution.

If it is a diagonal matrix, with different values on the diagonal, this implies that entries with a higher value correspond to the observations which are considered to be of more importance. If the weight matrix has non-zero elements on the off-diagonal positions, this implies that (some) observations are correlated.

**Exercise**

We have a set of 3 observations, which are correlated, where the weight of the third observation is four times higher than the weight of the second observation. Which weight matrix would match with this description?

- ○ \( \begin{bmatrix} 2 & 0 & 0\\0 & 1 & 0\\0 & 0 & 4\end{bmatrix} \)
- ○ \( \begin{bmatrix} 3 & 0 & 0\\0 & 2 & 0\\0 & 0 & 8\end{bmatrix} \)
- ○ \( \begin{bmatrix} 2 & 0 & 1\\0 & 1 & 1\\1 & 1 & 2\end{bmatrix} \)
- ○ \( \begin{bmatrix} 3 & 2 & 1\\2 & 2 & 1\\1 & 1 & 8\end{bmatrix} \)

Check

# Weighted least-squares estimator: properties

Until now, we have looked at the weighted least-squares solution of a single *realization* of the observations, where generally we assume that it is a realization of the *random* observable vector $Y$, since measurements are affected by random errors. As such it follows the the weighted least-squares *estimator* is given by:

$$\hat{X} = (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}Y$$

This estimator has two important properties: it is *linear* and *unbiased*.

The linearity property is due to the fact that $\hat{X}$ is a linear function of the observables $Y$.

The unbiased property means that the expectation of $\hat{X}$ is equal to the true (but unknown) $\mathrm{x}$. This can be shown as follows:

$$\mathbb{E}(\hat{X}) = (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathbb{E}(Y) = (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A}\mathrm{x} = \mathrm{x}$$

This a very desirable property. It applies that if we would repeat the measurements many times to obtain a new estimate, the *average of the estimated* values would be equal to the true values.

**Exercise**

What is the covariance matrix of the weighted least-squares estimator? You can assume the covariance matrix $\Sigma_Y$ of the observables to be given. Hence, you need to apply the (co)variance propagation law:

If $\hat{X} = \mathrm{L}^T Y$, then $\Sigma_{\hat{X}} = \mathrm{L}^T \Sigma_Y \mathrm{L}$

**Solution**

$$\Sigma_{\hat{X}} = \left(\mathrm{A}^T \mathrm{W} \mathrm{A}\right)^{-1} \mathrm{A}^T \mathrm{W} \Sigma_Y \mathrm{W} \mathrm{A} \left(\mathrm{A}^T \mathrm{W} \mathrm{A}\right)^{-1}$$

On the exam you might be asked for deriving this (i.e., not as multiple choice).

We collected 3 distance measurements to an object moving along a line at constant speed at \(t_i=i\), \(i=0,1,2\). The unknown initial position on the line is \(x_0\), and the unknown speed is \(v\).

The observations are given by \(\mathrm{y}=[1.10~~3.20~~6.03]^T\).

We apply weighted least-squares. The first 2 observations are given equal weight, the last observation is given twice the weight of the other observations.

Which of the following expressions is correct?

- ○ \(\hat{y}_i = 0.95 + 2.5  t_i \)
- ○ \(\hat{y}_i = 0.97 + 2.5 t_i \)
- ○ \(\hat{y}_i = 1.0 + 2.4  t_i \)

Check

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Notebook exercise: playing with the weights

In this interactive notebook you will fit several models to a time series of height observations of a point on a glacier, to assess whether it is melting.

Click 🚀 –> **Live Code** on the top right corner of this screen and then wait until all cells are executed.

**Learning objectives:**

- apply weighted least-squares
- discuss the impact of choosing different weight matrices

You have a time series of 8 measured heights and fit a model assuming a linear trend (constant velocity).

Times of observations, observed values and number of observations are given as the variables `t`, `y` and `m`.

`numpy` and `matplotlib.pyplot` are abbreviated as `np` and `plt`, respectively.

Fill in the correct $\mathrm{A}$-matrix.

```
A = ?
```

**Solution**

In case you did not get the right solution, use:

```
A = np.column_stack((np.ones(m), t))
```

Case 1: Define the weight matrix for $\mathrm{W} = I_m$

```
W_1 = ?
```

Case 2: Define the weight matrix with the weight of first 3 observations is five times as large as the rest

```
W_2 = ?
```

Case 3: Define the weight matrix with the weight of 4th and 5th observation is five times as large as the rest

```
W_3 = ?
```

**Solution**

In case you did not get the right solution, you can use:

```
W_1 = np.eye(m)

w=5

W_2 = np.eye(m)

W_2[0,0] = w

W_2[1,1] = w

W_2[2,2] = w

W_3 = np.eye(m)

W_3[3,3] = w

W_3[4,4] = w
```

Compute the weighted least-squares estimate for all three cases.

```
# compute weighted least squares estimates for all cases
xhat_1 = ?
xhat_2 = ?
xhat_3 = ?

print('Estimated parameters:')
print('Case 1:\t', xhat_1)
print('Case 2:\t', xhat_2)
print('Case 3:\t', xhat_3)
```

**Solution**

In case you did not get the right solution, you can use:

```
xhat_1 = np.linalg.solve(A.T @ W_1 @ A, A.T @ W_1 @ y)
```

Similarly for case 2 and 3.

**Running the next cell will plot the fitted models.** Note that the code with the function `plot_results` is hidden.

```
plot_results()
```

**Explain the different results, in particular the 'locations' of the different lines with respect to each other.**

**Solution**

The ordinary least-squares fit ($W = I_m$) is 'in the middle' as expected, since all observations get equal weight. For case 2, the first 3 observations get a larger weight, so the line is pulled 'down' to result in smaller errors for those 3 observations. For case 3, the opposite effect occurs, since the two observations that get the largest weight are also 'above' the red line.

**You can play with the weight factor of case 2 and 3 to see how the impact changes, depending on its size.**

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Best linear unbiased estimation

## Contents


- Video
- BLUE decomposed
- Covariance matrices of the BLU estimators
- Additional note on the linearity condition of BLUE


Best Linear Unbiased estimation (BLUE) is based on a fundamentally different principle than (weighted) least-squares (WLS), but still it turns out to be a special case of WLS.

Comparing the weighted least-squares estimator:

$$\hat{X}_{\text{WLS}} = (\mathrm{A}^{\mathrm{T}}\mathrm{W}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\mathrm{W}Y$$

with the estimator of the vector of unknowns following from BLUE:

$$\hat{X}_{\text{BLU}} = (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}Y$$

We see that the weight matrix $\mathrm{W}$ is now replaced by the inverse of the covariance matrix $\Sigma_Y$ (our stochastic model).

This makes sense intuitively: suppose the observables are all independent and thus uncorrelated, such that the covariance matrix is a diagonal matrix with the variances as its diagonal element. The weights for the individual observables are then equal to the inverse of their variances, which implies that a more precise observable (smaller variance) receives a larger weight.

Taking this particular weight matrix, i.e., $\mathrm{W} = \Sigma_Y^{-1}$, has a special meaning. It has the “Best” property. This means that using this particular weight matrix, we obtain a linear unbiased estimator which has minimal variance. In other words, with this particular weight matrix we

get the best possible estimator among all linear unbiased estimators, where ‘best’ represents optimal precision or minimal variance.

Given the BLU-estimator for $\hat{X}$, we can also find the BLU-estimators for $\hat{Y} = \mathrm{A}\hat{X}$,and for $\hat{\epsilon} = Y - \hat{Y}$,

$$\hat{Y} = \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}Y$$

$$\hat{\epsilon} = Y - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}Y$$

**Exercise stochastic model**

Which of the following statements are true?

- [ ] The covariance of two observables does not depend on the precision of those observables
- [ ] Two correlated observations carry less information than two uncorrelated observations (assume that the two observations have same precision in both cases).
- [ ] The functional model contains information about the uncertainty of the measurements
- [ ] The stochastic model is given by the covariance matrix, which is an \(n\times n\) square matrix, with \(n\) the number of unknowns.

**Exercise least-squares versus BLUE**

Let’s consider an estimation problem with $m$ observables and functional model $\mathbb{E}(Y) = \mathrm{Ax}$ and stochastic model $\Sigma_Y = \sigma^2 I_m$ (i.e., the covariance matrix is a scaled identity matrix).

Find the expressions for the ordinary (unweighted) least-squares estimator and the best linear unbiased estimator.

**True or False:** The ordinary least-squares and best linear unbiased estimator are equal to each other if $\Sigma_Y$ is a scaled identity matrix?

**Solution**

True:

$$\hat{X}_{\text{BLU}} = (\mathrm{A}^{\mathrm{T}}\frac{1}{\sigma^2}I_m\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\frac{1}{\sigma^2}I_mY = (\mathrm{A}^{\mathrm{T}}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}Y = \hat{X}_{\text{LS}}$$

# Video

1_6_1c Video BLUE

# BLUE decomposed

In BLUE, Best Linear Unbiased Estimation, the parts of the acronym ‘B’, ‘L’, and ‘U’ refer to specific properties.

*Linear* means that there is a linear (matrix) relation between the variables. Such linear relations imply that if we have a normally distributed vector $Y \sim N(\mathrm{Ax}, \Sigma_Y)$, which is multiplied with a matrix $\mathrm{L}$, this product will be normally distributed as well:

$$\hat{X} = \mathrm{L}^{\mathrm{T}}Y \sim N(\mathrm{L}^{\mathrm{T}}\mathrm{Ax}, \mathrm{L}^{\mathrm{T}}\Sigma_Y\mathrm{L})$$

where we use the linear propagation laws of the mean and covariance.

*Unbiased* means that the expected value of an estimator of a parameter is equal to the value of that parameter. In other words:

$$\mathbb{E}(\hat{X}) = \mathrm{x}.$$

The unbiased property also implies that:

$$\begin{aligned}\mathbb{E}(\hat{Y}) &= \mathrm{A}\mathbb{E}(\hat{X}) = \mathrm{Ax} \\ \mathbb{E}(\hat{\epsilon}) &= \mathbb{E}(Y) - \mathbb{E}(\hat{Y}) = \mathrm{Ax} - \mathrm{Ax} = 0\end{aligned}$$

*Best* means that the estimator has minimum variance (best precision), when compared to all other possible linear estimators:

$$\text{trace}(\Sigma_{\hat{X}}) = \sum_{i=1}^{m} \sigma^2_{\hat{X}_i} = \text{minimum}$$

(Note that this is equivalent to “minimum mean squared errors” $\mathbb{E}(\|\hat{X} - \mathrm{x}\|^2)$. )

# Covariance matrices of the BLU estimators

The precision of the estimator is expressed by its covariance matrix. For the ‘best linear unbiased’ estimator of $\hat{X}$, $\hat{Y}$ and $\hat{\epsilon}$ we obtain (by applying the linear covariance propagation laws):

$$\begin{aligned}\Sigma_{\hat{X}} &= \mathrm{L}^{\mathrm{T}}\Sigma_Y\mathrm{L} \\ &= (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1} \cdot \Sigma_Y \cdot \Sigma_Y^{-1}\mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1} \\ &= (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1} \\ &= (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\end{aligned}$$

$$\begin{aligned}\Sigma_{\hat{Y}} &= \mathrm{A}\Sigma_{\hat{X}}\mathrm{A}^{\mathrm{T}} \\ &= \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\end{aligned}$$

$$\begin{aligned}\Sigma_{\hat{\epsilon}} &= (\mathrm{I}_m - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}) \cdot \Sigma_Y \cdot (\mathrm{I}_m - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1})^T \\ &= (\Sigma_Y - \mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}})(\mathrm{I}_m - \Sigma_Y^{-1}\mathrm{A}(\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}) \\ &= \Sigma_Y - \Sigma_{\hat{Y}} - \Sigma_{\hat{Y}} + \Sigma_{\hat{Y}} \\ &= \Sigma_Y - \Sigma_{\hat{Y}}\end{aligned}$$

In summary, by applying the BLUE method, we obtain the best estimator among all linear unbiased estimators, where ‘best’ is quantitatively expressed via the covariance matrix.

# Additional note on the linearity condition of BLUE

The BLUE estimator is the best (or minimum variance) among all linear unbiased estimators. However, if we drop the condition of linearity, then BLUE is not necessarily the best. It means that there may be non-linear estimators that have even better precision than the BLUE. However, it can be proven that, in the case of normally distributed observations, the BLUE is also the best among all possible unbiased estimators. So we can say: for observations $Y$ that are normally distributed, the BLUE is BUE (best unbiased estimator).

**Exercises**

**Rate of Change in height of glacier**

The height of a point on a glacier has been measured at times $t_0 = 0, \ t_1 = 2 \ \text{and} \ t_2 = 4$ months. The first 2 observations have a precision of 1 cm; unfortunately then the instrument broke down and had to be replaced with a cheaper one, which had a precision of $\sqrt 2$ cm.

The observations are given by $y_0 = 202.5, \ y_1 = 200.5, \ y_2 = 206.5$ cm.

We are only interested in the rate of change with respect to time $t_0$, where it is assumed that the point is moving downwards due to glacier melt at a constant velocity. Therefore the functional model is set up using the *differential* observables:

$Y_{01} = Y_1 - Y_0$

$Y_{02} = Y_2 - Y_0$

The observation equations of the original height observation are given as $y_i = h_0 + v \cdot t_i$, for $i = 1,2,3$ with $h_0$ the initial height and $v$ the velocity. The functional and stochastic model for these observables are respectively:

$$ \mathbb{E}(Y) = \mathbb{E} (\begin{bmatrix}Y_0\\Y_1\\Y_2\end{bmatrix})=\begin{bmatrix}1 & t_0\\1& t_1\\1& t_2\end{bmatrix} \begin{bmatrix}h_0 \\v\end{bmatrix}=\begin{bmatrix}1 & 0\\1 & 2\\1 & 4\end{bmatrix} \begin{bmatrix}h_0 \\ v\end{bmatrix} $$

$$ \mathbb{D}(Y) = \Sigma_Y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2\end{bmatrix} $$

Now we define the differential observables:

$$Y_{\mathrm{d}}= \begin{bmatrix} Y_{01} \\ Y_{02}\end{bmatrix} = \begin{bmatrix} Y_1 -Y_0 \\ Y_2 - Y_0 \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2\end{bmatrix} = \mathrm{T} \cdot Y$$

Note that $ \mathbb{E}(Y_{0i}) = \mathbb{E}(Y_{i} - Y_{0}) = (h_0 + v \cdot t_i) - (h_0 + v \cdot t_0) = v \cdot (t_i - t_0) = v \cdot t_i$ (since $t_0 = 0$)).

$$ \mathbb{E}(Y_{\mathrm{d}}) = \mathbb{E} (\begin{bmatrix}Y_{01}\\Y_{02}\end{bmatrix}) = \begin{bmatrix}t_1\\t_2\end{bmatrix} v = \begin{bmatrix}2\\4\end{bmatrix} v = \mathrm{A}v$$

This quiz has three questions, press the arrow below this question to continue to the next one.

What is the corresponding stochastic model?

- ○ $\Sigma_{Y_{\mathrm{d}}} =\begin{bmatrix} 1 & 0 \\ 0 & 2\end{bmatrix}$
- ○ $\Sigma_{Y_{\mathrm{d}}} =\begin{bmatrix} 1 & 1 \\ 1 & 2\end{bmatrix}$
- ○ $\Sigma_{Y_{\mathrm{d}}} =\begin{bmatrix} 2 & 0 \\ 0 & 3\end{bmatrix}$

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Precision and confidence intervals

## Contents

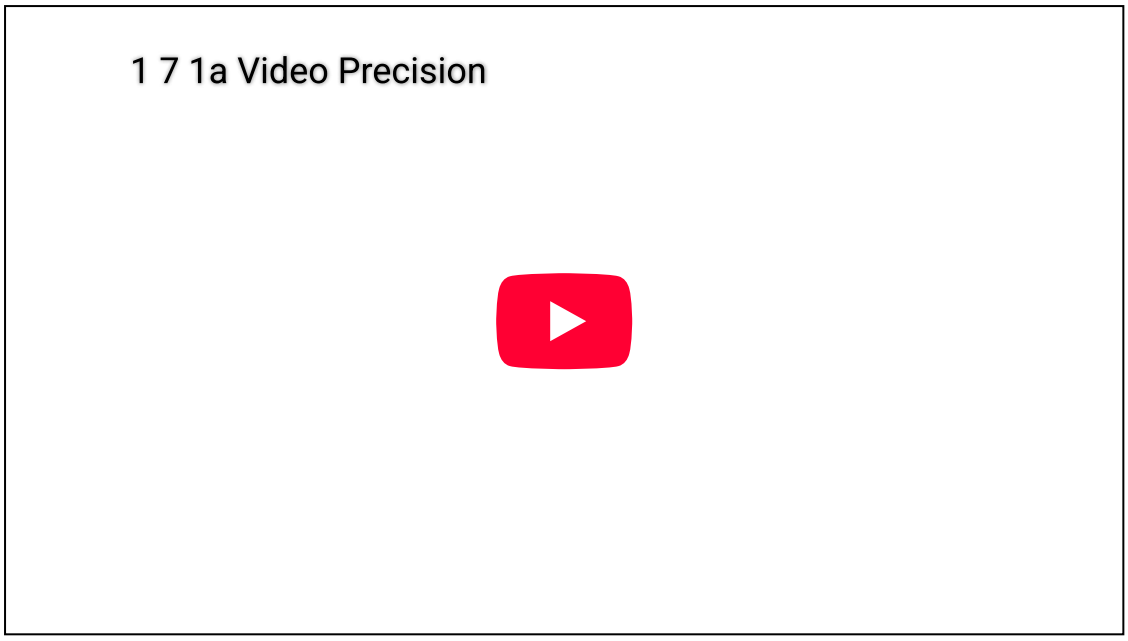

- Video
- Review normal distribution and probabilities
- Confidence interval of observations
- Confidence intervals of estimated parameters
- Factors influencing the precision


When evaluating and reporting estimation results, it is important to give a quality assessment. Thereby we should distinguish between:

- the precision of the estimated parameters (topic of this part)
- how well the model fits the data (topic of chapter Model testing)

The precision of the estimated parameters is provided by the covariance matrix $\Sigma_{\hat{X}}$. But the question is: how to present and interpret the numbers in the matrix? For that purpose we will introduce the concept of confidence intervals, which basically means we will ‘convert’ the precision to probabilities.

# Video




# Review normal distribution and probabilities

Let's start with a standard normally distributed random variable $Z \sim N(0, 1)$, which has an expectation equal to 0, and standard deviation (for us: precision) of 1. Fig. 3.6 shows the corresponding PDF.

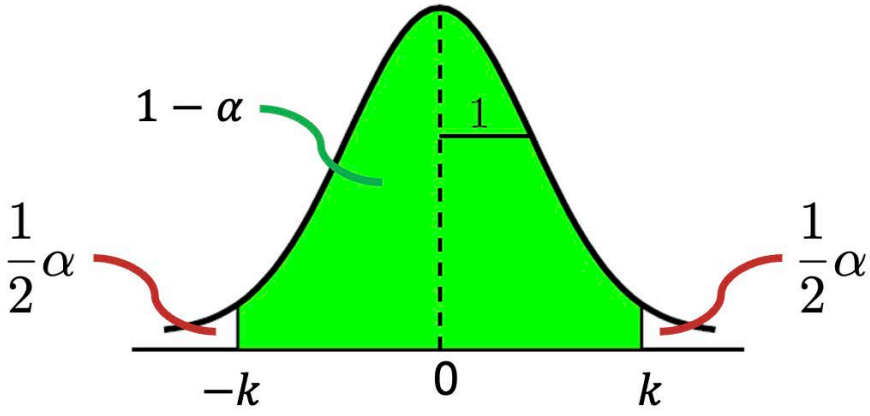



*Fig. 3.6* PDF of standard normally distributed random variable.

We are now interested in the probability that a realization of $Z$ will be within a certain interval $\pm k$ from the expectation 0:

$$P(-k < Z < k) = P(|Z| < k) = 1 - \alpha$$

In this case, $k$ is the $(1 - 0.5\alpha)$-quantile, since it can be seen in Fig. 3.6 that

$$P(Z < k) = 1 - 0.5\alpha$$

If we choose a certain value for $\alpha$ we can thus find the value of $k$ using the inverse CDF, e.g., with $\alpha = 0.05$ we would obtain $k = 1.96$, see Table Standard Normal distribution (note that you would have to look up the value for 0.025 in the table).

# Confidence interval of observations

**Definition**

A *confidence interval* of a random variable $\epsilon \sim N(0, \sigma_\epsilon^2)$ is defined as the range or interval $\epsilon \pm c$ such that:

$$P(|\epsilon| < c) = 1 - \alpha$$

where $(1 - \alpha)\cdot$ 100% is the *confidence level*.

On purpose we considered $\epsilon \sim N(0, \sigma_\epsilon^2)$ in the above definition, since in observation theory we generally assume that the random errors in our observations are normally distributed with zero mean and variance $\sigma_\epsilon^2$.

Similarly as for the standard normal distribution, the interval $\pm c$ can be computed. We will do this by first applying the transformation:

$$\frac{\epsilon - 0}{\sigma_\epsilon} = Z \sim N(0, 1)$$

With this we have that:

$$P(|\epsilon| < c) = P(|\frac{\epsilon}{\sigma_\epsilon}| < \frac{c}{\sigma_\epsilon}) = P(|Z| < k)$$

with $c = k\sigma_\epsilon$.

This allows us to compute $k$ as shown in Review normal distribution and probabilities and express the confidence interval as:

$$\epsilon \pm k\sigma_\epsilon$$

For example, for a confidence level of 95% we have $\alpha = 0.05$, for which we found $k = 1.96$. Hence the corresponding confidence interval is $\epsilon \pm 1.96\sigma_\epsilon$.

So far we considered the confidence level of a single random error. If we have multiple observations, the corresponding intervals can be evaluated. Even though the precision of each observation may be different, we only need to compute $k$ once to find the intervals $\epsilon_i \pm k\sigma_{\epsilon_i}$, $i = 1, \ldots, m$.

Fig. 3.7 shows how the confidence intervals can be visualized as error bars for the corresponding observations $y_i$.

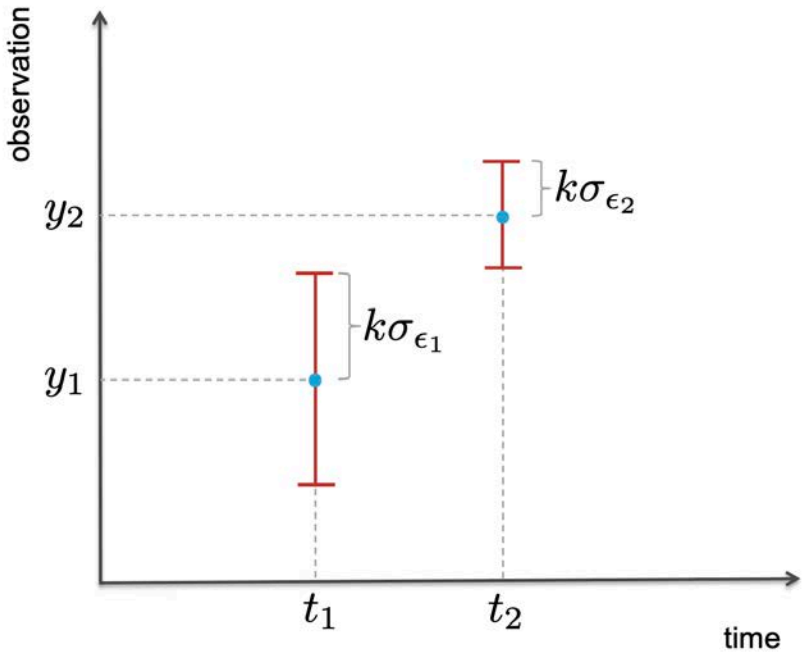



*Fig. 3.7* Confidence intervals of two observations $y_1$ and $y_2$ visualized with error bars.

**Note**

The error bars in Fig. 3.7 should not be interpreted as the interval such that there is $1-\alpha$ probability that the true value of $Y_i$ is in that interval. We are namely only looking at a single realization of each $Y_i$, while the confidence interval is computed based on the probability that the *random* error deviates $\pm k\sigma_{\epsilon_i}$ from 0, and hence that $Y_i$ deviates by that amount from its expected value. If we would repeat the two measurements, the values of $y_1$ and $y_2$ will be different and hence the error bars would be shifted.

# Confidence intervals of estimated parameters

The covariance matrix $\Sigma_{\hat{X}}$ of a linear estimator can be evaluated by applying the linear propagation law of covariances. Moreover, we know that if our observables $Y$ are normally distributed, applying linear estimation implies that the estimated parameters $\hat{X}$ are also normally distributed. Recall that the weighted least-squares and Best Linear Unbiased estimators are indeed linear estimators.

The covariance matrices of the BLUE estimator were obtained as:

$$\begin{aligned}\Sigma_{\hat{X}} &= (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\\ \Sigma_{\hat{Y}} &= \mathrm{A}\Sigma_{\hat{X}}\mathrm{A}^{\mathrm{T}}\\ \Sigma_{\hat{\epsilon}} &= \Sigma_Y - \Sigma_{\hat{Y}}\end{aligned}$$

Following the same logic as above, the confidence intervals for each estimated parameter can be evaluated as:

$$\begin{aligned}\hat{x}_i &\pm k\sigma_{\hat{X}_i}\\ \hat{y}_i &\pm k\sigma_{\hat{Y}_i}\\ \hat{\epsilon}_i &\pm k\sigma_{\hat{\epsilon}_i}\end{aligned}$$

where $k$ is determined by the chosen confidence level.

Fig. 3.8 shows a visualization of the confidence interval of the *predicted* observations, i.e., the fitted model.

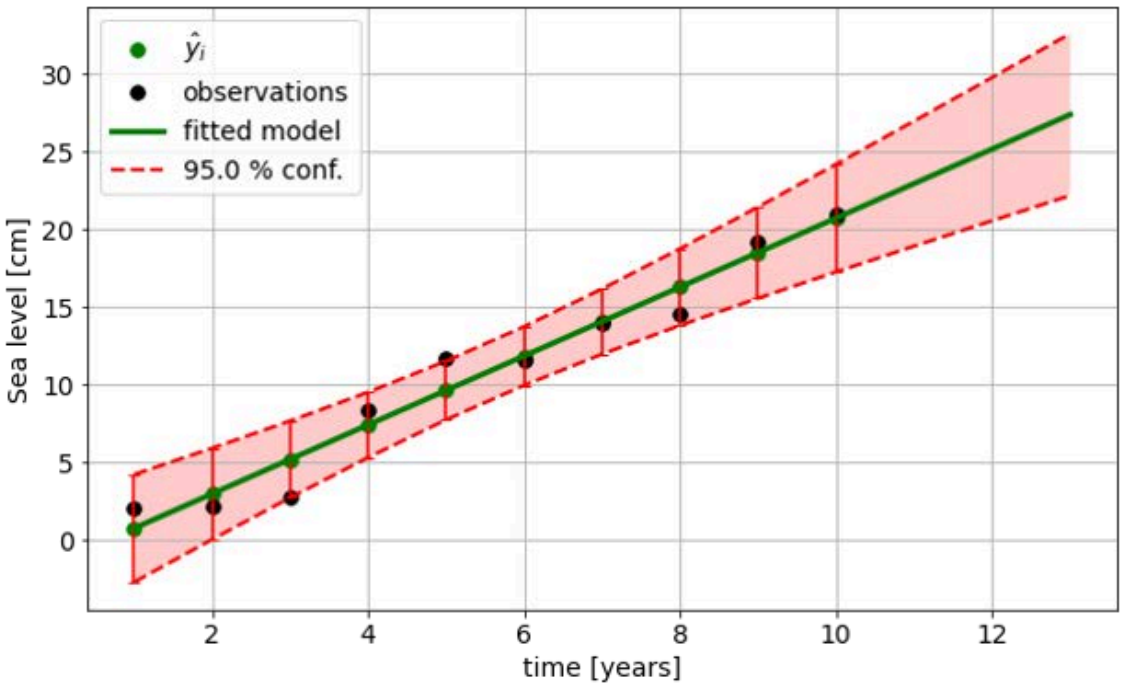



*Fig. 3.8* Confidence interval of fitted model, including error bars for the individual adjusted observations $\hat{y}_i$.

Note that covariance matrices can be computed without the need for actual observations. Hence, we can also set up a functional model including future epochs (add extra rows to the original design matrix $\mathrm{A}$). Let's call the corresponding design matrix $\mathrm{A_p}$, then $\Sigma_{\hat{Y}_p} = \mathrm{A_p}\Sigma_{\hat{X}}\mathrm{A_p^T}$. Note that the $\Sigma_{\hat{X}}$ is based on the original model, since the fitted model is only based on the actual observations. In this way, we can visualize the uncertainty of extrapolated values based on the fitted model.

# Factors influencing the precision

As can be seen from the expression $\Sigma_{\hat{X}} = (\mathrm{A^T}\Sigma_Y^{-1}\mathrm{A})^{-1}$, the precision of the estimates depends on the:

- design matrix $\mathrm{A}$: proper design of the measurement set-up can improve the precision
- covariance matrix $\Sigma_Y$: more precise observations will result in more precise estimates

Not immediately clear from the expression is that also the redundancy ($= m - n$) is an important factor: more observations will improve the precision, due to the averaging of the random errors (if $m$ large enough, the mean error will go to zero).

**Exercise uniform motion**

An object is moving along a straight line with constant but unknown speed $v$. It started at the origin $y = 0$ at $t = 0$. Uncorrelated observations $y_i$ of the object's distance from the origin $y = 0$ have been made at corresponding time instants $t_i = i$ seconds ($i = 1, 2, \ldots m$). The precision of the observations is given by $\sigma_{Y_i} = i \cdot \sigma$.

There are 7 questions for this problem, click the arrow below this question to continue the next question.

What is the functional model?

- $\mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ m \end{bmatrix} v$
- $\mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} v$
- $\mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ m \end{bmatrix} v$

Check

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Notebook: factors influencing precision

## Contents


- Notebook: factors influencing precision
- Observation model
- Plot observations and confidence intervals (95% and 99%) with all observations
- What if we reduce the number of observations (only the central 10)
- What if we thought we had better precision (factor 2 better)
- What if we have a different sampling interval?


In this notebook we will compute and plot confidence intervals for a constant velocity model.

Click 🚀 –> **Live Code** on the top right corner of this screen and then wait until all cells are executed. You will only have to run the code, and optionally may want to change some variables.

**Learning objectives:**

- evaluate the influence of changing some parameters: number of observations, standard deviation and sampling interval.
- discuss why the changes occur.
- discuss and understand the importance of correctly assessing the confidence intervals.

The code cells to import packages, with the functions used to apply BLUE and calculate the confidence intervals, and the `plot_all` function are hidden. Download the notebook to see the complete notebook.

The code cell below sets up our problem in three steps:

1. Set the observation model using the time when the observations were taken (epoch) to create our $\mathrm{A}$-matrix
2. Artificially create our random observations, `y`, with error `e`
3. Create the $\mathrm{A}$-matrix of our predictions

The simulated set of observations uses the `random.seed` method to ensure that the realizations in the random sample are always the sam. This is to ensure that everyone gets the same result; it is still a random sample.

```
t = np.vstack([1,3,3.5,5,5.5,7,9,10,10.5,
               11.5,13,14,16,16.5,18,19,
               20,22.5,24,25])
m = len(t)
A = np.hstack((np.ones((m,1)),t))
sigma = 2

np.random.seed(1613353294)
e = np.random.normal(0, sigma, m)
xo = [3,0.3]
y = A @ xo + e

m_pred = 30
t_pred = np.arange(1,m_pred+1)
A_pred = np.column_stack((np.ones(m_pred), t_pred))
```

You can also try different confidence levels.

```
# default model
plot_all(t, y, A, t_pred, A_pred, sigma, 95, 99)
```

Explain the shape of the confidence bounds and the diffence between 95% and 99% confidence interval.

**Discussion**

- The 99% confidence interval is based on a larger probability that the observation error should be in the interval, so the interval will be wider.
- Uncertainty in fitted line is due to uncertainty in estimated parameters: uncertainty in intercept manifests itself as a constant offset above and below the fitted line, the uncertainty in the slope implies uncertainty in the *angle*, which causes the widening towards the start and end.

You can also try to select a different (e.g., even shorter) time range by selecting a subset of the observations.

```
t_2 = t[5:15]
y_2 = y[5:15]
A_2 = np.hstack((np.ones((len(t_2),1)),t_2))

plot_all(t_2, y_2, A_2, t_pred, A_pred, sigma, 95, 99)
```

Compare with the first figure and discuss the differences.

**Discussion**

- Since we have a different set of observations, the estimates will be different.
- With fewer observations, the precision will be worse, hence the wider confidence intervals.
- Since all observations are in the centre, there is much more uncertainty in the predicted values outside the range of observation times.

**Note: in reality the precision of the observations is $\sigma = 2$m, so in fact this will show the confidence intervals based on an incorrect stochastic model.**

We can investigate different precision by changing the standard deviation.

```
sigma_wrong = 1
plot_all(t, y, A, t_pred, A_pred, sigma_wrong , 95, 99)
```

Compare with the first figure.

- The fitted line is the same, why is it not affected?
- Discuss the differences in confidence intervals.

**Discussion**

- Only if all observations are independent and have the same variance, the BLU estimate will be identical to the ordinary least-squares estimate (variance cancels). However, the precision will not be the same!
- With (assumed) higher precision, the confidence intervals will be tighter (errors expected to be smaller).

Note again: in this case the assumed precision is too optimistic, which is visible since the fitted model does not go through many of the blue error bars.

```
t_3 = np.vstack([7,8,8.5,9,9.5,10,10.5,11,11.5,12,13,
                 13.5,14.5,15,15.5,16,16.5,17.5,18.5,19])
A_3 = np.hstack((np.ones((len(t_3),1)),t_3))

plot_all(t_3, y, A_3, t_pred, A_pred, sigma, 95, 99)
```

Compare with the first figure.

- Discuss the differences. Especially compare the width of the confidence intervals at the centre and at the end of the time interval.
- Is it better or not to have observations with smaller sampling interval?

**Discussion**

- The number of observations is the same. The confidence interval in centre is tighter due to the smaller sampling interval, since here the the model is better constrained by the surrounding observations. However, outside the observation interval the uncertainty increases more rapidly.
- Observations on a larger range of observation times will especially contribute to less uncertainty in the slope estimation, so would be favorable, especially if we want to make predicitions outside the observation interval.

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Maximum Likelihood Estimation

## Contents


- Maximum likelihood principle
- Maximum Likelihood estimator of $\mathrm{x}$


Maximum likelihood estimation is yet another estimation principle. In contrast to (weighted) least-squares and Best Linear Unbiased estimation, it relies on the probability distribution of the observables. Recall that for BLUE, we need to know the first two moments, given by the functional and stochastic model, respectively:

$$\mathbb{E}(Y) = \mathrm{Ax}\ , \mathbb{D}(Y) = \Sigma_Y = \Sigma_\epsilon$$

in order to estimate the unknown parameters $\mathrm{x}$. However, the solution $\hat{X}$ does not depend on the underlying distribution of $Y$.

# Maximum likelihood principle

The principle of Maximum Likelihood estimation (MLE) is to find the *most likely* $\mathrm{x}$ given a realization $\mathrm{y}$ of $Y$.

This boils down to estimating the unknown parameters $\theta$ of the underlying distribution, which means that the probability density function (PDF) is known apart from the $n$ parameters in $\theta$. We will now distinguish between a PDF and likelihood function.

The *probability density function* $f_Y(\mathrm{y}|\theta)$ is given as function of $\mathrm{y}$ and with $\theta$ known.

The *likelihood function* $L(\theta|\mathrm{y})$ is given for a certain realization $\mathrm{y}$ of $Y$ as function of all possible values of $\theta$.

With MLE, the goal is to find the $\theta$ which maximizes the likelihood function for the given realization $\mathrm{y}$.

# Example exponential distribution

If $Y \sim \text{Exp}(\lambda)$, the goal would be to estimate $\lambda$ based on a realization $\mathrm{y}_{\text{obs}}$ of $Y$. Fig. 3.9 shows the PDF of $Y$

$$f_Y(\mathrm{y}|\lambda) = \lambda \exp(-\lambda \mathrm{y})$$

for different values of $\lambda$. The likelihood function for the given $\mathrm{y}_{\text{obs}}$ is shown on the right-hand side. It is shown that for instance the likelihood value for $\lambda_1$ is equal to the corresponding density value in the left panel. The maximum likelihood estimate $\hat{\lambda}$ is the value for which the likelihood function is maximized, in this case $\hat{\lambda} = \lambda_2$, as shown in the figure.

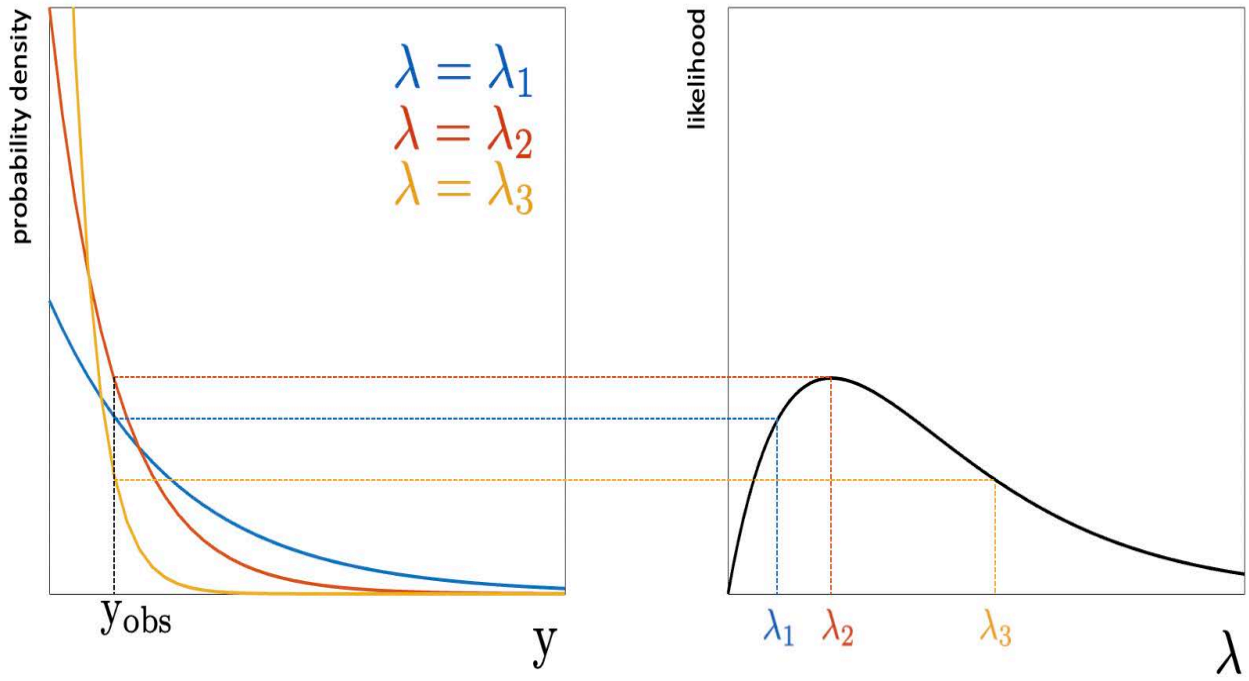



*Fig. 3.9* PDF and likelihood function for exponential distribution.

# Maximum Likelihood estimator of x

We have that our observables are assumed to be normally distributed: $Y \sim N(\mathrm{Ax}, \Sigma_Y)$, where $\mathrm{x}$ is unknown. The covariance matrix $\Sigma_Y$ is assumed to be known, for instance from a calibration campaign.

> **Note**
>
> It is also possible to consider the case that $\Sigma_Y$ is not (completely) known, for instance in case of a new sensor. Maximum Likelihood estimation can then also be applied to estimate this covariance matrix. However, this is beyond the scope of MUDE.

The likelihood function of the multivariate normal distribution is given by:

$$L(\mathrm{Ax}, \Sigma_Y|\mathrm{y}) = \underbrace{(\det 2\pi\Sigma_Y)^{-0.5}}_{c} \exp\left(-\frac{1}{2}(\mathrm{y}-\mathrm{Ax})^{\mathrm{T}}\Sigma_Y^{-1}(\mathrm{y}-\mathrm{Ax})\right)$$

Maximizing this likelihood function for $\mathrm{x}$ means that we have to find the $\mathrm{x}$ such that:

- the first-order partial derivatives (gradient) are zero: $\partial_{\mathrm{x}} L(\mathrm{Ax}, \Sigma_Y|\mathrm{y}) = 0$
- the second-order partial derivatives are negative.

Instead of working with the likelihood function, we prefer to work with the *loglikelihood* function:

$$\ln L(\mathrm{Ax}, \Sigma_Y|\mathrm{y}) = \ln c - \frac{1}{2}(\mathrm{y}-\mathrm{Ax})^{\mathrm{T}}\Sigma_Y^{-1}(\mathrm{y}-\mathrm{Ax})$$

since that is easier and results in the same maximum. Setting the gradient to zero gives:

$$\begin{aligned}\frac{\partial \ln L(\mathrm{Ax}, \Sigma_Y|\mathrm{y})}{\partial \mathrm{x}} &= \frac{\partial(-\frac{1}{2}(\mathrm{y}-\mathrm{Ax})^{\mathrm{T}}\Sigma_Y^{-1}(\mathrm{y}-\mathrm{Ax}))}{\partial \mathrm{x}} \\ &= -\frac{1}{2}(-2\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}Y + 2\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{Ax}) = 0 \\ \mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{Ax} &= \mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{y}\end{aligned}$$

The maximum likelihood estimate for a given realization $\mathrm{y}$ follows thus as:

$$\hat{\mathrm{x}} = (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{y}$$

It can be verified that the second-order partial derivatives are $-\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A}$ and hence indeed negative (since $\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A}$ is positive definite).

**MUDE exam information**

You will not be asked to derive the MLE solution as above.

If we now replace the realization $\mathrm{y}$ by the random observable vector $Y$ we obtain the *Maximum Likelihood estimator* of $\mathrm{x}$:

$$\hat{X} = (\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^{\mathrm{T}}\Sigma_Y^{-1}Y$$

It follows that the Maximum Likelihood estimator of $\mathrm{x}$ is identical to the BLUE **if** $Y$ is normally distributed.

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Non-linear least-squares estimation

## Contents


- Video
- Linearization
- Gauss-Newton iteration
- Remarks and properties


Let the non-linear system of observation equations be given by:

$$\mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = q(\mathrm{x}) = \begin{bmatrix} q_1(\mathrm{x}) \\ q_2(\mathrm{x}) \\ \vdots \\ q_m(\mathrm{x}) \end{bmatrix}$$

This system cannot be directly solved using the weighted least-squares or the best linear unbiased estimators. Here, we will introduce the non-linear least-squares principle, based on a linearization of the system of equations.

One example of a non-linear problem is the positioning model.

**Example volcano deformation**

Observable $Y_i$ is deformation rate at known location $(x_i, y_i)$ due to an unknown volume change $\Delta V$ of a magma chamber at unknown depth $d$. The observation equation according to the so-called Mogi model would be:

$$\mathbb{E}(Y_i) = q_i(\Delta V, d, x_s, y_s) = \frac{0.73 \Delta V}{\pi d^2} \left(1 + \frac{1}{d^2}\left((x_i - x_s)^2 + (y_i - y_s)^2\right)\right)^{-3/2}$$

where $(x_s, y_s)$ are the unknown horizontal coordinates of the centre of the magma chamber. This observation equation is non-linear in three out of the four unknowns.

Fig. 3.10 shows a spatial dataset with 10,000 observed deformation rates above a magma chamber. Hence we have a system of 10,000 non-linear observation equations in four unknowns. How to solve it?

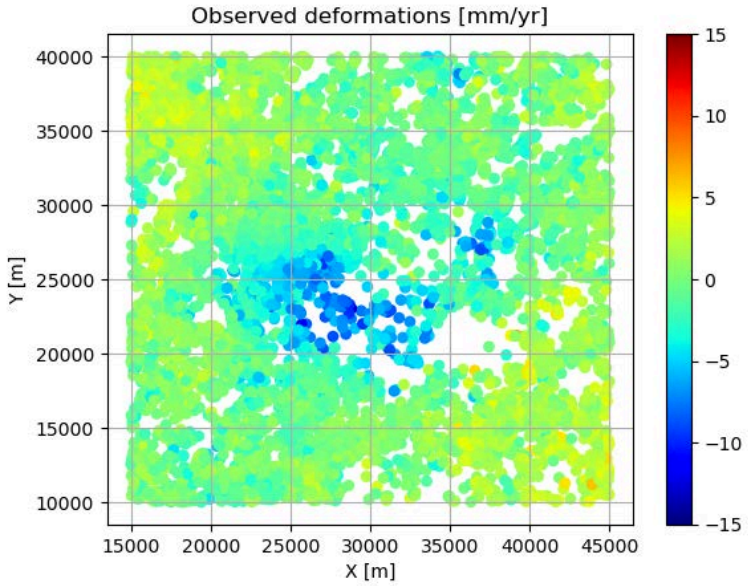



*Fig. 3.10* Observed deformation rates above a magma chamber.

# Video

1_6_3a Video NonLinearLSQ

# Linearization

For the linearization we need the first-order Taylor polynomial, which gives the linear approximation of for instance the first observable $Y_1 = q_1(\mathrm{x})$ at $\mathrm{x} = \mathrm{x}_{[0]}$ as:

$$Y_1 \approx q_1(\mathrm{x}_{[0]}) + \partial_{x_1} q_1(\mathrm{x}_{[0]})\Delta\mathrm{x}_{[0]} + \partial_{x_2} q_1(\mathrm{x}_{[0]})\Delta\mathrm{x}_{[0]} + \ldots + \partial_{x_n} q_1(\mathrm{x}_{[0]})\Delta\mathrm{x}_{[0]}$$

where $\Delta\mathrm{x}_{[0]} = \mathrm{x} - \mathrm{x}_{[0]}$ and $\mathrm{x}_{[0]}$ is the initial guess of $\mathrm{x}$. Note that for now we ignore the random errors in the equations.

Let us now consider the difference between the observable $Y_1$ and the solution of the forward model at $\mathrm{x}_{[0]}$:

$$\begin{aligned}\Delta Y_{1[0]} &= Y_1 - q_1(\mathrm{x}_{[0]}) \\ &\approx \begin{bmatrix}\partial_{x_1} q_1(\mathrm{x}_{[0]}) & \partial_{x_2} q_1(\mathrm{x}_{[0]}) & \ldots & \partial_{x_n} q_1(\mathrm{x}_{[0]})\end{bmatrix}\Delta\mathrm{x}_{[0]}\end{aligned}$$

which we refer to as the *observed-minus-computed* observable. Note that $\Delta\mathrm{x}_{[0]}$ is an $n \times 1$ vector.

We can now obtain the linearized functional model:

$$\begin{aligned}E\{\begin{bmatrix}\Delta Y_1 \\ \Delta Y_2 \\ \vdots \\ \Delta Y_m\end{bmatrix}_{[0]}\} &= \begin{bmatrix}\partial_{x_1} q_1(\mathrm{x}_{[0]}) & \partial_{x_2} q_1(\mathrm{x}_{[0]}) & \ldots & \partial_{x_n} q_1(\mathrm{x}_{[0]}) \\ \partial_{x_1} q_2(\mathrm{x}_{[0]}) & \partial_{x_2} q_2(\mathrm{x}_{[0]}) & \ldots & \partial_{x_n} q_2(\mathrm{x}_{[0]}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial_{x_1} q_m(\mathrm{x}_{[0]}) & \partial_{x_2} q_m(\mathrm{x}_{[0]}) & \ldots & \partial_{x_n} q_m(\mathrm{x}_{[0]})\end{bmatrix}\Delta\mathrm{x}_{[0]} \\ &= \mathrm{J}_{[0]}\Delta\mathrm{x}_{[0]}\end{aligned}$$

with Jacobian matrix $\mathrm{J}_{[0]}$.

And this allows to apply best linear unbiased estimation (BLUE) to get an estimator for $\Delta\mathrm{x}_{[0]}$:

$$\Delta\hat{X}_{[0]} = \left(\mathrm{J}_{[0]}^T \Sigma_Y^{-1} \mathrm{J}_{[0]}\right)^{-1} \mathrm{J}_{[0]}^T \Sigma_Y^{-1} \Delta Y_{[0]}$$

since the Jacobian $\mathrm{J}_{[0]}$ takes the role of the design matrix $\mathrm{A}$ and the observation vector is replaced by its $\Delta$-version as compared to the BLUE of the linear model $\mathbb{E}(Y) = \mathrm{A}\mathrm{x}$.

Since we have $\Delta\mathrm{x}_{[0]} = \mathrm{x} - \mathrm{x}_{[0]}$, an “estimator” of $\mathrm{x}$ can thus be obtained as:

$$\hat{X} = \mathbf{x}_{[0]} + \Delta\hat{X}_{[0]}$$

However, the quality of the linear approximation depends very much on the closeness of $\mathbf{x}_{[0]}$ to $\mathbf{x}$. Therefore, instead we apply an iterative procedure, which means that we repeat the process with $\mathbf{x}_{[1]} = \mathbf{x}_{[0]} + \Delta\hat{X}_{[0]}$ as our new guess. This procedure is continued until the $\Delta\hat{\mathbf{x}}_{[i]}$ becomes small enough. This is called the Gauss-Newton iteration procedure, and is shown in the scheme below.

# Gauss-Newton iteration

See Fig. 3.11 for an illustration of the iterative procedure, referred to as the Gauss-Newton iteration.

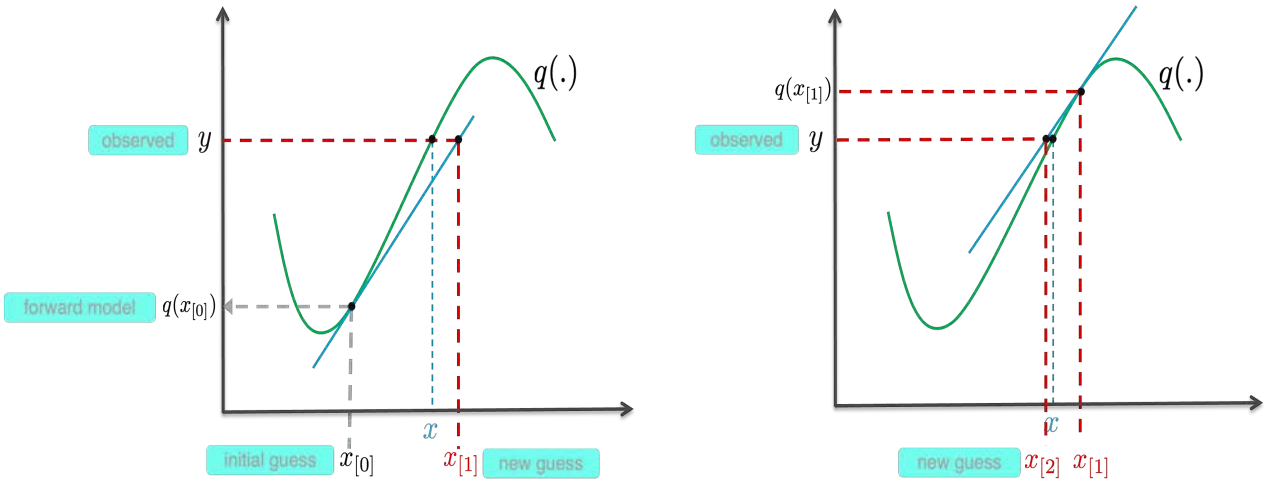



*Fig. 3.11* Visualization of two iteration steps of Gauss-Newton iteration. The non-linear function $q$ is shown, together with the linear approximations (blue line) at $q(x_{[i]})$ in each step.

Assume we have the observation vector $\mathbf{y}$ (realization of $Y$).

Start with initial guess $\mathbf{x}_{[0]}$, and start iteration with $i = 0$

1. Calculate observed-minus-computed $\Delta\mathbf{y}_{[i]} = \mathbf{y} - q(\mathbf{x}_{[i]})$
2. Determine the Jacobian $\mathbf{J}_{[i]}$
3. Estimate $\Delta\hat{\mathbf{x}}_{[i]} = \left(\mathbf{J}_{[i]}^T \Sigma_Y^{-1} \mathbf{J}_{[i]}\right)^{-1} \mathbf{J}_{[i]}^T \Sigma_Y^{-1} \Delta\mathbf{y}_{[i]}$
4. New guess $\mathbf{x}_{[i+1]} = \Delta\hat{\mathbf{x}}_{[i]} + \mathbf{x}_{[i]}$
5. If stop criterion is met: set $\hat{\mathbf{x}} = \mathbf{x}_{[i+1]}$ and break, otherwise set $i := i + 1$ and go to step 1

In this scheme we need to choose a stop criterion, for which we use:

$$\Delta\hat{\mathbf{x}}_{[i]}^T \mathbf{N}_{[i]} \Delta\hat{\mathbf{x}}_{[i]} < \delta \text{ with } \mathbf{N}_{[i]} = \mathbf{J}_{[i]}^T \Sigma_Y^{-1} \mathbf{J}_{[i]}$$

The normal matrix $\mathbf{N}_{[i]}$ is used as a weight matrix, since it is equal to the inverse of the covariance matrix of the estimated $\Delta\hat{\mathbf{x}}_{[i]}$, and estimated parameters with small variance should have a relatively small deviation compared to parameters with large variance.

The threshold $\delta$ must be set to a very small value, e.g., $10^{-8}$.

Once the stop criterion is met, we say that the solution *converged*, and the last solution $\mathbf{x}_{[i+1]}$ is then finally used as our estimate of $\mathbf{x}$.

**Exercise**

### Orbital period of satellite

The formula for the orbital period of a satellite is given by

$$p = 2\pi\sqrt{\frac{a^3}{GM}}$$

where $a$ is the unknown semi-major axis of the orbit, $G$ is the gravitational constant, $M$ is the mass of the Earth and $GM = 398600$ km$^3$s$^{-2}$ (known).

The period is measured two times: $p_1 = 6081.2$ s and $p_2 = 6081.5$ s. The goal is to estimate $a$ from these measurements using non-linear least squares; we will apply a Gauss-Newton iteration with the initial guess $a_{[0]} = 7200$ km.

We want to determine the $2 \times 1$ Jacobian matrix using the initial guess.

The first value of this Jacobian is:

The second value of this Jacobian is:

Check

# Remarks and properties

There is no default recipe for making the initial guess $\mathbf{x}_{[0]}$; it must be made based on insight into the problem at hand. A good initial guess is important for two reasons:

- a bad initial guess may result in the solution NOT to converge;
- a good initial guess will speed up the convergence, i.e. requiring fewer iterations.

The covariance matrix of $\hat{X}$ is given by:

$$\Sigma_{\hat{X}} = \left( \mathrm{J}_{[i]}^T \Sigma_Y^{-1} \mathrm{J}_{[i]} \right)^{-1}$$

although this is not strictly true. The estimator $\hat{X}$ is namely NOT a best linear unbiased estimator of $\mathbf{x}$ since we used a linear approximation. And since the estimator is not a linear estimator due to ignoring the higher-order terms, it is not normally distributed.

However, in practice the linear (first-order Taylor) approximation often works so well that the performance is very close to that of BLUE and we may assume the normal distribution.

As a final note we would like to mention that the Gauss-Newton method is just one approach for solving a non-linear least-squares problem. In practice, especially for highly non-linear problems, the Levenberg-Marquardt method is often used. It can be considered as a refinement of the Gauss-Newton method with a higher chance of convergence.

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

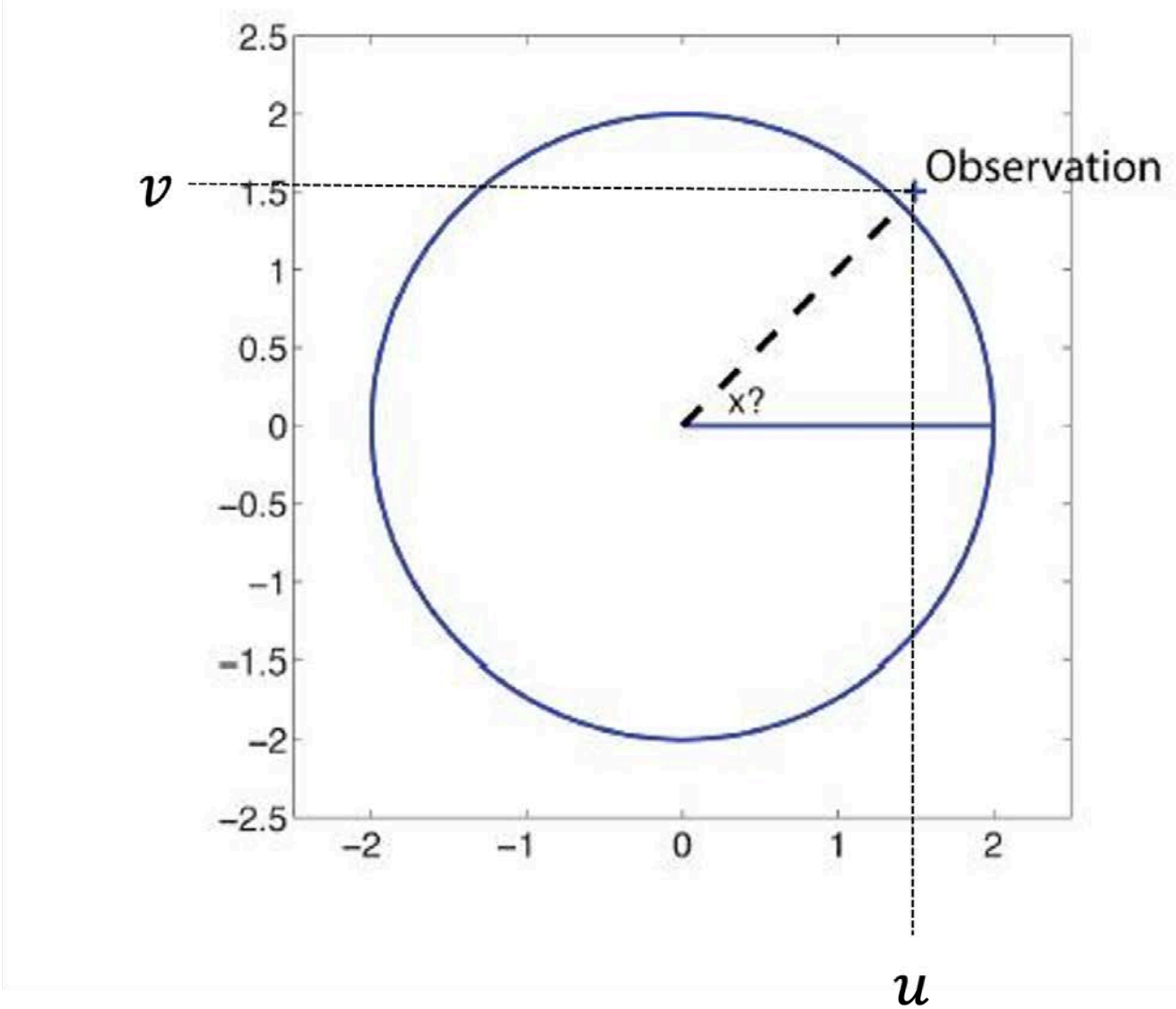
# Notebook Gauss-Newton iteration for a point on a circle

If you want to download the notebook, select the .zip after clicking on .

Click –> Live Code on the top right corner of this screen and then wait until all cells are executed.

In this exercise we will consider the following problem:

we have a point on a circle with known radius $r = 2$ meters. We are interested in the unknown angle $x = \alpha$. We do have an observation of both the $u$- and $v$-coordinate. Our observations are $u = 1.4957$ and $v = 1.5048$ m.



By completing the code below you will perform the Gauss-Newton iteration to obtain an estimate for the angle. Note that we express the angle in radians.

Our functional model is given by:

$$\mathbb{E}(Y) = \begin{bmatrix} 2\cos x \\ 2\sin x \end{bmatrix}$$

We assume that both observables have a precision of 5 mm.

The observation vector `y` and `inv_Sigma_Y`$= \Sigma_Y^{-1}$ are already defined.

You are asked to derive what the Jacobian matrix is and complete the code to compute:

`y_comp` $= q(x_{[i]})$

`dy` $= \Delta y_{[i]}$

`J`$= J_{[i]}$

Change the value of the initial value `x_init` to see the effect in the figures.

In this case we will not use a stop criterion for the iteration, but simply stop after `N=10` iterations.

```
y = np.array([1.4957, 1.5048])
var_Y = 5e-3**2
inv_Sigma_Y = 1/var_Y * np.eye(2)

x_init = 2

N = 10
dx_i = np.zeros(N) # initialize the dx_i vector
x_i = np.zeros(N+1) # initialize the x_i vector
x_i[0] = x_init

for i in np.arange(N):
    y_comp = np.array([ ? , ? ])
    dy = ?
    J = np.array([ ? , ? ])

    # note that (J.T @ inv_Sigma_Y @ J) is a scalar, therefore we
    # use 1/(...) instead of np.linalg.inv(...)
    dx_i[i] = 1/(J.T @ inv_Sigma_Y @ J) * J.T @ inv_Sigma_Y @ dy
    x_i[i+1] = x_i[i] + dx_i[i]

# set final estimate equal to the last value in x_i and evaluate its precision
xhat = x_i[-1]
sigma_xhat = np.sqrt(1/(J.T @ inv_Sigma_Y @ J))

print(f'The final estimate for the angle is: {xhat:.3f} radians\n')
print(f'The precision of the estimated angle is: {sigma_xhat:.2e} radians')

fig, ax = plt.subplots(1, 2, figsize = (16, 6))
ax[0].plot(dx_i, 'r-o')
ax[0].set_xlabel('iteration number $i$', fontsize = 15)
ax[0].set_ylabel('$\Delta \hat{x}_{[i]}$ [rad]', fontsize = 15)
ax[0].grid()
ax[1].plot(x_i, 'r-o')
ax[1].set_xlabel('iteration number $i$', fontsize = 15)
ax[1].set_ylabel('$x_{[i]}$ [rad]', fontsize = 15)
ax[1].grid()
```

The left-side figure shows for each iteration the value of $\Delta\hat{x}_{[i]}$, which should become smaller and smaller if all goes well. On the right-hand side you see how $x_{[i]}$ is updated in each iteration step, where the final value will be used as our estimate $\hat{x} = x_{[10]}$.

**Solutions**

```
y_comp = np.array([ 2 * np.cos(x_i[i]) , 2 * np.sin(x_i[i]) ])

dy = y - y_comp

J = np.array([ -2 * np.sin(x_i[i]) , 2 * np.cos(x_i[i]) ])
```

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Model testing

## Contents


- Statistical hypothesis testing: principle


In Chapter Precision and confidence intervals one part of the quality assessment was presented. But evaluating the precision does not tell us how well the model fits the data we collected. It might be that we worked with a wrong functional model (e.g., too simplistic), or a wrong stochastic model. Or it may be that our observations are affected by blunders or systematic biases.

In some cases, a first indication that something may be wrong can be obtained by plotting the observations and the fitted model together with the confidence intervals. If the residuals are large compared to the confidence interval this is obviously an indication that something is wrong. Another indication is if you see a clear pattern in the residuals once you plot those as function of time or location.

**Example volcano deformation**

For the volcano deforomation example, the observed and modelled deformations are shown in Fig. 3.12, as well as the residuals (on the right). In this case, we can see some ‘patterns’: two clusters with positive residuals in the upper-left and lower-right corner, where an uplift is observed which is not accounted for by the model. There is also a cluster of negative residuals, where the subsidence is larger than the model predicts. The model assumed a spherical deformation, which is apparently too simplistic.

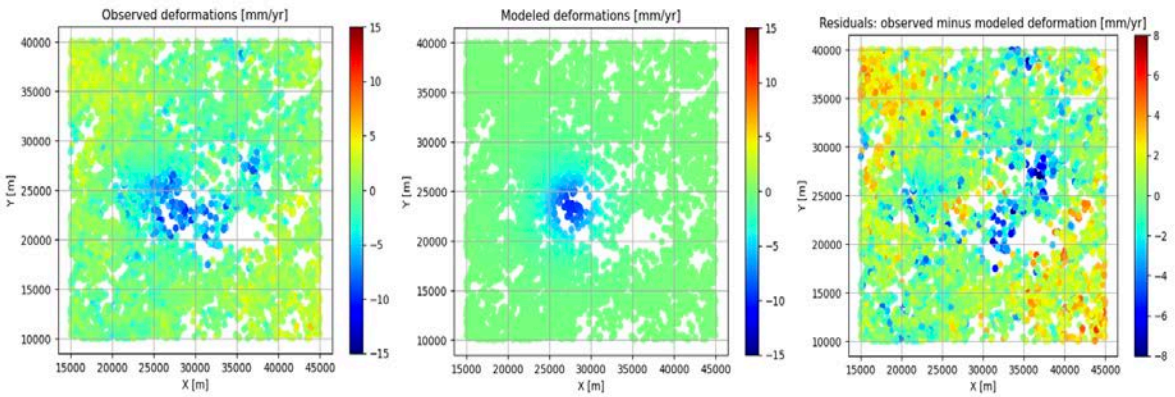


*Fig. 3.12* Observed deformation rates above a magma chamber (left), the modelled deformation after applying non-linear least-squares (middle), and the residuals (right).

However, it is not always straightforward to decide whether the residuals are too large or show a suspicious pattern, especially for models with more than two unknown parameters. Moreover, the question remains: is the fit good enough or not? And if not, what is the underlying misspecification/error?

To answer these questions, we need to apply statistical hypothesis testing. In this chapter, we will introduce the hypothesis testing principle, and then explain how it can be applied in sensing and monitoring to:

- test for blunders or systematic biases in the observations;
- test for misspecifications of the functional model and/or decide between two competing hypotheses regarding the functional model.

# Statistical hypothesis testing: principle

Statistical hypothesis testing means that we apply a certain test to decide between two (or more) competing hypothesis regarding the underlying model (in our case the functional or stochastic model). Therefore we assume a nominal model, corresponding to the null hypothesis $\mathcal{H}_0$, and another model for the alternative hypothesis $\mathcal{H}_a$.

An important note to be made here is that the null hypothesis is presumed to be true unless the data provides convincing evidence against it (i.e., there is significant misfit).

> This is similar as in the court house: a defendant is presumed to be innocent until proven guilty based on convincing evidence.

**Example laser distometer**

We would like to test whether a laser distometer is well-calibrated, implying that the mean error of repeated measurements should be zero (null hypothesis). An alternative hypothesis in this case could be that all measurements are affected by a certain constant offset (i.e., systematic bias). We can also write this as:

$$\mathcal{H}_0 : \mathbb{E}(\epsilon) = 0$$
$$\mathcal{H}_a : \mathbb{E}(\epsilon) = \nabla$$

The next step is then to define a so-called *test statistic* $T$ with a known distribution. Based on the actual observations, a realization of this test statistic can be computed and it can be assessed whether this realization is a likely outcome given the distribution. This can be illustrated based on the example.

**Example laser distometer (continued)**

Assuming that the random errors of the observed distances with the laser distometer are normally distributed with zero mean and standard deviation $\sigma$, a logical choice for the test statistic would be simply to use the mean error. To test whether the laser distometer is well-calibrated we measure a known distance $d$ $m$ times, such that we can calculate the errors $\epsilon_i = y_i - d$ $(i = 1, \ldots, m)$. All measurements are assumed to be independent. Applying the variance propagation law, we find that the variance of the mean error will be $\frac{\sigma^2}{m}$.

We thus have for the null and alternative hypothesis:

$$\mathcal{H}_0 : T = \frac{1}{m} \sum_{i=1}^{m} \epsilon_i \sim N(0, \frac{\sigma^2}{m})$$
$$\mathcal{H}_a : T = \frac{1}{m} \sum_{i=1}^{m} \epsilon_i \sim N(\nabla, \frac{\sigma^2}{m})$$

Fig. 3.13 shows the PDFs of the test statistic under the null and alternative hypothesis. Based on the distributions, the null hypothesis is obviously more likely in case the observed mean error (realization of $T$) is smaller than $k$. Therefore the null hypothesis would be accepted if $T \leq k$ and rejected otherwise. The corresponding *acceptance* and *critical region* are also shown.

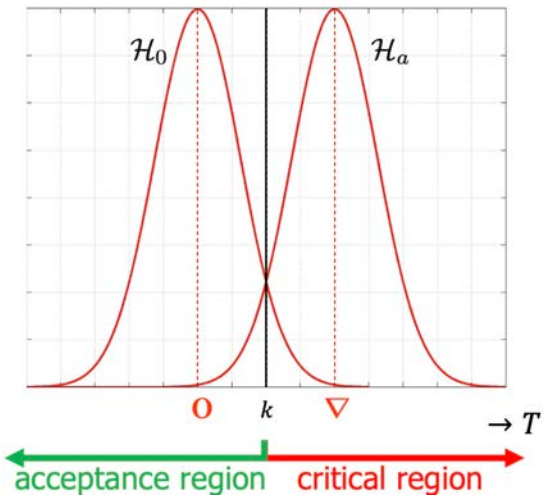



*Fig. 3.13* PDFs of test statistic $T$ under null and alternative hypothesis, with threshold value $k$.

Even though the choice for the threshold value in the example above may seem logical, it does not account for the requirement that there should be *convincing evidence* against the null hypothesis being true. Even though the probability that the test statistic is larger than $k$ if the null hypothesis is true, $P(T > k|\mathcal{H}_0)$, is small, it is not zero! A better approach would therefore be to determine the threshold value based on the choice of an acceptable *false alarm probability* $\alpha$, since a false alarm (also referred to as type I error) corresponds to rejecting the null hypothesis while it is true.

**Definition**

The false alarm or type I error probability $\alpha$ is defined as:

$$\alpha = P(T \in \mathcal{C}|\mathcal{H}_0)$$

where $\mathcal{C}$ is the critical region.

In the example, the bias was assumed to be positive, such that there is a right-side critical region $\mathcal{C}$, but in practice the bias may also be negative (left-side critical region), or we may have a two-sided critical region.

See Fig. 3.14, where the principle has been applied for the same example as above. Given that under the null hypothesis $T \sim N(0, \sigma^2/m)$, we can easily determine the threshold value $k_\alpha$ for a given $\alpha$ using the inverse CDF. The critical region $\mathcal{C}$ is thus given by $T > k_\alpha$.

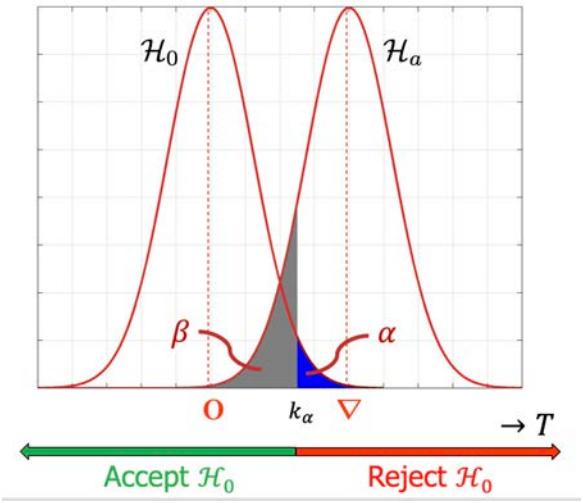



*Fig. 3.14* PDFs of test statistic $T$ under null and alternative hypothesis, with false alarm probability $\alpha$ and threshold value $k_\alpha$.

Besides the probability of a false alarm, the figure also shows the probability of *false acceptance* of $\mathcal{H}_0$, referred to as the type II error probability, or missed detection probability. In this case, the alternative hypothesis is true, but still the null hypothesis is accepted.

**Definitions**

The missed detecion or type II error probability $\beta$ is defined as:

$$\beta = P(T \notin \mathcal{C} | \mathcal{H}_a)$$

where $\mathcal{C}$ is the critical region.

The detection power or probability of detection is:

$$\gamma = 1 - \beta$$

Ideally, both the type I and type II error probabilities should be small. However, as can be seen from the figure, choosing a smaller value for $\alpha$ implies a larger $\beta$ and vice versa.

The *decision matrix* in `decision` summarizes the four types of decisions and corresponding probabilities.

| | | reality | |
|---|---|---|---|
| | | $\mathcal{H}_0$ true | $\mathcal{H}_a$ true |
| decision | Accept $\mathcal{H}_0$ | $1-\alpha$ | $\beta$ |
| | Reject $\mathcal{H}_0$ | $\alpha$ | $\gamma$ |

*Fig. 3.15* Decision matrix for hypothesis testing.

**Exercise**

### Hypothesis Testing

Which of the following statements is correct?

- [ ] Probability of a type I error is equal to 1 minus the probability of false alarm
- [ ] Probability of a type II error is equal to 1 minus the power of the test.
- [ ] A larger threshold value results in a larger probability of false alarm.
- [ ] The power of the test decreases if probability of false alarm increases.
- [ ] The power of a test is the probability that you accept the null hypothesis correctly.
- [ ] A large probability of false alarm implies a larger probability that we reject the null hypothesis.

Check

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Hypothesis testing for Sensing and Monitoring

## Contents


- Common alternative hypotheses are:
- Testing procedure
- Overall model test
- Estimation and testing procedure


Statistical hypothesis testing is applied for many applications and in different forms. Here, we will restrict ourselves to applications in observation theory with the following general form for the hypotheses:

$$\begin{aligned} \mathcal{H}_0 &: \mathbb{E}(Y) = \mathrm{A}\mathrm{x}; & \mathbb{D}(Y) &= \Sigma_Y \\ \mathcal{H}_a &: \mathbb{E}(Y) = \mathrm{A}\mathrm{x} + \mathrm{C}\nabla = [\mathrm{A} \quad \mathrm{C}] \begin{bmatrix} \mathrm{x} \\ \nabla \end{bmatrix}; & \mathbb{D}(Y) &= \Sigma_Y \end{aligned}$$

where $\nabla$ is a $q$-vector with extra parameters compared to the nominal model under $\mathcal{H}_0$, and $\mathrm{C}$ is the $m \times q$ matrix augmenting the design matrix.

## Common alternative hypotheses are:

- One or more observations are actually outliers (e.g., due to malfunctioning / external impact on sensor).
- A systematic offset or drift in a subset of the measurements.
- More complicated model, e.g., linear trend versus second-order polynomial, or an additional seasonal signal.

**Exercises**

Consider the linear trend model as our null hypothesis:

$$\mathcal{H}_0 : \mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \underbrace{\begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}}_{\text{A}} \underbrace{\begin{bmatrix} x_0 \\ v \end{bmatrix}}_{\text{x}}$$

Note that $x_0$ is the intercept at $t = 0$, $v$ is the velocity (or slope).

What is the alternative hypothesis if we assume the second observation to be an outlier?

**Solution**

$$\mathcal{H}_0 : \mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \mathrm{Ax} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \nabla$$

A bias of size $\nabla$ is added to the observation equation of the second observations. In this case $q = 1$.

What is the alternative hypothesis if we assume the last three observations to be affected by a systematic offset?

**Solution**

$$\mathcal{H}_0 : \mathbb{E}(\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix}) = \mathrm{Ax} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \nabla$$

Systematic offset of size $\nabla$ is added to the observation equations of the last three observations. In this case $q = 1$. This alternative hypothesis could also represent a deformation event.

What is the alternative hypothesis if we assume that the velocity $w$ after a certain $t_i$ is different than before (change of slope)? Hint: first try to see that the observation equation for an observation at $t_j > t_i$ becomes $\mathbb{E}(Y_j) = x_0 + vt_i + w(t_j - t_i)$.

**Solution**

$$\mathcal{H}_0 : \mathbb{E}(\begin{bmatrix} Y_1 \\ \vdots \\ Y_i \\ Y_{i+1} \\ \vdots \\ Y_m \end{bmatrix}) = \mathrm{Ax} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ t_{i+1} - t_i \\ \vdots \\ t_m - t_i \end{bmatrix} v_i$$

Also in this case $q = 1$.

# Testing procedure

## Step 1 is to apply BLUE for both the null and alternative hypothesis

Under $\mathcal{H}_0$ we obtain the estimator of $\mathrm{x}$ and the residuals as:

$$\hat{X} = (\mathrm{A}^T \Sigma_Y^{-1} \mathrm{A})^{-1} \mathrm{A}^T \Sigma_Y^{-1} Y$$
$$\hat{\epsilon} = Y - \mathrm{A}\hat{X}$$

The residuals are important to measure the misfit between individual observations and the fitted model. Recall that BLUE provides us with the solution for $\mathrm{x}$ which minimizes the weighted squared norm of residuals $\hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon}$ (note that $\Sigma_Y = \Sigma_\epsilon$), see Weighted least-squares estimation, and it makes sense that this is a good measure for the discrepancy between data and model. This is especially clear if we consider (the common) case that $\Sigma_Y$ is a diagonal matrix with the variances $\sigma_i^2$ $(i = 1, \ldots, m)$ of the random errors on the diagonal, since then:

$$\hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} = \sum_{i=1}^{m} \frac{\epsilon_i^2}{\sigma_i^2}$$

This is thus the sum of the *squared residuals* (= estimated errors) divided by the variance of the random errors.

It can be shown that the distribution of the weighted squared norm of residuals is given by the central $\chi^2$-distribution:

$$\hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} \sim \chi^2(m-n, 0)$$

with $m-n$ the redundancy (number of observations $m$ minus number of unknowns $n$).

For the alternative hypothesis, the BLUE solution follows as:

$$\begin{bmatrix} \hat{X}_a \\ \hat{\nabla} \end{bmatrix} = ([\mathrm{A} \quad \mathrm{C}]^T \Sigma_Y^{-1} [\mathrm{A} \quad \mathrm{C}])^{-1} [\mathrm{A} \quad \mathrm{C}]^T \Sigma_Y^{-1} Y$$

$$\hat{\epsilon}_a = Y - [\mathrm{A} \quad \mathrm{C}] \begin{bmatrix} \hat{X}_a \\ \hat{\nabla} \end{bmatrix}$$

where the subscript $_a$ is used to indicate that the solution is different from the one with the null hypothesis. Also, the corresponding weighted squared norm of residuals $\hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a$ is to be computed.

# Step 2: apply Generalized Likelihood Ratio test

As a test statistic to decide between $\mathcal{H}_0$ and $\mathcal{H}_a$ the difference between the weighted squared norms of residuals is used, which is known to have a Central $\chi^2$-distribution if $\mathcal{H}_0$ is true:

$$T_q = \hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} - \hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a \sim \chi^2(q, 0)$$

where $q$ is equal to the number of additional parameters in $\mathcal{H}_a$ (i.e., the length of the $\nabla$-vector).

We would reject $\mathcal{H}_0$ if:

$$T_q = \hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} - \hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a > k_\alpha$$

with $k_\alpha$ the threshold value based on a choice for the false alarm probability $\alpha$, see Fig. 3.16

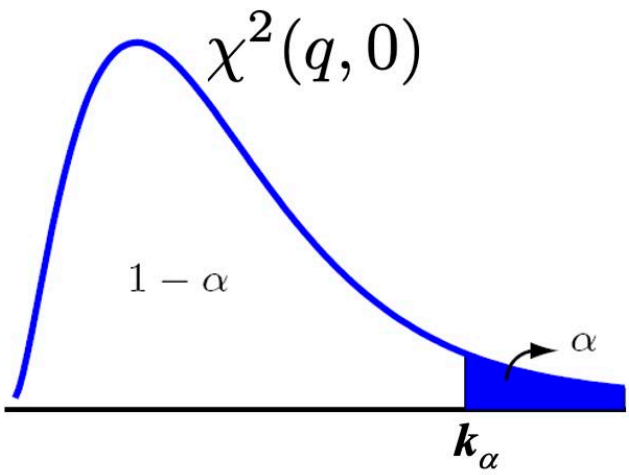



*Fig. 3.16* PDF of $\chi 2$(q,0)-distribution with threshold value $k_\alpha$ based on chosen false alarm probability $\alpha$.

This test is known as the *Generalized Likelihood Ratio Test* (GLRT). It can be derived by using the the ratio of the likelihood functions for $\mathcal{H}_0$ and $\mathcal{H}_a$ (recall that the BLUE solution is equal to the maximum likelihood solution in our case since the data are assumed to be normally distributed). This is a logical choice since the purpose of our test is to decide whether $\mathcal{H}_0$ is sufficiently likely compared to $\mathcal{H}_a$. The derivation is beyond the scope of MUDE.

Important to mention is that this is the *most powerful* test: no other test statistic can be found that would result in a larger detection power $\gamma$ for the same $\alpha$.

**Exercise**

## Test Statistic

*There are three subquestions, click on the arrow at the bottom to continue to the next.*

For a given estimation problem we applied BLUE for both $\mathcal{H}_0$ and $\mathcal{H}_a$ and obtained the Generalized Likelihood Ratio test statistic value $T_q$ of 9.5. In $\mathcal{H}_0$ we have 5 unknowns, in $\mathcal{H}_a$ there are 7 unknowns. We will use a false alarm probability of 1%.

What is the threshold value $k_\alpha$?

Check

# Overall model test

Many options for the alternative hypothesis are possible, but instead of immediately testing the null hypothesis against a very specific alternative, it common to first apply the so-called *overall model test* (OMT): it is a test which just focuses on the validity of $\mathcal{H}_0$ (how well do data and model fit) without having a specific $\mathcal{H}_a$ in mind. This is identical to choosing the alternative hypothesis as follows:

$$\mathcal{H}_a: \quad \mathbb{E}(\underset{m\times 1}{Y}) = \underset{m\times m}{\begin{bmatrix} \underset{m\times n}{\mathrm{A}} & \underset{m\times m-n}{\mathrm{C}} \end{bmatrix}} \underset{m\times 1}{\begin{bmatrix} \mathrm{x} \\ \nabla \end{bmatrix}}$$

Hence, $q = m - n$ and the matrix $[\mathrm{A} \quad \mathrm{C}]$ becomes a square invertible matrix, such that the system becomes determined and the BLUE follows as (try to show yourself):

$$\begin{bmatrix} \hat{X}_a \\ \hat{\nabla} \end{bmatrix} = [\mathrm{A} \quad \mathrm{C}]^{-1} Y$$

**Answer**

$$\begin{aligned} \begin{bmatrix} \hat{X}_a \\ \hat{\nabla} \end{bmatrix} &= ([\mathrm{A} \quad \mathrm{C}]^T \Sigma_Y^{-1} [\mathrm{A} \quad \mathrm{C}])^{-1} [\mathrm{A} \quad \mathrm{C}]^T \Sigma_Y^{-1} Y \\ &= [\mathrm{A} \quad \mathrm{C}]^{-1} \Sigma_Y [\mathrm{A} \quad \mathrm{C}]^{-T} [\mathrm{A} \quad \mathrm{C}]^T \Sigma_Y^{-1} Y \\ &= [\mathrm{A} \quad \mathrm{C}]^{-1} Y \end{aligned}$$

From this follows that the residuals become zero:

$$\hat{\epsilon}_a = Y - [\mathrm{A} \quad \mathrm{C}] \begin{bmatrix} \hat{X}_a \\ \hat{\nabla} \end{bmatrix} = Y - [\mathrm{A} \quad \mathrm{C}][\mathrm{A} \quad \mathrm{C}]^{-1} Y = 0$$

Hence, any alternative hypothesis for which the design matrix $[\mathrm{A} \quad \mathrm{C}]$ is an $m \times m$ invertible matrix will result in residuals being equal to 0. This implies that any observed $\mathrm{y}$ will satisfy the alternative hypothesis, which can thus also be formulated as

$$\mathcal{H}_a: \quad Y \in \mathbb{R}^m$$

The OMT is obtained from the GLRT:

$$T_{q=m-n} = \hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} > k_\alpha$$

since $\hat{\epsilon}_a = 0$.

**Exercise**

## Overall model test

*There are three subquestions, press the arrow to continue to the next one.*

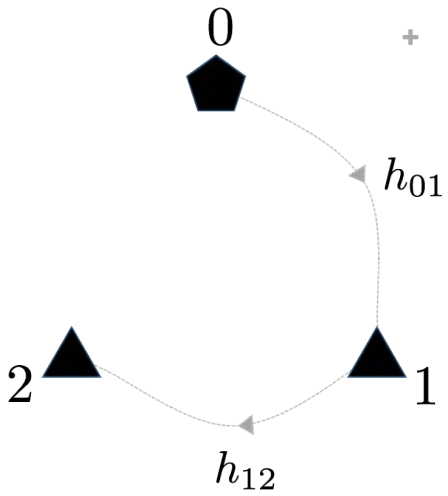
We apply an overall model test for a linear trend model (unknown intercept and velocity) and assume all 24 observations to be independent and have a standard deviation of 1 cm. We choose $\alpha = 0.01$.

What is the threshold value (also known as critical value) for this test? You can use Python or use the $\chi^2$-distribution table.

Threshold value (2 decimals): [ ]

Check

**Exercise**




We are interested in the unknown heights of two points, and therefore we measure the height differences with respect to a reference point 0 with known height $h_0 = 0$ mm. The observed height differences are shown in the figure, where $h_{ij} = h_j - h_i$ m. The observations are independent and have a precision of $\sigma$ mm.

The observation model is therefore given by:

$$\mathbb{E}(Y) = \begin{bmatrix} h_1 - h_0 \\ h_2 - h_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}, \quad \Sigma_Y = \sigma^2 I_2$$

Try to argue which of the following is true.

- ○ Precision of both $\hat{h}_1$ and $\hat{h}_2$ will be the same.
- ○ $\sigma_{\hat{h}_1} > \sigma_{\hat{h}_2}$
- ○ $\sigma_{\hat{h}_1} < \sigma_{\hat{h}_2}$

Check

# Estimation and testing procedure

If the OMT is accepted you are done and can present your estimation results, including precision and testing results. However, if the OMT is rejected, the next step would be to identify an alternative hypothesis that can be accepted. This might be an iterative procedure, since many options are possible for the alternative model.

The complete procedure is:

1. Define the nominal functional and stochastic model (under $\mathcal{H}_0$)
2. Apply BLUE
3. Apply the overall model test (OMT)
4. If OMT accepted go to step 8; if OMT rejected go to step 5
5. Choose (another) specific alternative hypothesis
6. Apply generalized likelihood ratio test (GLRT) to decide whether the alternative hypothesis is indeed more likely
7. If alternative hypothesis accepted go to step 8; if rejected go back to step 5
8. Present estimates, precision and testing results

See the examples of common alternative hypotheses.

# Outlier testing

Note that if you want to test for an outlier in an individual observation, there are in fact $m$ alternative hypotheses to be considered. In that case, in step 6 we would apply the test for all $m$ hypotheses and the one with the largest test statistic would be selected. However, it could be that there is more than one outlier. Therefore we would remove the outlying observation and go back to step 1. This procedure is repeated until no more outliers can be identified.

**Exercise**

Why would you select the alternative model with the largest test statistic?

**Solution**

The test statistic is given by $T_q = \hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} - \hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a$. Hence,the alternative hypothesis with largest $T_q$ must have the smallest weighted squared norm of residuals $\hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a$, and thus the best fit in a weighted least-squares sense.

Fig. 3.17 shows an example where on the left a linear trend is fitted to four observations without outliers. Here the overall model test would be accepted. On the right, the same example but with a bias of 10 mm added to the third observation. The bias has a large impact: the fitted line is pulled upwards with a much steeper slope. Consequently, the

residuals of three observations become very large, which would result in a rejection of the overall model test. The green line shows the fitted model after removing the third observation.

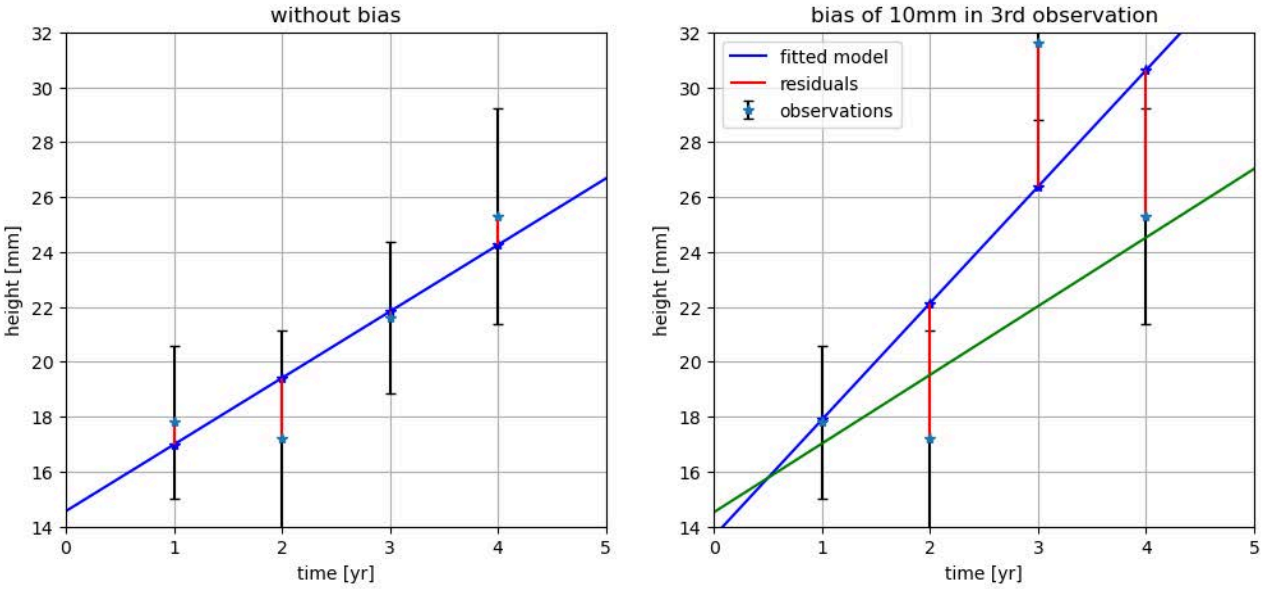



*Fig. 3.17* Linear trend fitted to a set of four observations without (left) and with one outlier (right).

# Model selection

In some cases, you may immediately want to test between two competing models, for instance if you are interested in the height change of a point over time you may want to test a linear trend model (constant velocity) versus a model including a constant acceleration. In that case, the procedure would be:

1. Define the functional and stochastic models (under $\mathcal{H}_0$ and $\mathcal{H}_a$)
2. Apply BLUE for both models
3. Apply the generalized likelihood ratio test (GLRT)
4. Present estimates, precision and testing results for accepted model

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Notebook exercise: which melting model is better?

## Contents


- Plot with observations and the fitted models
- Apply the generalized likelihood ratio test
- Challenge:


In this interactive notebook two models are fitted to a time series of height observations of a point on a glacier.

It is the same problem as we used in the notebook in the chapter on Weighted least-squares.

You will apply a statistical test to decide between two different models to describe the ice melt as function of time.

Click 🚀 –> **Live Code** on the top right corner of this screen and then wait until all cells are executed.

You have 12 monthly measurements of the height of a point on a glacier. The measurements are obtained from a satellite laser altimeter.

- Time [months]: `t` $= [0, 1, 2, \ldots, 11]$
- Observed heights [meters]: `y` $= [102.4, 98.2, 97.5, 97.9, 99.7, 100.7, 98.3, 94.2, 90.9, 86.1, 81.2, 76.9]$

We will consider three different models, with the following observation equations:

- Model 1: constant velocity, $\mathbb{E}\left(Y_i\right) = x_0 + x_1 t_i$
- Model 2: constant velocity + annual signal, $\mathbb{E}\left(Y_i\right) = x_0 + x_1 t_i + x_2 cos\left(\frac{2\pi t_i}{12}\right)$

The observation are independent and have a precision of 1 m.

**The first part of the code is hidden in the interactive version in the book.**

# Plot with observations and the fitted models

The squared norm of residuals is equal to $\hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon}$.

# Apply the generalized likelihood ratio test

The null hypothesis is that the linear trend model is applicable; the alternative hypothesis is that the linear trend + annual signal model is correct. Implement the test to decide whether the null hypothesis is accepted, or rejected in favor of the alternative hypothesis. Use a false alarm probability of 0.025.

## Generalized likelihood ratio test:

Accept alternative hypothesis and reject null hypothesis if: $T_q = \hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon} - \hat{\epsilon}_a^T \Sigma_Y^{-1} \hat{\epsilon}_a > k_\alpha$

First calculate the threshold value `k` $= k_\alpha$. Recall that $T_q \sim \chi^2(q, 0)$, where $q$ is the number of *extra* parameters in the alternative hypothesis. You can use chi2.ppf from SciPy.

Then calculate the test statistic value `Tq`. The squared norms of residuals of model 1 and 2 have already been calculated above (code is hidden) and have variable names `eTe_1` and `eTe_2`. You will also need to add the 'test' in the `if` statement.

```
alpha = 0.025
k = ?
```

```
Tq = ?

if ?:
    print(f'Test statistic = {Tq:.2f}\n')
    print(f'Threshold value = {k:.2f}\n')
    print('Null hypothesis rejected, Alternative hypothesis accepted')
else:
    print(f'Test statistic = {Tq:.2f}\n')
    print(f'Threshold value = {k:.2f}\n')
    print('Null hypothesis accepted')
```

**As we could have guessed from the figure, model 2 (linear trend + annual signal) provides a much better fit. In many practical situations it may be less obvious whether a model fits significantly better than the null hypothesis, and the general likelihood test provides a statistical test to assess this.**

# Challenge:

Try to come up with another model with at most 4 parameters, and then:

1. set-up the $\mathbb{A}$-matrix,
2. apply BLUE,
3. compute the sum of squared residuals $\hat{\epsilon}^T \Sigma_Y^{-1} \hat{\epsilon}$ for that model
4. apply the GLRT with model 2 as null hypothesis, and your model as alternative hypothesis.

**Solutions**

```
k = chi2.ppf(1 - alpha, 1)

Tq = eTe_1 - eTe_2

Tq > k:
```

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Notebook exercises: is my null hypothesis good enough?

## Contents


- Case 0: linear trend model is correct
- Case 1: linear trend model, outlier in data
- CASE 2: linear trend model is too simplistic
- CASE 3: linear trend model, too optimistic about precision


Click 🚀 –> Live Code on the top right corner of this screen and then wait until all cells are executed.

If you want to download the notebook, select the .zip after clicking on 📥.

In the following exercise, we will look at the curve fitting problem to a synthetic sea level rise dataset. The dataset contains yearly sea level measurements over 20 years (in total 20 observations). The time of observations are given in years as [1, 2, 3, …, 20]. The observations are assumed to be normally distributed and have a precision of $\sigma = 2$ cm. The sea level at time zero is unknown and should also be estimated together with the other parameters.

**In this exercise, we will:**

1. Import the data, create the design matrix $\mathrm{A}$, and the covariance matrix $\Sigma_Y$ (assuming the linear trend model).
2. Apply BLUE and overall model test to check whether the null hypothesis (linear trend model) is sufficiently supported by the data.

**We will look at 3 cases where the null-hypothesis is rejected:**

1. Dataset is corrupted by one large outlier.
2. We apply an incorrect functional model.
3. We use a too optimistic value for the precision.

If we detect that the overall model test (i.e., our null hypothesis) is rejected, we would in practice need to apply the Generalized Likelihood Ratio test to test an alternative hypothesis against the null hypothesis, to decide whether it is sufficiently more likely.

# Case 0: linear trend model is correct

First, import the data and compute the number of observations `m`, vector with observation times `t`, standard deviation of the measurements `std_y`, covariance matrix of observations `Sigma_Y`, design matrix `A`.

```
y_0 = np.genfromtxt('./data/Dataset_0.txt')
m = len(y_0)
t = np.arange(1,m+1)

std_y = 2
Sigma_Y = std_y**2 * np.eye(m)
A = np.column_stack((np.ones(m), t))
```

The function `plot_results` will perform BLUE and the overall model test with a false alarm probability `alpha`, and then plot results with confidence intervals with confidence level `conf_level`:

```
plot_results(t, A, y_0, Sigma_Y, conf_level = 0.95, alpha = 0.05)
```

# Case 1: linear trend model, outlier in data

In Dataset_1 one of the observations is an outlier… We will apply BLUE and perform the overall model test with this new dataset.

```
y_1 = np.genfromtxt('./data/Dataset_1.txt')
plot_results(t, A, y_1, Sigma_Y, conf_level = 0.95, alpha = 0.05)
```

**Question**

Why are more residuals negative in this case?

**Answer**

The fitted line is pulled towards the large outlier, and therefore more observations will lie below.

# CASE 2: linear trend model is too simplistic

For Dataset_2 the linear trend model assumptions is not correct… We will apply BLUE and perform the overall model test with this new dataset.

```
y_2 = np.genfromtxt('./data/Dataset_2.txt')
plot_results(t, A, y_2, Sigma_Y, conf_level=0.95, alpha = 0.05)
```

**Questions**

How can you see from the residuals plot that the linear model is too simplistic?

What would be a better model? Try yourself by completing the code below.

**Answers**

There is a clear 'non-random' pattern over time.

A second-order polynomial. You can try by inserting the corresponding $\mathrm{A}$-matrix `A_2` in the next code cell.

```
A_2 = ?
plot_results(t, A_2, y_2, Sigma_Y, conf_level=0.95, alpha = 0.05)
```

**Solution**

```
A_2 = np.column_stack((np.ones(m), t, t**2))
```

# CASE 3: linear trend model, too optimistic about precision

The observations in Dataset_3 are less precise than before… We will apply BLUE and perform the overall model test with this new dataset.

```
y_3 = np.genfromtxt('./data/Dataset_3.txt')
plot_results(t, A, y_3, Sigma_Y, conf_level=0.95, alpha = 0.05)
```

**Question**

How can you see that we were too optimistic about the precision of the observations? Play with the factor below to see if indeed the model is accepted if we choose a larger value for the precision.

**Solution**

Much more than 5% of the residuals is outside the 95% confidence interval, which is determined by the assumed precision of 2 cm. If we would use `2 * Sigma_Y` we would see that the model is accepted (hence the precision should have been $2\sqrt{2}$ cm).

```
factor = 1
plot_results(t, A, y_3, factor * Sigma_Y, conf_level=0.95, alpha = 0.05)
```

**Attribution**

This chapter was written by Sandra Verhagen. Find out more here.

# Notation and formulas

## Contents


- Estimators of $\mathrm{x}$
- Test statistics (with distribution under $\mathcal{H}_0$)
- Linear propagation laws if $\hat{X} = \mathrm{L}^{\mathrm{T}} Y$


**MUDE exam information**

This overview with Notation and Formulas will be given on the MUDE exam.

| Description | Notation / formula |
|---|---|
| Observables (random) | $Y = [Y_1, Y_2, \ldots, Y_m]^T$ |
| Observations (realization of $Y$) | $\mathrm{y} = [y_1, y_2, \ldots, y_m]^T$ |
| Unknown parameters (deterministic) | $\mathrm{x} = [x_1, x_2, \ldots, x_n]^T$ |
| Random errors | $\epsilon = [\epsilon_1, \epsilon_2, \ldots, \epsilon_m]^T$ with $\epsilon \sim N(0, \Sigma_\epsilon)$ |
| Functional model (linear) | $\mathbb{E}(Y) = \mathrm{A}\mathrm{x}$ or $Y = \mathrm{A}\mathrm{x} + \epsilon$ |
| Stochastic model | $\mathbb{D}(Y) = \Sigma_Y = \Sigma_\epsilon$ |
| Estimator of $\mathrm{x}$ | $\hat{X}$ |
| Estimate of $\mathrm{x}$ (realization of $\hat{X}$) | $\hat{\mathrm{x}}$ |
| Adjusted (or predicted) observations | $\hat{\mathrm{y}} = \mathrm{A}\hat{\mathrm{x}}$ |
| Residuals | $\hat{\epsilon} = \mathrm{y} - \mathrm{A}\hat{\mathrm{x}}$ |

# Estimators of x

| Estimators | Formula |
|---|---|
| Least-squares (LS) | $\hat{X} = \left(\mathrm{A}^T\mathrm{A}\right)^{-1}\mathrm{A}^T Y$ |
| Weighted least-squares (WLS) | $\hat{X} = \left(\mathrm{A}^T\mathrm{W}\mathrm{A}\right)^{-1}\mathrm{A}^T\mathrm{W}Y$ |
| Best linear unbiased (BLU) | $\hat{X} = \left(\mathrm{A}^T\Sigma_Y^{-1}\mathrm{A}\right)^{-1}\mathrm{A}^T\Sigma_Y^{-1}Y$ |

# Test statistics (with distribution under $\mathcal{H}_0$)

| Test statistic | Formula |
|---|---|
| Generalized likelihood ratio test | $T_q = \hat{\epsilon}^T\Sigma_Y^{-1}\hat{\epsilon} - \hat{\epsilon}_a^T\Sigma_Y^{-1}\hat{\epsilon}_a \sim \chi^2(q,0)$ |
| Overall model test | $T_{q=m-n} = \hat{\epsilon}^T\Sigma_Y^{-1}\hat{\epsilon} \sim \chi^2(q,0)$ |
| w-test | $W = \frac{\mathrm{C}^T\Sigma_Y^{-1}\hat{\epsilon}}{\sqrt{\mathrm{C}^T\Sigma_Y^{-1}\Sigma_{\hat{\epsilon}}\Sigma_Y^{-1}\mathrm{C}}} \sim N(0,1)$ |

# Linear propagation laws if $\hat{X} = \mathrm{L}^{\mathrm{T}}Y$

| Propagation law of the … | Formula |
|---|---|
| … mean | $\mathbb{E}(\hat{X}) = \mathrm{L}^{\mathrm{T}}\mathbb{E}(Y)$ |
| … covariance matrix | $\Sigma_{\hat{X}} = \mathrm{L}^{\mathrm{T}}\Sigma_Y\mathrm{L}$ |

> **Attribution**
>
> This chapter was written by Sandra Verhagen. Find out more here.

# Numerical Modelling

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

The Finite Difference Method (FDM) is a fundamental numerical approach used to solve differential equations that govern physical phenomena. These phenomena include heat flow through media, the distribution of stresses in solid structures, fluid flow through porous media, pollutant dispersion in the air and water, amongst others. Virtually, any physical phenomena can be tackled by solving numerically differential equations. The FDM provides engineers with the flexibility to model and analyze a complex geometries with varied past, present and future scenarios, optimize designs, ensure safety, reduce costs, and predict failures. Even more!

In this Chapter, the fundamentals of numerical modelling and basic numerical methods are treated. More advanced techniques are also revised. The main focus is on Ordinary Differential Equations of varying orders.

> **Attribution**
>
> This figure was made by Amgad Omer of Deltares. Find out more here.

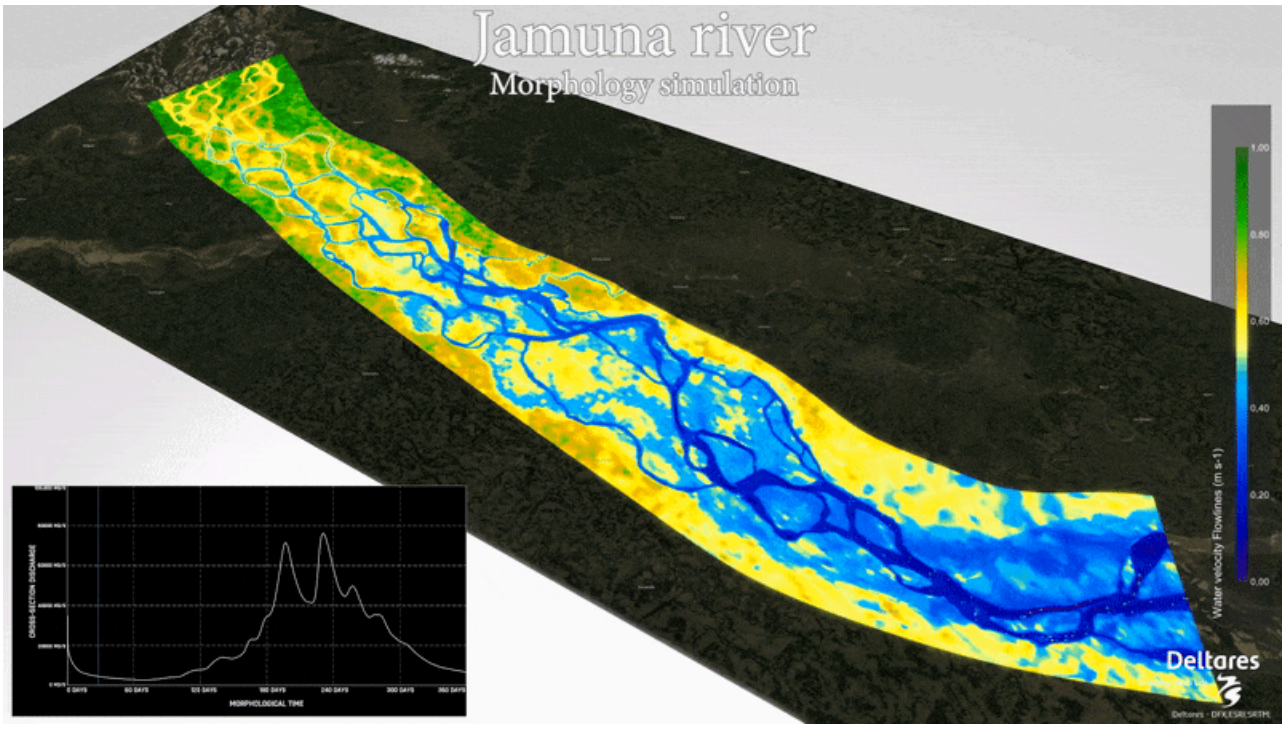



*Fig. 4.1* Cool numerical model of the Jamuna river. Included here with permission of Amgad Omer of Deltares. Find out more here

# Revision of Concepts

## Contents


- Classification of Differential Equations
- Analytical vs Numerical Solutions


## Classification of Differential Equations

> **Note**
>
> **Important things to retain from this block:**
>
> - Identify characteristics of differential equations
> - Understand how numerical and analytical solutions might differ as we get to more complex problems which need simplification
> - Remember that analytical solutions are not always possible

Differential equations can be classified as Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs). **ODEs** have derivatives with respect to a **single** independent variable (either time **or** space), for example:

$$\frac{dx(t)}{dt} = \cos t$$

describes the rate of change of the variable $x$ in **time** $t$.

**PDEs** have derivatives with respect to **multiple** independent variables (often time **and** space), for example:

$$\frac{\partial c(x,t)}{\partial t} + u\frac{\partial c(x,t)}{\partial x} = 0$$

describes the propagation of the concentration $c$ in **time** $t$ along **dimension** $x$.

The classification can be done more precisely by the equation's order and linearity (linear or non-linear). The order refers to the highest derivative while non-linear equations are those in which the dependent variable or its derivative(s) are non linear (exponential, sine or with a power different than 1). See the following examples:

$$\frac{d^3y}{dx^3} - x\frac{d^2y}{dx^2} + y = 0 \qquad \text{third-order linear ODE}$$

$$\frac{dy}{dx} = y^2 + x \qquad \text{first-order non-linear ODE}$$

$$\frac{d^2y}{dx^2} + y\left(\frac{dy}{dx}\right)^2 = \sin(y) \qquad \text{second-order non-linear ODE}$$

# Analytical vs Numerical Solutions

Equations can be solved in two ways: analytically and numerically. The analytical solution is **exact** while a numerical solution requires computational methods to **approximate** the solution. So why would we use anything other than analytical solutions? Well, analytical solutions are difficult/impossible to find for complex equations, specially when the problem involves a complex geometry. This complexity will be treated later in the book. For now, let's illustrate analytical and numerical solutions considering a simple problem.

> Find the value $x$ for $f(x) = 3x - 2$ when $f(x) = 0$ in the interval $[0, 1]$.

**Analytical Solution**

$$f(x) = 0 \Leftrightarrow 3x - 2 = 0 \Leftrightarrow x = \frac{2}{3} = 0.666666666666666..$$

Note that there is no need for any extra computation. You can say that there is only **one computation needed**: the assignment of the $x$ value.

**Numerical Solution**

The code below shows an iterative method to find the x value.

```python
def f(x):
    return 3*x-2

dx = 0.001
for i in range(1000):
    x = dx*i
    if f(x) * f(x+dx) < 0:
        print("Number of computations needed to find a solution",i*4)
        break

print("Answer x = ", x+dx)
```

```
Number of computations needed to find a solution 2664
Answer x =  0.667
```

Note that the 2664 computations needed is highly dependent on the method (this one is not very efficient). Here the search starts at x=0 and increases in steps of 0.001, which also limits the accuracy of the solution.

**Exercise**

Let us now look to another simple example:

> Find the value $x$ for $f(x) = 3\sin(x) - 2$ when $f(x) = 0$ in the interval $[0, 1]$.

**Analytical Solution**

$$f(x) = 0 \Leftrightarrow 3\sin(x) - 2 = 0 \Leftrightarrow x = \arcsin\left(\frac{2}{3}\right) = 0.7297276562269663..$$

**Numerical Solution**

```
import math

def f(x):
    return 3*math.sin(x)-2

dx = 0.001
for i in range(1000):
    x = dx*i
    if f(x) * f(x+dx) < 0:
        print("Number of computations needed to find a solution",i*4)
        break

print("Answer x = ", x+dx)
```

```
Number of computations needed to find a solution 2916
Answer x =  0.73
```

Note that there is no attempt to solve f(x) directly. Only the function is defined and the exact same steps of the previous problem are followed.

A better (more efficient) solution would be to use Newton's method, which would look like:

```
def df(x):
    dx = 1e-5
    return (f(x+dx)-f(x)) / dx


x = 0.5             # start halfway the search interval
i = 0
error = 1
while error > 1e-10:
    x_new = x - f(x) / df(x)
    error = abs(x_new - x)
    x = x_new
    i += 1
print("Answer x = ", x)
print("Number of iterations",i)
```

```
Answer x =  0.7297276562269663
Number of iterations 5
```

This algorithm requires the derivative of the function, which is approximated by considering a small interval dx around point x. The result is a significant reduction of the number of computations needed to get a solution that is also more accurate than the first algorithm.

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# The First Derivative

## Contents


- Numerical Freedom to compute derivatives


> **Note**
>
> **Important things to retain from this block:**
>
> - Understand what the derivative represents
> - Recognize that the derivative can be approximated in different ways

In your calculus classes, you might remember working with the analytical definition of a derivative, such as $\frac{df}{dx}$. This involved finding the limit as the distance between two points becomes infinitesimally small.

**Definition:**

The derivative of a function $f(x)$ evaluated at the point $x_0$ is

$$\left.\frac{df}{dx}\right|_{x_0} = f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}.$$

**Numerically:**

$x - x_0$ **cannot tend to 0!** Thus:

$$f'(x_0) \approx \frac{f(x) - f(x_0)}{\Delta x}, \quad \text{where } \Delta x = x - x_0.$$

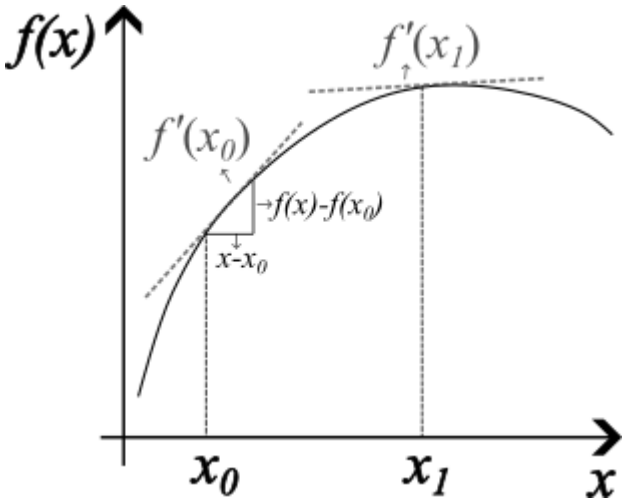



*Fig. 4.2* derivatives of a function $f(x)$ at a specific points $x_0$ and $x_1$

**What does the derivative represents?** A rate of change! In this case, how fast $f(x)$ changes with respect to $x$ (or in the direction of $x$). The derivative can also be with respect to time or another independent variable (e.g., $y, z, t, etc$). Look at the figure above. The first derivative evaluated at $x_0$ is illustrated as a simple slope, as is the first derivative evaluated at $x_1$ . You can also see that the rate of change at $x_0$ is larger than at $x_1$.

# Numerical Freedom to compute derivatives

In the Figure above, the derivative approximation was illustrated arbitarly using two points: the one at which the derivate was evaluated and another point in front of it. However, there are more possibilities. Instead of using absolute points at $x_{-1,0,1}$ a more general notation is used: $x_{i-1,i,i+1}$. The simplest ways to approximate the derivative evaluated point $x_i$ is to use two points:

forward: $\left.\frac{df}{dx}\right|_{x_i} \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$ backward: $\left.\frac{df}{dx}\right|_{x_i} \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$ central:

Note that the finite difference formula requires distances $x_{i+1} - x_i$ and $x_i - x_{i+1}$. This will be introduced in the next chapter ‘FDM intro’ and will become more evident in the ‘Taylor Series Expansion chapter’.

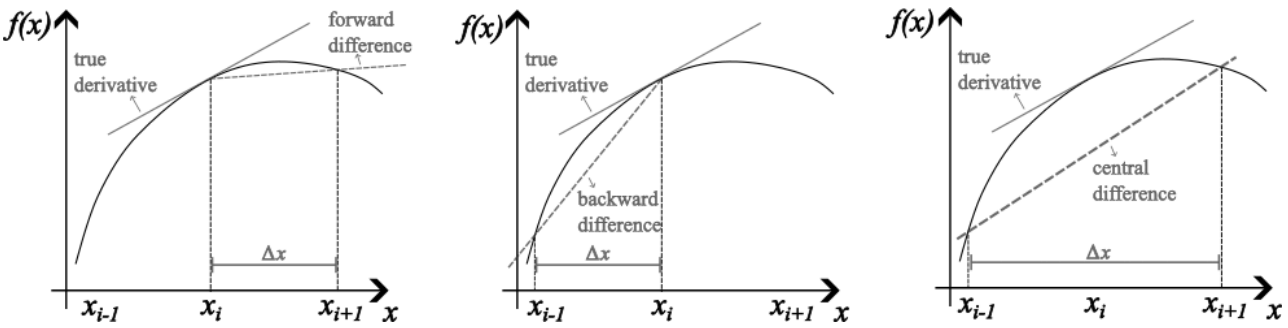



*Fig. 4.3* graphs illustrating three different ways to approximate derivatives.

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Finite Difference Method

The Finite Difference Method (FDM) is an indispensable numerical approach, which plays a fundamental role in solving differential equations that govern physical phenomena. In this method, the derivatives in the differential equation are approximated using numerical differences, just like the forward, backward and central differences treated in the previous chapters.

Consider the following equation that describes ice growth of an ice layer assuming that the air and water temperatures are constant:

$$\rho_{ice} \frac{dh_{ice}}{dt} = -k_{ice} \frac{T_{water} - T_{air}}{h_{ice}}$$

where $h_{ice}$ is the ice thickness, $\rho_{ice}$ ice density and $k_{ice}$ refers to the heat transfer capabilities of the ice.

The growth rate (derivative) of the ice thickness can be approximated with one of the numerical derivatives and the resulting equation can be solved for the ice thickness in the next time step. In essence, the finite difference method transforms a differential equation into an algebraic form (discretization).

Still a definition of the solution domain is needed. For this example, an inital and an end time $[a, b]$, where $a$ and $b$ are called **end points**. The domain is represented as **grid points** and therefore in a number of subintervals $N$. The length of the subinterval is $h = (b - a)/N$.

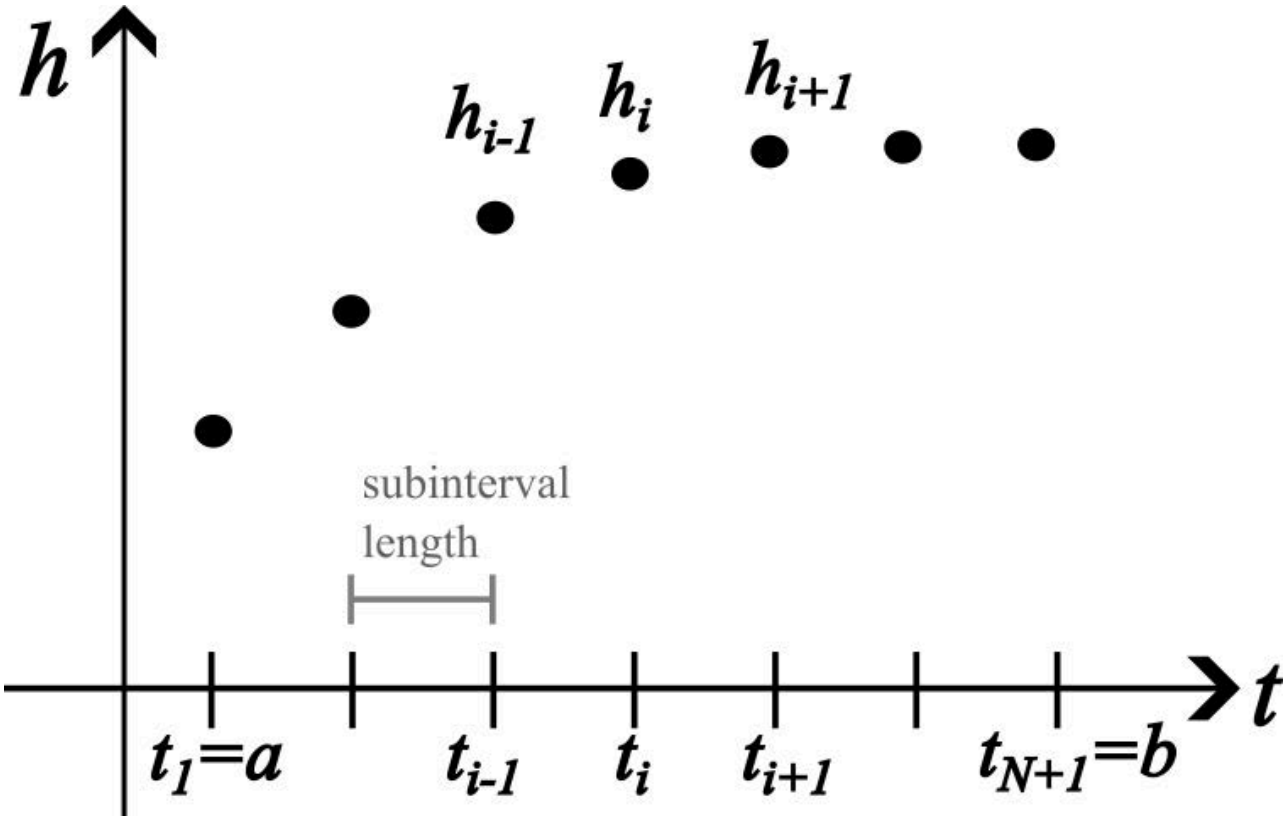



*Fig. 4.4* Discretized differential equation ``

The figure above shows the grid of which the following property holds:

$$a = t_1 < t_2 <.. < t_N < t_{N+1} = b$$

The Points $t_i$ on the grid are called **nodes**. Just note that your indexing is a choice. For example $i$ can be for $i = (1, 2, \ldots, N, N+1)$ as shown in the figure above. The indexing can also start at 0 where $i = (0, 1, \ldots, N-1, N)$.

Lets simplify the above ice equation and put it in a standard ODE form:

$$\frac{dh}{dt} = f$$

We can rewrite this equation in a discretized form:

$$h_i' = f_i$$

Note that $h_i'$ is the first derivative at node $i$ and at that node is equal to function $f_i$. Using FDM we numerically compute the solution $h_i$ at each node $t_i$ The connection between the derivative $h_i'$ and $t_i$ can be made by using **Taylor Series Expansions**.

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Taylor Series Expansion

## Contents


- Definition
- Use of TSE to define the first derivative
- TSE to define second derivatives
- Higher-accuracy Finite-Difference Approximations


> **Note**
>
> **Important things to retain from this block:**
>
> - Understand how to compute Taylor series expansion (TSE) of a given function around a given point and its limitations
> - Understand how numerical derivatives are derived from TSE
> - Understand that the magnitude of the error depends on the expression used
> - **Things you do not need to know:**
> - Any kind of Taylor expansion of a function by heart

## Definition

The Taylor Series Expansion (TSE) of an arbitrary function $f(x)$ around $x = x_i$ is a polynomial of varying order given by

$$f(x) = f(x_i) + (x - x_i)f'(x_i) + \frac{(x - x_i)^2}{2!}f''(x_i) + \frac{(x - x_i)^3}{3!}f'''(x_i) + \dots$$

This series is **exact** as long as we include infinite terms. We, however, are limited to a **truncated** expression: an **approximation** of the real function.

Using only 3 terms and defining $\Delta x = x - x_i$, the TSE can be rewritten as

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \frac{\Delta x^3}{3!} f'''(x_i) + \mathcal{O}(\Delta x)^4.$$

Here $\Delta x$ is the distance between the point we "know" and the desired point. $\mathcal{O}(\Delta x)^4$ means that we do not take into account the terms associated to $\Delta x^4$ and therefore **that is the truncation error order**. From here we can also conclude that **the larger the step $\Delta x$, the larger the error**!

**Tip**

We will use $\Delta x$ more frequently from this point on, so it is good to recognize now, using the equations above, that it is a different way of representing the differential increment, for example, $f(x_{i+1}) = f(x_i + \Delta x)$ or $f(x_{i+2}) = f(x_i + 2\Delta x)$.

Now let's see an example.

---

**Example**

Compute $e^{0.2}$ using 3 terms of TSE around $x_i = 0$

**Solution**

We want to evaluate $e^x = e^{x_i + \Delta x} = e^{0.2}$. Therefore, $\Delta x = 0.2$. The value of $f(x_i = 0) = e^0 = 1$. For the case of this exponential, the derivatives have the same value $f'(x_i = 0) = f''(x_i = 0) = f'''(x_i = 0) = e^0 = 1$. The TSE looks like

$$e^{x_i + \Delta x} \approx e^0 + \Delta x e^0 + \frac{\Delta x^2}{2!} e^0 + \frac{\Delta x^3}{3!} e^0 \approx 1 + \Delta x + \frac{\Delta x^2}{2} + \frac{\Delta x^3}{6} \approx 1 +$$

Compute the TSE polynomial truncated until $\mathcal{O}(\Delta x)^6$ for $f(x) = \sin(x)$ around $x_i = 0$

**Solution**

Applying the definition of TSE:

$$\sin(x) \approx \sin(x_i + \Delta x) \approx \sin(0) + x\cos(0) - \frac{x^2}{2}\sin(0) - \frac{x^3}{6}\cos(0) + \frac{x^4}{24}$$

$$\sin(x) \approx x - \frac{x^3}{6} + \frac{x^5}{120} \mathcal{O}(\Delta x)^6 \text{error}$$

Note that $x$ and $\Delta x$ can be written interchangeably when $x_i = 0$ as $\Delta x = x - x_i$. If this would not be the case, for example if $x_i = \pi$ the result is completely different.

Compute the TSE polynomial truncated until $\mathcal{O}(\Delta x)^6$ for $f(x) = \sin(x)$ around $x_i = \frac{\pi}{2}$

**Solution**

Applying the definition of TSE:

$$\sin(x) \approx \sin(x_i + \Delta x) \approx \sin(\frac{\pi}{2}) + (x - \frac{\pi}{2})\cos(\frac{\pi}{2}) - \frac{(x - \frac{\pi}{2})^2}{2}\sin(\frac{\pi}{2}) \cdot$$

$$\sin(x) \approx \sin(x_i + \Delta x) \approx 1 - \frac{(x - \frac{\pi}{2})^2}{2} + \frac{(x - \frac{\pi}{2})^4}{24} \mathcal{O}(\Delta x)^6 \text{error}$$

How good is this polynomial? How does the result varies on the number of terms used?

Press `rocket` –> `Live Code` to interact with the figure. You should then click 'run all' in the cell below.

```
#run to interact with the figure
interact(taylor_plt, order_aprox = widgets.ToggleButtons(options=['1st
```

run | run all | add cell | clear

Relevant conclusions:

- The 1st order, which depends only on the first derivative evaluation, is a straight line.
- The more terms used (larger order) the smaller the error.
- The further from the starting point (e.g., in the plots $x_i = 0$), the larger the error.

# Use of TSE to define the first derivative

The TSE definition when $\Delta x = x_{i+1} - x_i$ can be rewritten to solve for the first derivative:

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} - \frac{\Delta x}{2!} f''(x_i) - \frac{\Delta x^2}{3!} f'''(x_i) - \dots$$

By truncating the derivative to avoid the calculation of the second derivative, we find the **forward difference**

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} + \mathcal{O}(\Delta x).$$

This is the same definition of the numerical derivative! Now we have the added knowledge that this comes with an error of the order of the step.

The **backward difference** can be found by redefining $-\Delta x = x_i - x_{i-1}$

$$f'(x_i) = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + \frac{\Delta x}{2!} f''(x_i) - \frac{\Delta x^2}{3!} f'''(x_i) - \dots$$

$$f'(x_i) = \frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} + \mathcal{O}(\Delta x).$$

The **central difference** can be found by summing the forward and backward difference expressions of the derivative and dividing it by 2:

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} - \frac{\Delta x^2}{3!} f'''(x_i) - \dots$$

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2).$$

The second derivative terms cancel each other out, therefore **the order error is the step size squared!** From here, it is obvious that the central difference is more accurate. You can notice it as well intuitively in the figure of the previous chapter.

**Exercises**

Derive the forward difference for the first derivative $f'(x_i)$ with a first order error $\mathcal{O}(\Delta x)$ using Taylor series

**Solution**

Start from the TSE (tip: first derivative and first order error (1+1=2), we need to do a TSE truncated until the 2nd order):

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \mathcal{O}(\Delta x)^2$$

Rearange to get:

$$-\Delta x f'(x_i) = -f(x_i + \Delta x) + f(x_i) + \mathcal{O}(\Delta x)^2$$

Divide by $-\Delta x$:

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} + \mathcal{O}(\Delta x)$$

Use taylor series to derive the backward difference for the first derivative $f'(x_i)$ with a first order error $\mathcal{O}(\Delta x)$

**Solution**

Start from the TSE

(tip: first derivative and first order error, 1+1=2, we need to do a TSE truncated until the 2nd order):

$$f(x_i - \Delta x) = f(x_i) - \Delta x f'(x_i) + \mathcal{O}(\Delta x)^2$$

Rearrange to get:

$$\Delta x f'(x_i) = -f(x_i - \Delta x) + f(x_i) + \mathcal{O}(\Delta x)^2$$

Divide by $\Delta x$:

$$f'(x_i) = \frac{f(x_i) - f(x_i + \Delta x)}{\Delta x} + \mathcal{O}(\Delta x)$$

Use taylor series to derive the Central difference for the first derivative $f'(x_i)$ with a 2nd order error $\mathcal{O}(\Delta x)^2$

**Solution**

Start from the TSE of $f(x_i + \Delta x)$ and $f(x_i - \Delta x)$

(tip: first derivative and second order error (1+2=3), we need to do a TSE truncated until the 3rd order):

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2} + f''(x_i)\mathcal{O}(\Delta x)^3 \quad (1)$$

$$f(x_i - \Delta x) = f(x_i) - \Delta x f'(x_i) + \frac{\Delta x^2}{2} - f''(x_i)\mathcal{O}(\Delta x)^3 \quad (2)$$

Rearrange both equations:

$$\Delta x f'(x_i) = f(x_i + \Delta x) - f(x_i) - \frac{\Delta x^2}{2} f''(x_i) - f'''(x_i)\mathcal{O}(\Delta x)^3 \quad (3)$$

$$\Delta x f'(x_i) = -f(x_i - \Delta x) + f(x_i) + \frac{\Delta x^2}{2} f''(x_i) - f'''(x_i)\mathcal{O}(\Delta x)^3 \quad (4)$$

Take the combination $2\Delta x f'(x_i) + \Delta x f'(x_i)$ by adding equation 3 and 4:

$$2\Delta x f'(x_i) = f(x_i + \Delta x) - f(x_i - \Delta x) - f'''(x_i)\mathcal{O}(\Delta x)^3 \quad (4)$$

Divide by $2\Delta x$ :

$$f'(x_i) = \frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} - f''(x_i)\mathcal{O}(\Delta x)^2 \quad (4)$$

# TSE to define second derivatives

There are equations that require second derivatives. One example is the diffusion equation. The 1-D diffusion equation reads:

$$\frac{\partial f}{\partial t} = v\frac{\partial^2 f}{\partial x^2}$$ where $v$ is the diffusion coefficient.

For the moment we will use TSE to find **only** a numerical expression of the second derivative $\frac{\partial^2 f}{\partial x^2}$.

The procedure is simple but cumbersome. The general idea is to isolate the second derivative in the TSE without there being a dependency on other derivatives. Below you can find more details about the algebraic manipulation (if you are curious) but you do not need to know it. Here is the result:

$$f''(x_i) = \frac{f(x_i + 2\Delta x) - 2f(x_i + \Delta x) + f(x_i)}{\Delta x^2} + \mathcal{O}(\Delta x).$$

This is the **forward** difference approximation of the second derivative. Two aspects come to mind: one more point is needed to compute the second derivative and the error (of the simplest second derivative) is also of the order of the step. There are also **backward** and **central** approximations of the second derivative (not shown here).

**Derivation of the forward difference approximation of the second derivative**

First define a TSE for a point two steps farther from $x_i$:

$$f(x_i + 2\Delta x) = f(x_i) + 2\Delta x f'(x_i) + \frac{(2\Delta x)^2}{2!}f''(x_i) + \mathcal{O}(\Delta x)^3.$$

Now multiply the TSE by two for a point one step farher from $x_i$:

$$2f(x_i + \Delta x) = 2f(x_i) + 2\Delta x f'(x_i) + \frac{2\Delta x^2}{2!}f''(x_i) + \mathcal{O}(\Delta x)^3.$$

By substracting the first expression from the second one the first derivative dissapears:

$$f(x_i + 2\Delta x) - 2f(x_i + \Delta x) = -f(x_i) + \frac{2\Delta x^2}{2!}f''(x_i) + \mathcal{O}(\Delta x)^3.$$

By solving for $f''$ we obtain the **forward** expression:

$$f''(x_i) = \frac{f(x_i + 2\Delta x) - 2f(x_i + \Delta x) + f(x_i)}{\Delta x^2} + \mathcal{O}(\Delta x).$$

# Higher-accuracy Finite-Difference Approximations

So far we have found expressions with a relative large magnitude error $\mathcal{O}(\Delta x)$ with the exception of the central difference approximation. Sometimes a higher accuracy is desired for which better expressions can be found. The procedure is similar to the algebraic manipulation to find the **forward** approximation of the second derivative: a series of TSE are defined at varying distances from $x_i$ and after algebraic manipulation a more accurate expression is found. For example, the **forward** approximation of the first derivative:

$$f'(x_i) = \frac{-f(x_i + 2\Delta x) + 4f(x_i + \Delta x) - 3f(x_i)}{2\Delta x} + \mathcal{O}(\Delta x^2).$$

The error magnitude has improved to $\mathcal{O}(\Delta x^2)$ at the expense of using one more point. The accuracy can be even better by using more points. It is important to note that central differences are more accurate than forward and backward differences when using the same number of points.

**Exercise**

Derive a first derivative $f('x)$ with an 2nd error order $\mathcal{O}(\Delta x^2)$ finite-difference equations, using the following nodes $x_i$, $x_i + \Delta x$ and $x_i + 2\Delta x$ or $x_i$, $x_{i+1}x$ and $x_{i+2}$

The finite-difference equations will have the following form:

$$f'(x_i) = \frac{\alpha f(x_i) + \beta f(x_i + \Delta x) + \gamma f(x_i + 2\Delta x)}{\Delta x} + \mathcal{O}(\Delta x^2)$$

Use the Taylor series to find $\alpha$, $\beta$ and $\gamma$

**Solution**

Taylor series for $f(x_i + \Delta x)$ and $f(x_i + 2\Delta x)$ (Tip: first order derivative and second error order, 1+2=3, meaning a truncation until 3rd order ):

$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{(\Delta x)^2}{2!} f''(x_i) + \mathcal{O}(\Delta x)^3.$$

$$f(x_i + 2\Delta x) = f(x_i) + 2\Delta x f'(x_i) + \frac{(2\Delta x)^2}{2!} f''(x_i) + \mathcal{O}(\Delta x)^3.$$

Times $f(x_i + \Delta x)$ by 4 and expand the term $\frac{(2\Delta x)^2}{2!} f''(x_i)$ in TSE $f(x_i + 2\Delta x)$:

$$4f(x_i + \Delta x) = 4f(x_i) + 4\Delta x f'(x_i) + 2(\Delta x)^2 f''(x_i) + 4\mathcal{O}(\Delta x)^3.$$

$$f(x_i + 2\Delta x) = f(x_i) + 2\Delta x f'(x_i) + 2(\Delta x)^2 f''(x_i) + \mathcal{O}(\Delta x)^3.$$

Now take $4f(x_i + \Delta x) - f(x_i + 2\Delta x)$:

$$4f(x_i + \Delta x) - f(x_i + 2\Delta x) = 4f(x_i) + 4\Delta x f'(x_i) + 2(\Delta x)^2 f''(x_i) + 4\mathcal{O}$$

$$= 3f(x_i) + 2\Delta x f'(x_i) + 3\mathcal{O}(\Delta x)^3$$

Rearrange for $f'(x_i)$:

$$-2\Delta x f'(x_i) = 3f(x_i) - 4f(x_i + \Delta x) + f(x_i + 2\Delta x) + 3\mathcal{O}(\Delta x)^3$$

Divide by $-2\Delta x$:

$$f'(x_i) = \frac{-\frac{3}{2} f(x_i) + 2f(x_i + \Delta x) - \frac{1}{2} f(x_i + 2\Delta x)}{\Delta x} + \mathcal{O}(\Delta x)^2$$

***The solution is:*** $\alpha = -\frac{3}{2}, \beta = 2, \gamma = -\frac{1}{2}$

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Exercises on Taylor expansion

## Contents


- Exercise 1
- Exercise 2
- Exercise 3


Use this page to practice calculating Taylor expansions. This requires

> **Tip**
>
> This page presents several functions of $x$ that you should approximate with Taylor Series. You can click `rocket` –> `Live Code` to check your answers, however, note the following:
>
> 1. You should work out the exercises “by hand” first (using paper or a digital notebook)
> 2. You can type your answer in the code cell and then click “Check Answer” to see if it is correct. Use basic Python syntax for representing the equation, for example, `x**2 + 1` for $x^2 + 1$; use `pi` or `sin()` and `cos()` for the trigonometric functions (not numpy!)
> 3. You can click the “Show correct answer” button to see the solution; however, note that the form of the equation will be different than that which you are expected to derive, so you will need to do some arithmetic to confirm you are correct.
>
> *The reason the Python syntax is different is because it uses Sympy, a library for symbolic mathematics.*

# Exercise 1

Calculate the taylor series expansion of:

$2x^2 - 2x$

around:

$x = 2$

Discard any $O(x^3)$ terms.

Fill in your answer and run the cell before clicking ‘Check answer’.

```
1 eq1_answer =
```

run | run all | add cell | clear

Check answer

Show correct answer

**Tip**

Note that for Exercise 1, the equation is an even-order polynomial, which means that the Taylor Series should not introduce any error!

# Exercise 2

Calculate the taylor series expension of:

$5\tan(x)$

around:

$x = -5\pi$

discard any $O(x^3)$ terms.

Fill in your answer and run the cell before clicking 'Check answer'. Furthermore, use `pi` for $\pi$:

```
1 eq2_answer =
```

run | run all | add cell | clear

Check answer

Show correct answer

# Exercise 3

Calculate the taylor series expension of:

$$\frac{1}{1-x}$$

around:

$$x = -1$$

discard any $O(x^3)$ terms.

Fill in your answer and run the cell before clicking ‘Check answer’:

```
1 eq3_answer =
```

run | run all | add cell | clear

Check answer

Show correct answer

**Attribution**

This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Numerical Integration

## Contents


- Motivation
- Discrete Integration


NOTE

> **Note**
>
> **Important things to retain from this block:**
>
> - Know how to apply different discrete integration techniques
> - Have an idea on how the truncation errors change from technique to technique

## Motivation

Numerical integration is used often in diverse fields of engineering. For example to develop advanced numerical model techniques or to analyze field or laboratory data where a desired physical quantity is an integral of other measured quantitites. So, the analytic integral is not always known and has to be computed (approximated) numerically. For example, the variance density spectrum (in ocean waves) is a discrete signal obtained from the water level variations and its **integral** is a quantity that characterizes the wave field.

## Discrete Integration

In a 1 dimensional situation, the integral:

$$I = \int_a^b f(x)dx$$

is defined between points and represents the area under the curve $f$ limited by $[a, b]$. A visual way to approximate that area would be to use a squared page and count the number of squares that fall under that curve. In the example figure, you can count 11 squares.

**The numerical integration is an approximation** to that area done in a strict way. There are several paths to follow using polynomial representation. These widely used formulas are known as Newton-Cotes formulas.

- First, we split the interval of integration (a,b) on several partial intervals.
- Then, we replace $f(x)$ by the interpolation polynomial in each interval and calculate the integral over it.
- Finally, sum all the areas to find (approximate) the integral.

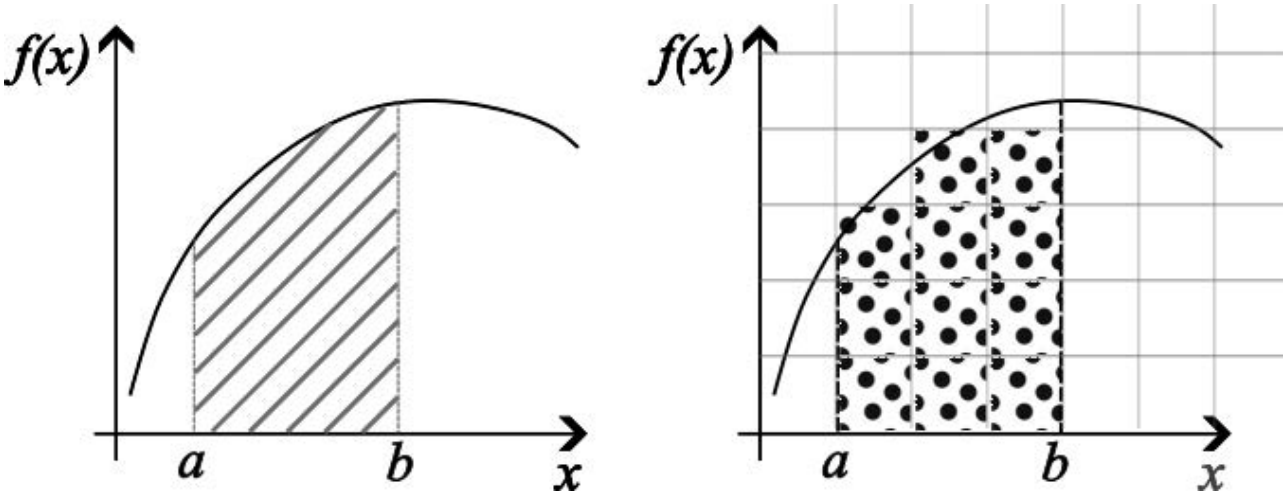



*Fig. 4.5* two integral illustration showing the area under the curve for interval [a,b]

# Left Riemann Integration

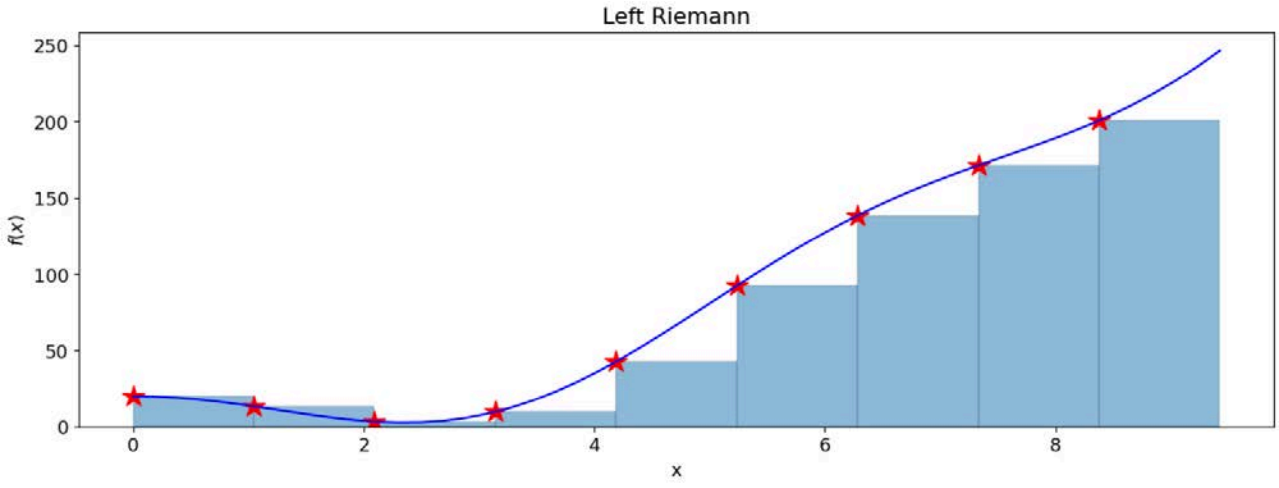



*Fig. 4.6* Illustrating the left Riemann Sum over interval [a,b]

Using the lowest polynomial order leads to the Riemann integral also known as the rectangle rule. Basically, the function is divided in segments with a known $\Delta x$ and the area of each segment is represented by a rectangle. Its height, related to the discretized points of the function, can be defined by the left or by the right.

Therefore, we get the left Riemann integration formula given by:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f(x_i)\Delta x, \quad \Delta x = \frac{b-a}{n}$$

where $n$ is the number of segments.

# Right Riemann Integration

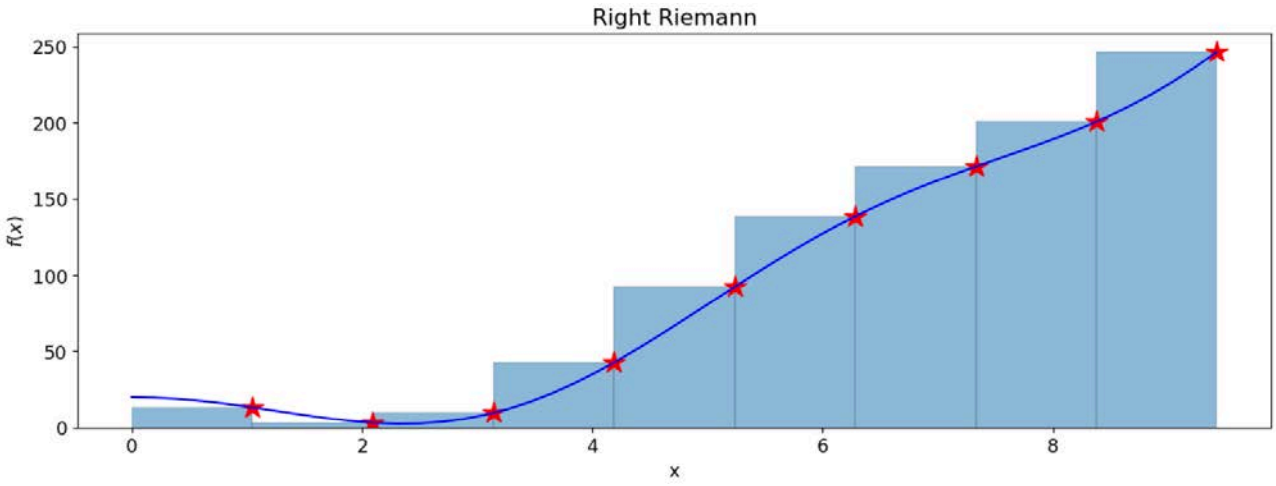



*Fig. 4.7* Illustrating the right Riemann Sum over interval [a,b]

In an analogous way, the formula for the right Riemann integration is:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i)\Delta x \approx \sum_{i=0}^{n-1} f(x_{i+1})\Delta x$$

Note the difference respect to the left Riemann integration, specially the starting and end points.

# Midpoint Rule

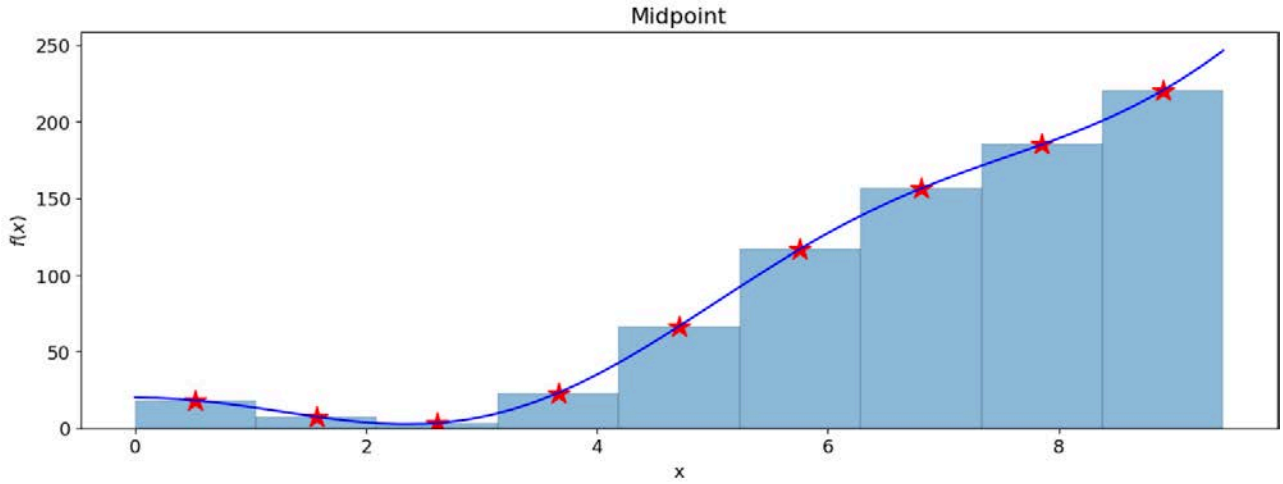



*Fig. 4.8* Illustrating the right Midpoint Rule over interval [a,b]

The midpoint rule defines the height of the rectangle at the midpoint between the two end points of each interval:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f(\frac{x_i + x_{i+1}}{2})\Delta x$$

# Trapezoidal Rule

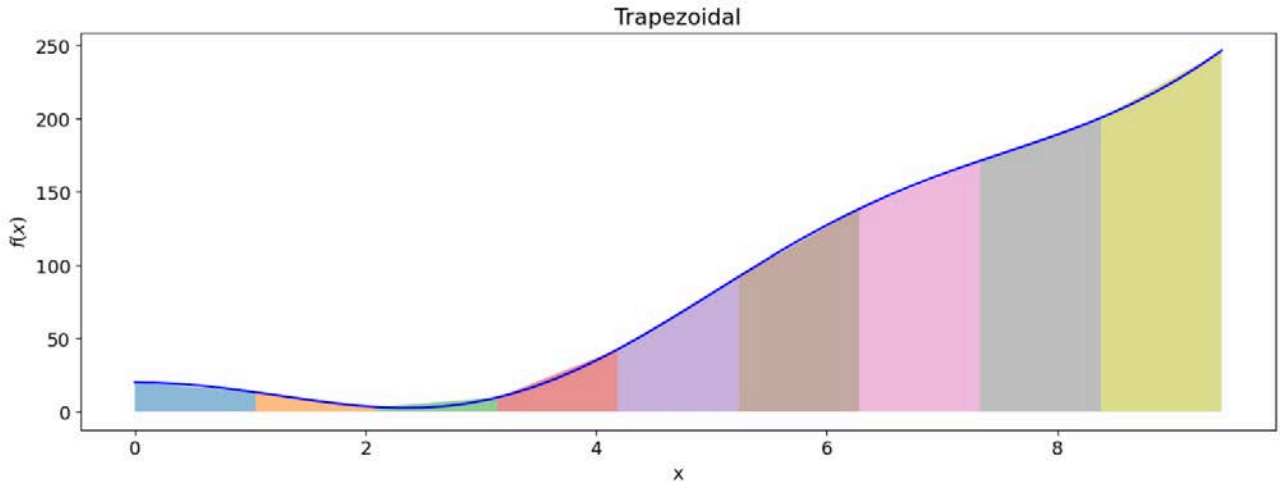



*Fig. 4.9* Illustrating the Trapezoidal Rule over interval [a,b]

The trapezoidal rule uses a polynomial of the first order. As a result the area to be computed is a trapezoid (the top of the rectangle has a slope now) where the bases are the values assumed by the function in the end points of each of the partial intervals, and its height is

just the difference between two consecutive points. Mathematically it reduces to the average of two rectangles:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2} \Delta x$$

# Simpson's Rule

> **Attribution**
>
> This figure is from Strang and Herman (2016). Find out more here.

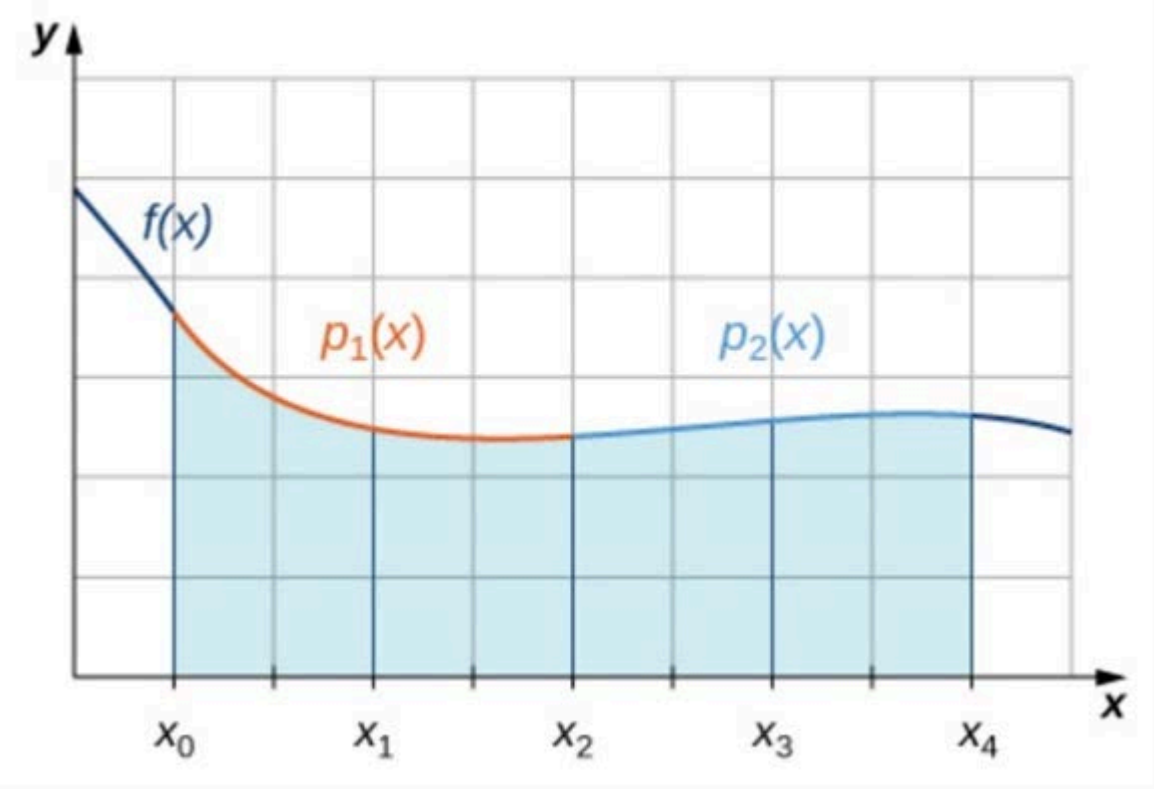



*Fig. 4.10* Illustrating the Simpson Rule over interval [a,b]

Simpson's rule uses a second order polynomial. It computes the integral of the function $f(x)$ by fitting parabolas needing three consecutive function points and adding the areas under them (see $p_1(x)$ and $p_2(x)$ in the figure). In formula form:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n/2} \frac{f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})}{6} 2\Delta x$$

Note that the interval length is multiplied with two, since three points are needed.

***Exercise***

Estimate the integral $\int_{0.5}^{1} x^4$ with a total number of steps n=2:

- Left Riemann Sum
- Midpoint rule
- Trapezoidal

and then compute the real error $\epsilon = |f(x) - f(\hat{x})|$, where f(x) is the exact integral and $f(\hat{x})$ the approximated integral

> **Tip**
>
> Tip: feel free to use the code cell below as a calculator.

**Solution**

- find $\Delta x$,

$$\frac{b-a}{n} = \frac{1-0.5}{2} = 0.25$$

**Left Riemann Sum**:

$$\sum_{i=0}^{n-1} f(x_i)\Delta x = f(x_0)\Delta x + f(x_1)\Delta x$$

$$0.5^4 \cdot 0.25 + 0.75^4 \cdot 0.25 = 0.0947265625$$

**Midpoint Rule**:

$$\sum_{i=0}^{n-1} f(\frac{x_i + x_{i+1}}{2})\Delta x = f(\frac{x_0 + x_1}{2})\Delta x + f(\frac{x_1 + x_2}{2})\Delta x$$

$$0.625^4 0.25 + 0.875^4 0.25 = 0.1846923828125$$

**Trapezoidal Rule**:

$$\sum_{i=0}^{n-1} \frac{f(x_i) + f(x_{i+1})}{2}\Delta x = \frac{f(x_0) + f(x_1)}{2}\Delta x + \frac{f(x_1) + f(x_2)}{2}\Delta x$$

$$\frac{f(0.5) + f(0.75)}{2}\Delta x + \frac{f(0.75) + f(1)}{2}\Delta x = \frac{(0.5^4 + 0.75^4)}{2} \cdot 0.25 + \frac{(0.}{}$$

**Exact**

$$\int_{0.5}^{1} x^4\$ = (\frac{1}{5}x^5)\Big|_{0.5}^{1} = \frac{1}{5}1^5 - \frac{1}{5}0.5^5 = 0.19375$$

**Error**

$$\epsilon = |f(x) - f(\hat{x})|$$

error Left Riemann = $|0.19375 - 0.0947265625|$ = 0.0990234375

error Midpoint Rule = $|0.19375 - 0.1846923828125|$ = $0.009057617187500006$

error Trapezoidal Rule = $|0.19375 - 0.2119140625|$ = $0.018164062499999994$

```
#feel free to use this as your calculator for the exercise above

def f(x):
    return x**4

f_left_riemann=
f_Midpoint=
f_trapezoidal=
f_exact=

print(f"f_left_riemann = {f_left_riemann}")
print(f"f_Midpoint = {f_Midpoint}")
print(f"f_trapezoidal = {f_trapezoidal}")
print(f"f_exact = {f_exact}")
print(f"Error left Riemann = {abs(f_exact-f_left_riemann)}")
print(f"Error Midpoint = {abs(f_exact-f_Midpoint)}")
print(f"Error trapezoidal = {abs(f_exact-f_trapezoidal)}")
```

# Truncation Errors

The truncation error for each method is the difference between the exact integral and the approximated integral. In the case of the left Riemann method:

$$\text{Left Riemann Error } = \int_a^b f(x)dx - \sum_{i=0}^{n-1} f(x_i)\Delta x$$

Now, we can use the Taylor Series Expansion of $f(x)$ instead of $f(x)$ itself and solve that integral. If you are curious about the mathematical steps, further below you can find the detailed derivation. For the moment, let's see the absolute errors for the different techniques.

$$\text{Left Riemann Error } = \left|\bar{f}'(b-a)\Delta x/2\right| \text{ therefore } \mathcal{O}(\Delta x)$$

$$\text{Right Riemann Error } = \left|\bar{f}'(b-a)\Delta x/2\right| \text{ therefore } \mathcal{O}(\Delta x)$$

$$\text{Midpoint Error } = \left|\bar{f}''(b-a)\Delta x^2/2\right| \text{ therefore } \mathcal{O}(\Delta x^2)$$

$$\text{Trapezoidal Error } = \left|\bar{f}''(b-a)\Delta x^2/2\right| \text{ therefore } \mathcal{O}(\Delta x^2)$$

$$\text{Simpsons Error } = \left|\bar{f}''''(b-a)\Delta x^4/2\right| \text{ therefore } \mathcal{O}(\Delta x^4)$$

## Analysis

As we can see, for the left and right Riemann integration techniques we find errors $\sim \frac{1}{n}$, while for the midpoint and trapezoidal rules, we find them $\sim \frac{1}{n^2}$, and with Simpson's rule $\sim \frac{1}{n^4}$.

## Truncation Error Derivation for Left Riemann integration

> **Note**
>
> You don't need to know the derivation for the exam.

**Derivation**

$$\text{Left Riemann Error } = \left|\int_a^b f(x)dx - \sum_{i=0}^{n-1} f(x_i)\Delta x\right| \quad (1)$$

**Derivation**

Start by using a Taylor series expansion:

$$f(x) = f(x_i) + (x - x_i)f'(\xi(x)) + \mathcal{O}(\Delta x)^2 \quad (2)$$

Substitute this Taylor expansion (2) in the term $\int_a^b f(x)dx$ in equation (1):

$$\int_a^b \left[f(x_i) + (x - x_i)f'(x_i) + \mathcal{O}(\Delta x)^2\right]dx \quad (3)$$

Rewrite this as the sum of the integrals over the sub-intervals $[x_i, x_{i+1}]$:

$$= \sum_{i=0}^{n-1} \int_{x_i}^{x_{x+1}} \left[f(x_i) + (x - x_i)f'(x_i) + \frac{1}{2}(x - x_i)^2 f''(x) + \mathcal{O}(\Delta x)^2\right]dx$$

Using integration by substitution, $u = x - x_i$, therefore $\frac{du}{dx} = 1$ meaning du=dx

$$= \sum_{i=0}^{n-1} \int_{x_i}^{x_{x+1}} \left[f(x_i) + (u)f'(x_i) + \mathcal{O}(\Delta x)^2\right]du \quad (5)$$

$$= \sum_{i=0}^{n-1} \left[(u)f(x_i) + \frac{(u)^2}{2}f'(x_i) + \mathcal{O}(\Delta x)^2\right]\Bigg|_{x_i}^{x_{i+1}} \quad (6)$$

$$= \sum_{i=0}^{n-1} \left[(x_{i+1} - x_i)f(x_i) + \frac{(x_{i+1} - x_i)^2}{2}f'(x_i)) + \mathcal{O}(\Delta x)^2\right] \quad (7)$$

$$= \sum_{i=0}^{n-1} \left[\Delta x f(x_i) + \frac{\Delta x^2}{2}f'(x_i)) + \mathcal{O}\Delta x^2\right] \quad (8)$$

Plug equation (8) in equation (1):

$$|\sum_{i=0}^{n-1} \left[(\Delta x)f(x_i) + \frac{\Delta x^2}{2}f'(x_i)) + \mathcal{O}(\Delta x)^2\right] - \sum_{i=0}^{n-1} f(x_i)\Delta x| \quad (9)$$

$$= |\sum_{i=0}^{n-1} \left[\Delta x f(x_i) + \frac{\Delta x^2}{2} f'(x_i)) + \mathcal{O}(\Delta x)^2 - f(x_i)\Delta x\right]| \quad (10)$$

The term $\Delta x f(x_i)$ falls away:

$$= |\sum_{i=0}^{n-1} \left[\frac{\Delta x^2}{2} f'(x_i)) + \mathcal{O}(\Delta x)^2\right]| \quad (11)$$

$$= n \cdot (b-a)\frac{\Delta x^2}{2} f'(x_i) \quad (12)$$

$\Delta x = \frac{b-a}{n}$ this gives:

$$\text{Left Riemann Error } = |\int_a^b f(x)dx - \sum_{i=0}^{n-1} f(x_i)\Delta x| = |\bar{f}'(b-a)\frac{\Delta x}{2}|$$

The resulting left Riemann error is proportional to the average of the derivatives, the length of the domain (b-a) and the step size ($\Delta x$). This result tells us that for steeper functions the error will be higher than for milder ones. For example, a function of a straight horizontal line with average derivatives equal to 0 would have an error equal to 0! Also, the error will be larger if the domain is larger, which is kind of obvious. As those first two terms (average derivative and domain length) are inherent to the problem requirements, we say that the error is of the order $\Delta x$.

**Exercise**

Estimate the integral $\int_1^3 \frac{5}{x^4} dx$ with a step-size $\Delta x = 0.5$

- Right Riemann Sum
- Simpson Rule

**Tip**

Tip: feel free to use the code cell below as a calculator.

**Solution**

Find $n$

$$n = \frac{b-a}{\Delta x} = \frac{3-1}{0.5} = 4$$

***Right Riemann Sum***

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} f(x_i)\Delta x \approx \sum_{i=0}^{n-1} f(x_{i+1})\Delta x$$

$$= \frac{5}{1.5^4}\cdot 0.5 + \frac{5}{2^4}\cdot 0.5 + \frac{5}{2.5^4}\cdot 0.5 + \frac{5}{3^4}\cdot 0.5$$

$$= 0.7449413580246914$$

***Simpsons Rule***

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n/2} \frac{f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})}{6} 2\Delta x$$

$$\frac{\frac{5}{1^4} + 4\frac{5}{1.5^4} + \frac{5}{2^4}}{6} 2\cdot 0.5 + \frac{\frac{5}{2^4} + 4\frac{5}{2.5^4} + \frac{5}{3^4}}{6} 2\cdot 0.5 = 1.69155761316872$$

***Exact***

$$\int_1^3 \frac{5}{x^4}\,dx = -\frac{5}{3x^3}\Big|_1^3 = 1.60493827160$$

***error***

$$\text{Right Riemann error} = |1.60493827160 - 0.744941358| = 0.8599969$$

$$\text{Simpsons Rule error} = |1.60493827160 - 0.744941358| = 0.0866193$$

```
#feel free to use this as your calculator for the exercise above
def f(x):
    return 5/(x**4)

f_right_riemann =
f_simps =
f_exact =

print(f"f_right_riemann = {f_right_riemann}")
print(f"f_simps = {f_simps}")
print(f"f_exact = {f_exact}")
print(f"Error right Riemann = {abs(f_exact-f_right_riemann)}")
print(f"Error Simpsons = {abs(f_exact-f_simps)}")
```

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Initial Value Problem for ODE: single-step methods

## Contents


- Forward (Explicit) Euler
- Error analysis
- Stability
- Backward (Implicit) Euler
- Nonlinear ODE


The solution process of an ODE is performed by steps. In a single-step approach, the solution of the following step depends on the current one. In a multiple step method, the solution of the next step is calculated from several steps. A multiple step approach is more accurate, you can think of a similar construct as the higher-order derivatives treated previously.

For first order ODEs with the general form:

$$\frac{dy}{dx} = f(x, y)$$

**One initial condition is needed to find only one solution!** Without it, there would be an infinite number of solutions possible. The initial condition is:

$$y(x_0) = y_0$$

# Forward (Explicit) Euler

The simplest initial value numerical integration is the Forward Euler, which is an explicit method. Although simple, it contains the basic characteristics as more advanced and accurate methods. It looks as:

$$x_{i+1} = x_i + \Delta x$$

$$y_{i+1} = y_i + \Delta x \cdot slope|_i$$

where $\Delta x$ is the step size and the slope is a constant that approximates the rate of change of $y$ with respect to $x$ (a.k.a. the derivative) in the interval $x_i$ to $x_{i+1}$. The solution starts at $i = 0$ given by the initial condition, then $i$ is increased to 1 where the values are calculated using the previous equations. This loop continues until the points cover the desired domain. **The computation of the slope is the key difference between single step methods. For forward euler, the slope is computed with the values at the current step!**

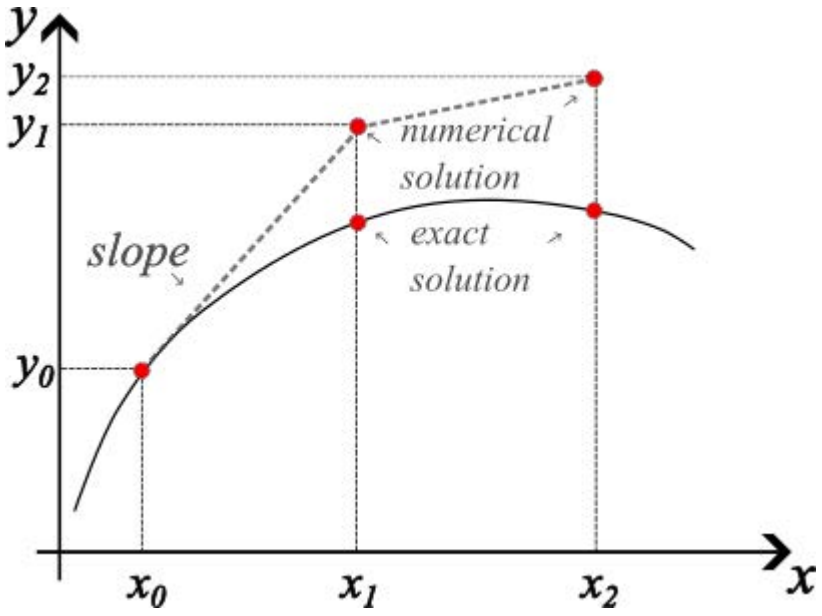



*Fig. 4.11* Illustrating the forward Euler method

# Forward Euler Example

Consider the following equation:

$$\frac{dy}{dx} = y' = -y$$

We want to know the solution in the domain $[a = 0, b = 30]$. The **initial condition** is $y(x_0) = 1$ and the step size is $\Delta x = 0.2$.

The discretization of the differential equation transforms the problem into an algebraic one. Following the formula of explicit Euler for the discretization yields:

$$y_{i+1} = y_i + \Delta x * y_i'$$

In the following code, Forward Euler is implemented.

```
import numpy as np
import matplotlib.pyplot as plt

dx = .2
x = np.arange(0,30+dx,dx)
y = np.zeros(len(x))

##-----------------------------
##Forward Euler Implementation
##-----------------------------
alpha= 1
y[0] = 1
for i in range(len(x)-1):
    y[i+1] = y[i] + dx*( -alpha*y[i])
##-----------------------------
```

run | run all | add cell | clear

**Exercise**

The **Forward Euler** is a Finite Difference Method where it uses a grid-based approach and computes the solution at the nodes.

Can you show how the Forward Euler gets derived from the Finite Difference method

hint: Use the forward difference that we derived earlier in the Taylor Series Expansion chapter .

**Solution**

In the Taylor expansion chapter we derived the forward difference for a first derivative:

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{\Delta x} + \mathcal{O}(\Delta x)$$

$$\Delta x f'(x_i) = f(x_{i+1}) - f(x_i) + \mathcal{O}(\Delta x)^2$$

$$f(x_{i+1}) = f(x_i) + \Delta x f'(x_i) + \mathcal{O}(\Delta x)^2$$

**Exercise**

Use the forward Euler to approximate the solution to the initial-value problem.

$$\frac{dy}{dt} = y^2, \quad , 0 \leq t \leq 1$$

$$y(t_0) = 1$$

with $\Delta t = 0.5$

**Solution**

$t_0 = 0, t_1 = 0.5, t_2 = 1$

Use the forward euler:

$$y_{i+1} = y_i + \Delta x \cdot y_i'$$

for i =0

$$y_1 = y_0 + \Delta x \cdot y_0' = 1 + 0.5 \cdot 1 = 1.5$$

for i = 1

$$y_2 = y_1 + \Delta x \cdot y_1' = 1.5 + 0.5 \cdot 1.5^2 = 2.625$$

The approximated solution is 2.625

# Error analysis

There are two types of errors: round-off and truncation. The **round-off** error is due to the computer's limitation to represent a floating number (decimal). To illustrate it consider the following: the difference between 1 and 0.9 is 0.1. If you subtract 0.1 from itself, you should obtain 0, even if this subtraction is repeated ten thousand times. The following code demonstrates this behavior, using floating-point numbers with varying precision.

```python
def accumulated_error(a,b,solution,iterations):
    error = solution - np.abs(a-b)
    error_accumulated = 0
    for i in range(iterations):
        error_accumulated = error_accumulated + error

    return error_accumulated

print('Error using 16 bits of memory = ',accumulated_error(np.float16(
print('Error using 32 bits of memory = ',accumulated_error(np.float32(
print('Error using 64 bits of memory = ',accumulated_error(np.float64(
```

run | run all | add cell | clear

As you can see, this simple operation can give discernible errors. Imagine a computation spanning 100 years with a time step of seconds and more complex operations: the round-off error will be present! As you can see, this can be reduced by increasing the precision or the number of digits used to represent numbers but be careful, this is not free as the memory the computer uses increases as well as the computation time.

The truncation error is related to the method chosen for the slopes approximation. This can be obtained using our reliable TSE as this gives the exact solution. Therefore **the truncation error per step** is

$$\text{Truncation error } = y^{TSE}(x_{i+1}) - y^{\text{Forward Euler}} \qquad (x_{i+1})$$

$$\text{Truncation error } = y(x_i) + \Delta x y_i' + \frac{(\Delta x)^2}{2!} y_i'' + \ldots - y(x_i) - \Delta x y_i' \approx \mathcal{O}(\Delta x^2)$$

The **total truncation error** accounts for the number of steps as:

$$\text{Total truncation error } \approx \sum_{i=0}^{n-1} \frac{(\Delta x)^2}{2!} y_i'' \approx \frac{(\Delta x)^2}{2!} \frac{b-a}{\Delta x} \bar{y}'' \approx \mathcal{O}(\Delta x)$$

In honor to this first order total truncation error, Forward Euler is referred to as a first-order method.

# Stability

The error that is introduced in each step of the numerical solution ideally does not to increase as the solution advances. Under a well posed and proper solution, the error is expected to reduce with smaller steps. In some cases, the error increases without bound as the solution advances (even with smaller steps): **the solution becomes unstable**. The stability depends on the numerical method, the step size and the behavior of the differential equation. Therefore, the stability conditions will differ when applying the same numerical method to different equations.

Let's consider the **stability for Forward Euler** applied to a more general form of the problem above.

$$\frac{dy}{dx} = y' = -\alpha y$$

With initial condition $y(0) = 1$ and $\alpha > 0$, the exact solution is:

$$y^{\text{exact}}\ (x_i) = y_0 e^{-\alpha x_i}$$

The Forward Euler equivalent is

$$y_{i+1} = y_i - \alpha y_i \Delta x = y_i(1 - \alpha \Delta x)$$

Following the initial steps of the numerical solution a pattern arises:

$$y_1 = y_0(1 - \alpha \Delta x)$$

$$y_2 = y_1(1 - \alpha \Delta x) = y_0(1 - \alpha \Delta x)^2$$

$$y_3 = y_2(1 - \alpha \Delta x) = y_0(1 - \alpha \Delta x)^3$$

$$y_n = y_0(1 - \alpha \Delta x)^n$$

Comparing this last equation with the exact solution, it can be seen that the term $(1 - \alpha \Delta x)^n$ in the numerical solution is approximating the term $e^{-\alpha \Delta x_i}$ in the exact solution. The latter tends to decay with larger values of $x_i$ and positive $\alpha$. To make sure that the term $(1 - \alpha \Delta x)^n$ decays with larger $n$ values $1 - \alpha \Delta x$ should be less than $|1|$, i.e.,

$$|1 - \alpha\Delta x| < 1 \implies 0 < \alpha\Delta x < 2$$

**Click here for a more detailed derivation**

$$-1 < 1 - \alpha\Delta x < 1$$

Subtract 1 of both sides

$$-2 < -\alpha\Delta x < 0$$

Divide by -1

$$0 < \alpha\Delta x < 2$$

This is the stability criterion. If not complied, then the solution will be unstable. It is said that **Forward Euler is conditionally stable! This is true for every explicit numerical method.**

**Exercise**

Lets go back to the last example described in the code above.

$$\frac{dy}{dx} = y' = -y$$

We want to know the solution in the domain $[a = 0, b = 30]$. The **initial condition** is $y(x_0) = 1$ and the step size is $\Delta x = 0.2$.

What is the step size at which the problem described using the Forward Euler becomes unstable? Feel free to use the interactive figure below and change dx to test what happens when the step size becomes larger.

Click `rocket` –>`Live Code` to interact with the plot below

**Solution**

You can see in the interactive figure above that when the stepsize becomes larger the forward euler solution becomes less accurate. You can also see that when the step size becomes bigger then 2 the Forward Euler solution starts to blow up. This is because of the stability criteria of the Forward Euler.

$$0 < \alpha \Delta x < 2$$

When $\alpha = 1$, then $\Delta x < 2$ (Note that $\Delta x$ can not be negative and therefore will always be > 0 )

# Backward (Implicit) Euler

Just as Forward Euler is the simplest Explicit method, **Backward Euler method is the simplest implicit method**. They look very similar but the slope is computed at the next step! That's where the implicit term comes from: the slope depends on the unknown value.

$$x_{i+1} = x_i + \Delta x$$

$$y_{i+1} = y_i + \Delta x \cdot slope|_{i+1} = y_i + \Delta x \cdot y'_{i+1}$$

Let's consider the same problem as before:

$$\frac{dy}{dx} = y' = -\alpha y$$

We want to know the solution in the domain $[a = 0, b = 30]$. The **initial condition** is $y(x_0) = 1$ and the step size is $\Delta x = 0.2$.

The discretization of the differential equation transforms the problem into an algebraic one. Following the formula of implicit Euler, the discretization yields:

$$y_{i+1} = y_i - \Delta x \cdot \alpha y_{i+1}$$

then

$$y_{i+1} + \Delta x \cdot \alpha y_{i+1} = y_i$$

finally

$$y_{i+1} = y_i/(1 + \Delta x \cdot \alpha)$$

In the following code, Implicit Euler is implemented.

```
dx = 0.3
x = np.arange(0,30+dx,dx)
y = np.zeros(len(x))

##-----------------------------
##Backward Euler Implementation
##-----------------------------
alpha=1
y[0] = 1
for i in range(len(x)-1):
    y[i+1] = y[i]/(1+dx*alpha)
##-----------------------------

y_exact = np.exp(-alpha*x)
plt.plot(x,y_exact)
```

run | run all | add cell | clear

You can see that the result is similar to Forward Euler, except that the curve for Backward Euler is slightly above than the exact solution. This makes sense as the derivative is taken to be in front, so the slope is initially underestimated while the explicit solution initially overestimated the slope.

The round-off errors and truncation error are similar. Backward Euler is also a first-order method. But what about the stability?

**Exercise:**

Modify the step size in the Implicit Euler code, try to make the solution unstable. What do you notice?

Let's understand what happens by following the same procedure as for the stability analysis of Explicit Euler.

The exact solution is the same:

$$y^{\text{exact}}\ (x_i) = y_0 e^{-\alpha x_i}$$

For the numerical solution, the numerical solution pattern is slightly different:

$$y_1 = y_0 \left( \frac{1}{1+\alpha \Delta x} \right)$$

$$y_2 = y_1 \left( \frac{1}{1+\alpha \Delta x} \right) = y_0 \left( \frac{1}{1+\alpha \Delta x} \right)^2$$

$$y_3 = y_2 \left( \frac{1}{1+\alpha \Delta x} \right) = y_0 \left( \frac{1}{1+\alpha \Delta x} \right)^3$$

$$y_n = y_0 \left( \frac{1}{1+\alpha \Delta x} \right)^n$$

This last term to the power $n$ approximates the decaying exponential in the exact solution, just like in the Explicit Euler method. Here, the only condition that must be satisfied for stability is

$$\left| \frac{1}{1+\alpha \Delta x} \right| < 1$$

This can be expressed as:

$$-1 < \frac{1}{1+\alpha \Delta x} < 1$$

Lets first look at the right side condition $\frac{1}{1+\alpha \Delta x} < 1$.

If we multiply by the term $1 + \alpha \Delta x$, we get:

$$1 < 1 + \alpha \Delta x$$

As $\alpha$ is positive and the step size must be larger than 0, the right side of the criterion is always true.

Lets now look at the left side condition $-1 < \frac{1}{1+\alpha\Delta x}$:

We again multiply by $1 + \alpha\Delta x$:

$$-1 - \alpha\Delta x < 1$$

Again this left side criterion always holds.

Hence, **an implicit Backward Euler scheme is unconditionally stable.** This is also the case for more advanced implicit schemes.

# Global errors

The global error is the sum of round-off and total truncation errors. The plot below displays the global errors for both the Forward and Backward Euler methods, alongside the reference line representing the slope of the truncation error $\mathcal{O}(\Delta x)$ (1 to 1). As shown, the global errors for both methods align with the expected slope, indicating that the errors decrease in accordance with the methods' first-order accuracy.

$$\text{Global Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - y_{\text{exact}\ ,i}\right)^2}$$

```
import numpy as np
import matplotlib.pyplot as plt

def global_error(dx_values):

    errors_forward = []
    errors_backward = []

    for dx in dx_values:

        x = np.arange(0, 30 + dx, dx)
        y_forward = np.zeros(len(x))
        y_backward = np.zeros(len(x))

```

run run all add cell clear

# Nonlinear ODE

In the previous example, the explicit and implicit implementation seem to only differ in the formulation. However, this changes when treating non-linear equations. Let's see an example to identify the "implicitness".

A certain pollutant $p$ decays over time at a rate proportional to its concentration to the power $3/2$. There is a second mechanism that produces $p$ in time and scales with $p_{cst} = 1000$, $cst$ indicates a constant:

$$\frac{dp(t)}{dt} = -p^{3/2} + 5 \cdot p_{cst}(1 - e^{-t})$$

where $p(t)$ is the pollutant concentration in time. The initial concentration is also $p(t = 0) = p_0 = 1000$. Solve the equation from $t = 0$ until $t = 0.5$ with a time step $\Delta t = 0.002$ s. Use first the explicit Forward Euler and then the implicit Backward Euler.

The expression for the explicit Forward Euler becomes:

$$p_{i+1} = p_i + \Delta t \left( -p_i^{3/2} + 5p_{cst} \cdot (1 - e^{-t_i}) \right)$$

The algorithm to solve the explicit Forward Euler is exactly the same as before.

```
import numpy as np
import matplotlib.pyplot as plt

dt = 0.0002
t = np.arange(0,1+dt,dt)
p = np.zeros(len(t))

##-----------------------------
##Explicit Euler Implementation
##-----------------------------
p_cst = 1000
p[0] = p_cst
for i in range(len(t)-1):
    p[i+1] = p[i] + dt*( -p[i]**(3/2) + 5*p_cst*(1-np.exp(-t[i])))
##-----------------------------
```

run | run all | add cell | clear

Now, using the implicit Backward Euler, the expression becomes:

$$p_{i+1} = p_i + \Delta t \left( -p_{i+1}^{3/2} + 5p_{cst} * (1 - e^{-t_{i+1}}) \right)$$

Obtaining the solution requires a different procedure because now the unknown $p_{i+1}$ appears in both sides of the equation and cannot be solved directly instead an iterative procedure is needed! For example, a widely use method like Newton Rhapson.

# Newton-Raphson Method

In the Observation Theory week, you used Gauss-Newton to solve a non-linear least square problem. That method is actually an extension of the Newton-Rhapson method. Here, it is described with an emphasis in the tricky aspects. Also, the algorithm is summarized and its main advantages and disadvantages are mentioned.

Think of a function $g(z) = 0$. Its solution is graphically represented as the intersection of $g(z)$ with the $z$ axis (see the Figure below). Numerically that solution can be found by selecting an initial guess $z_0$, estimating the tangent (slope) $g'(z_0)$ and finding its intersection with the $z$ axis, which indicates the value $z_1$. Now, the solution is closer. We repeat this process to find $z_2$ but now the guess is $z_1$. The process is repeated until the value $g(z_*)$ is very small: $|g(z_*) < \epsilon|$. Here, $\epsilon$ is the tolerance e.g., $\epsilon = 10^{-6}$ (see the red square in the Figure below).

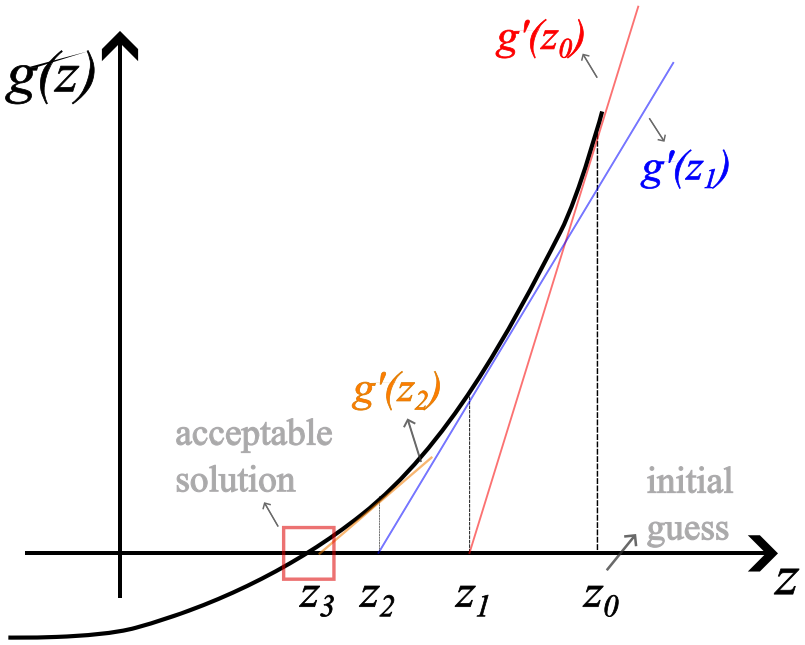



*Fig. 4.12* Illustrating the Newton-rhapson method

# Some remarks:

Finding the point $z_{j+1}$ where the tangent crosses the $z$-axis is done following an approximation of first order of $g'(z_j)$:

$$g'(z_j) = \frac{g(z_j) - 0}{z_j - z_{j+1}}$$

solving for $z_{j+1}$ gives the expression

$$z_{j+1} = z_j - \frac{g(z_j)}{g'(z_j)}.$$

Look at the implicit Backward Euler approximation of the pollutant problem. We can rewrite it to be $g(z) = q(p_{i+1}) = 0$ as follows:

$$p_{i+1} - p_i - \Delta t \left( -p_{i+1}^{3/2} + 5p_{cst} \cdot (1 - e^{-t_{i+1}}) \right) = 0$$

and $g'(z) = q'(p_{i+1})$ is its derivative:

$$g'(z) = q'(p_{i+1}) = 1 + \frac{3}{2} \Delta t p_{i+1}^{1/2}$$

Finally, the expression used to iterate is

$$p_{i+1}^{j+1} = p_{i+1}^{j} - \frac{p_{i+1}^{j} - p_i - \Delta t \left( (-p_{i+1}^{j})^{3/2} + 5p_{cst} \cdot (1 - e^{-t_{i+1}}) \right)}{1 + \frac{3}{2} \Delta t (p_{i+1}^{j})^{1/2}}$$

Notice that the subscript $i$ relates to the time steps of the backward Euler method and the superscript $j$ relates to the Newton-Raphson iteration, where $p_{i+1}^{j}$ when $j = 0$ is the initial guess ($p_{i+1}^{0}$)! The solution is found and the iteration stops at $j = m$, once $|g(z) = q(p_{i+1}, m)| < \epsilon$.

**The Newton-Raphson method:**

- Converges quickly and can provide highly accurate approximations.
- It can be used for both real and complex roots.

**But the method also has its limitations:**

- It may not converge if the initial guess is far from the actual value or if the function is discontinuous near the value.
- The derivative of the function is needed, which may not always be known.

# Pseudo code: The Backward-Euler-Newton-Raphson Algorithm

$$p' = -p^{3/2} + 5 \cdot p_{\text{cst}}\ (1 - e^{-t}), \quad p(t_0) = p_0$$

## Backward Euler Scheme:

The update equation for $p_{i+1}$ is given by:

$$p_{i+1} = p_i + \Delta t \left( -p_{i+1}^{3/2} + 5 \cdot p_{\text{cst}}\ \cdot (1 - e^{-t_{i+1}}) \right)$$

We solve for $p_{i+1}$ at $t_i$, for $i = 0, 1, \ldots, m$ using Newton-Raphson.

## Newton-Raphson Method:

$$g(z) = p_{i+1}^{j} - p_i - \Delta t \left( -(p_{i+1}^{j})^{3/2} + 5 \cdot p_{\text{cst}}\ \cdot (1 - e^{-t_{i+1}}) \right)$$

The iteration for $p_{i+1}$ is as follows:

- Initialize: $p_{i+1}^{(0)} = p_i$
- Set $j = 0$
- **while** $\left| g(p_{i+1}^{j}) \right| > \epsilon$ **do**:
  - Update $p_{i+1}^{j+1}$ using the Newton-Raphson formula:

$$p_{i+1}^{j+1} = p_{i+1}^{j} - \frac{p_{i+1}^{j} - p_i - \Delta t \left( -(p_{i+1}^{j})^{3/2} + 5 \cdot p_{\text{cst}}\ \cdot (1 - e^{-t_{i+1}}) \right)}{1 + \frac{3}{2} \Delta t \cdot (p_{i+1}^{j})^{1/2}}$$

  - Increment $j = j + 1$
- Set $p_{i+1} = p_{i+1}^{j}$

Finally, update $p_{i+1}$ using the Backward Euler formula:

$$p_{i+1} = p_i + \Delta t \left( -p_{i+1}^{3/2} + 5 \cdot p_{\text{cst}} \cdot (1 - e^{-t_{i+1}}) \right)$$

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Multi-step and multi-stage methods

## Contents


- Adams-Bashfort
- Advantages of each method
- Runge-Kutta


## Adams-Bashfort

Multi-stage methods approximate the slope in multiple stages but still are dependent **only on one point** managing to increase the accuracy but increasing heavily the computational cost. A multi-step method uses more than one point to approximate the slope. A well-known explicit method is the two-step Adams-Bashforth method:

$$y_{i+1} = y_i + \frac{\Delta t}{2}\left(3y'_i - y'_{i-1}\right)$$

It can be derived using TSE applied at multiple points. As you might guess by now, the more points used, the more accurate the solution will be. A problem arises at the very beginning of the solution. The explicit method above requires to solve $y_{i+1}$ that $y_i$ and $y_{i-1}$ are known. That is fine at step $y_2$ because $y_1$ and $y_0$ would already have been computed but what happens when you compute $y_1$? Then a point will be missing. That is why this multi-step methods are considered to not be self-starting. This method and also what happens when you compute $y_1$ is shown in the figure below.

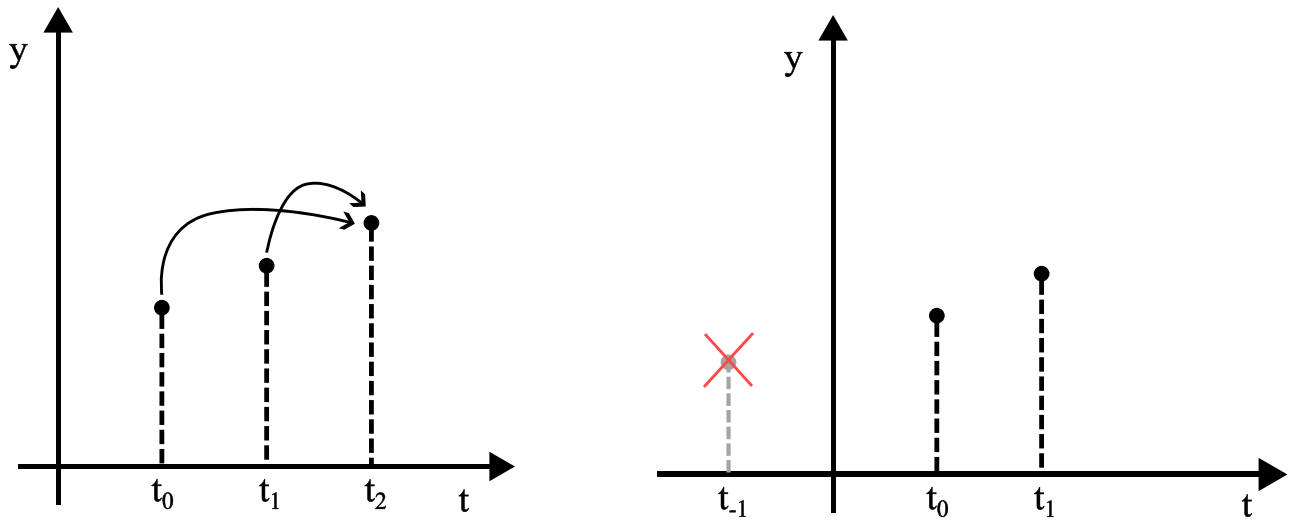



*Fig. 4.13* Illustrating the Adams-Bachfort method

Multi-step methods require an initialization. Either a method that requires less points is applied at the beginning (with the disadvantage that the order of error is not consistent throughout the solution) or another method is applied, one that has the same accuracy as the multi-step method but that depends on less points. This is where multi-stage methods are used. The most well-known of their kind are the Runge-Kutta methods.

There are also multi-step methods that are implicit! The most well-known are called Adams-Moulton.

# Advantages of each method

Multi-step methods are easier to implement, computationally more efficient and stabler, allowing the use of larger time steps. Multi-stage methods require less points but are computationally expensive.

Regardless, each method has its advantages and are generally employed based on a desired accuracy or computational efficiency. The choice, however, can often be complicated and requires thorough knowledge of the underlying mathematics. The RK4 and Adams-Bashfort have to comply with stability criteria, just as expected from any explicit method.

# Runge-Kutta

Here, it is shown how the 4th order explicit Runge-Kutta (RK4) method has four stages and the whole process only uses one single point.

$$y_{i+1} = y_i + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) + \mathcal{O}(\Delta t^5)$$

The coefficients $k$ are calculated using:

$$\begin{cases} k_1 &= y'_i \\ k_2 &= y'_{i+1/2} \text{ with } y_{i+1/2} = y_i + \frac{\Delta t}{2} k_1 \\ k_3 &= y'_{i+1/2} \text{ with } y_{i+1/2} = y_i + \frac{\Delta t}{2} k_2 \\ k_4 &= y'_{i+1} \text{ with } y_{i+1} = y_i + \Delta t\, k_3 \end{cases}$$

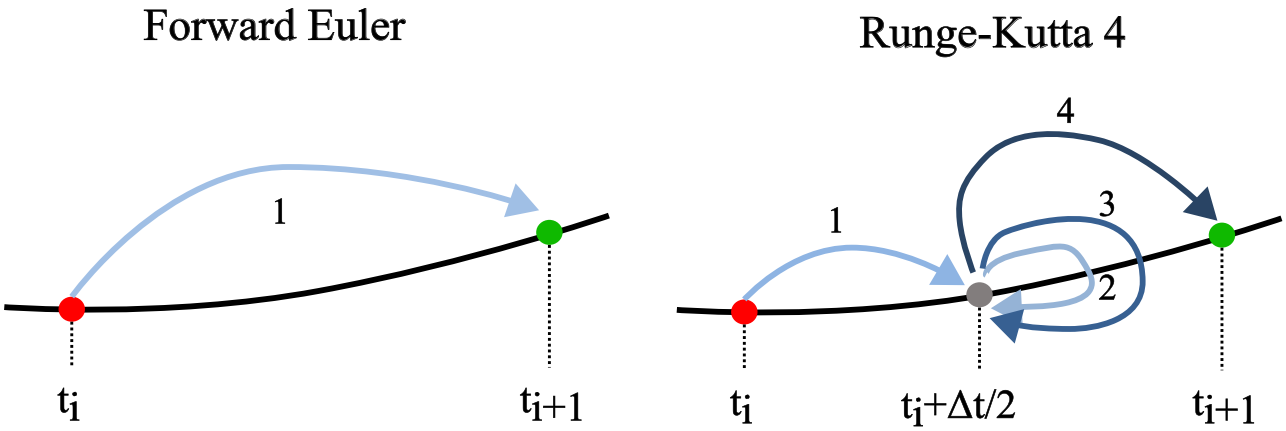



*Fig. 4.14* Illustrating two different methods Forward Euler (on the left) and Runge-Kutta 4 (on the right)

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Boundary-value Problems: second-order ODE

## Contents


- Boundary Conditions
- Solving a BVP using Finite Differences


To solve a first order ODE, one constraint is needed (initial value problem, IVP). In the case of a second order ODE, two constraints are needed. **If the constraints are defined at different locations of the domain, then you will be dealing with a Boundary-value problem (BVP)**. Mostly when we use time derivatives we have an IVP and when we have a derivative in space we have to deal with spacial constraints and therefore BVP's. See the following examples:

$\frac{dp}{dt} = -p^{3/2} + 5 * p_{cst}(1 - e^{-t})$ first order ODE -> IVP with 1 constraints $p[0] = 1000$

$\frac{d^2y}{dt^2} = \frac{dy}{dt} - y + \cos t = 0$ second order ODE -> IVP with 2 constraints $y[0] = 2$ and $\frac{dy}{dt}$

$\frac{d^2T}{dx^2} - \alpha_1(T - T_s) = 0$ second order ODE -> BVP with 2 constraints $T[0] = 10$ and $T[-$

Even for higher ODE orders, the number of constrains required corresponds to the order of the ODE. It is clear that a second order ODE can either be an IVP or a BVP. The former case is common when describing time derivatives and the latter case is common when describing space derivatives. From now on, we focus on BVP.

## Boundary Conditions

A second order ODE BVP of the type:

$$\frac{d^2y}{dx^2} = g(x, y, \frac{dy}{dx})$$

with a domain solution $a <= x <= b$ and boundary conditions (or constraints) defined at $a$ and $b$ typically has two types of boundary conditions:

- Dirichlet boundary conditions

$$y(x = a) = Y_a \text{ and } y(x = b) = Y_b$$

- Neumann boundary conditions

$$\left.\frac{dy}{dx}\right|_{x=a} = D_a \text{ and } \left.\frac{dy}{dx}\right|_{x=b} = D_b$$

Of course, a combination of BC can also exist.

Consider the following equation that describes the deformation $y$ of a beam with length $L$ clamped at $x = 0$ and $x = L$:

$$\frac{d^4y}{dx^4} = \alpha \text{ load}(x)$$

where $\alpha$ represents the beams material characteristics.

As it is and ODE of fourth order, it needs 4 BCs. The nature of the problem states that the **deformation** and the **slope of the deformation** at the ends is 0, thus, two Dirichlet and two Neumann BCs:

$$y(0) = 0; \left.\frac{dy}{dx}\right|_{x=0} = 0; \quad y(L) = 0; \left.\frac{dy}{dx}\right|_{x=L} = 0;$$

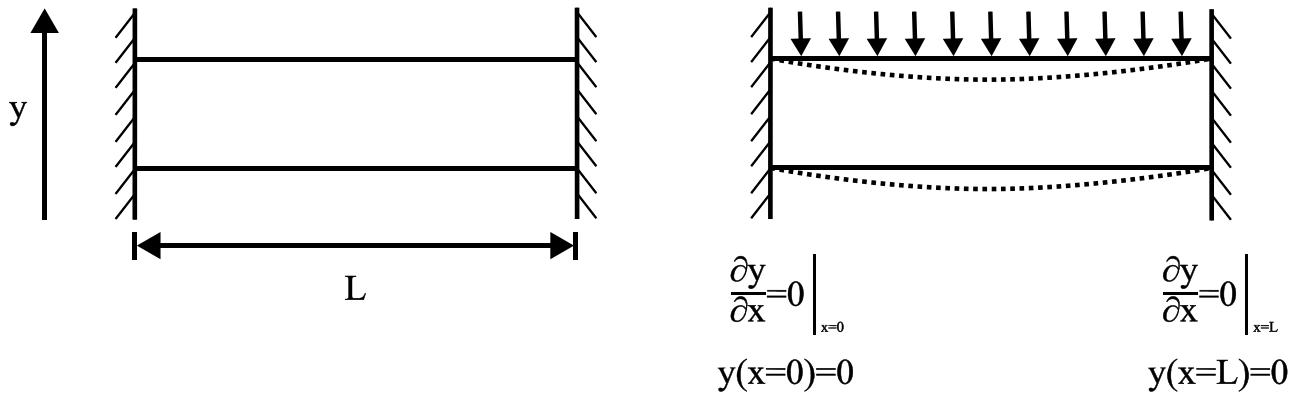


*Fig. 4.15* Illustrating the boundary conditions of a fourth order ODE that describes the deformation of a beam

# Solving a BVP using Finite Differences

Just like in the initial value problem section, here the derivatives are approximated numerically following a desired method and order of accuracy. Now the domain going from $a$ to $b$ is discretized using a determined number of grid points $n$ and, thus, a number of sub-intervals $N$. There is always one more point than sub-intervals, $n = N + 1$. This is the grid (see Figure below) and, if the spacing is regular, then the length of the sub-interval is $\Delta x = (b - a)/N$.

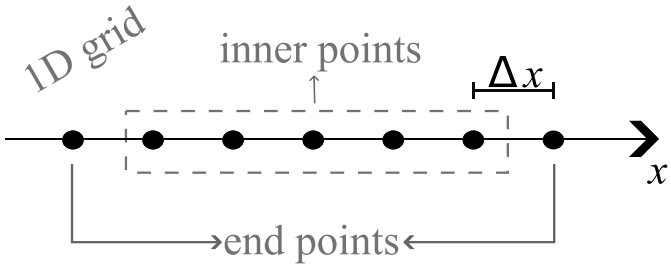



*Fig. 4.16* Illustration of the grid, highlighting the two external boundary nodes with known temperatures ($T$) and the internal nodes where the temperature values must be estimated.

The Boundary Conditions are defined at the end points and the discretization is applied at **almost** every point. This depends on the numerical approximation. The following steps are followed to solve a BVP:

1. Discretization of the differential equation with a numerical approximation of choice
2. Parameter definition
3. Grid creation
4. Define Boundary conditions
5. Building a system of equations according to the discretization: $Ay = b$
6. Solving the system

Lets visualize these steps above with an exercise:

Let’s consider the following problem 2nd order differential equation BVP:

$$\frac{d^2T}{dx^2} - \alpha(T - Ts) = 0, \quad x \in (0, 0.1)$$

$$T(0) = 473[K], \;\; T(0.1) = 293[K]$$

The parameter $\alpha = 166$ and your starting temperature $Ts = 293[K]$

Use the Finite Central Difference method to approximate the above differential equation at 5 equally spaced intervals.

Central Difference: $f'' = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x)^2.$

**Click here for the derivation of the 2nd order central finite difference method**

Solution:

Remember the trick: 2nd order derivative and 2nd error order (2+2=4), so expand the Taylor series until the 4th order

$$f(x_{i+1}) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2} f''x_i + \frac{\Delta x^3}{6} f'''_{x_i} + \mathcal{O}(\Delta x^4)$$

$$f(x_{i-1}) = f(x_i) - \Delta x f'(x_i) + \frac{\Delta x^2}{2} f''x_i - \frac{\Delta x^3}{6} f'''_{x_i} + \mathcal{O}(\Delta x^4)$$

Now, summing the two expansions (for $x_{i+1}$ and $x_{i-1}$):

$$f(x_{i+1}) + f(x_{i-1}) = 2y(x_i) + \frac{\Delta x^2}{2} f''x_i + \mathcal{O}(\Delta x^4)$$

Rearranging to isolate the second derivative and divide by $\Delta x^2$:

$$f'' = \frac{f(x_i - \Delta x) - 2f(x_i) + f(x_i + \Delta x)}{\Delta x^2} + \mathcal{O}(\Delta x).$$

1. **First discretize the differential equation**

We use the Central Difference method to discretize our above differential equation

$$\frac{1}{\Delta x^2}(T_{i-1} - 2T_i + T_{i+1}) - \alpha(T_i - Ts) = 0$$

2. **Define parameters**

Our parameters are $\alpha$ and Ts

3. **Define your grid**

We want 5 equally spaced intervals. $\Delta x = (0.1 - 0)/5 = 0.02$ and $\Delta x^2 = 0.0004$

Define the grid: $x_0 = 0, x_1 = 0.02, x_2 = 0.04, x_3 = 0.06, x_4 = 0.08, x_5 = 0.1$

4. **Define boundary conditions**

Our external nodes $x_0$ and at $x_5$ are our dirichlet boundary conditions, which are our known solutions at:

$$T_0 = 473[K], \quad T_5 = 293[K]$$

5. **Building a System of Equations:** $\mathbf{AT} = \mathbf{y}$

We will use the central difference method to approximate the derivative at the internal nodes.

**For $i = 1$:** The finite difference approximation for the first internal node is:

$$\frac{1}{\Delta x^2}(T_0 - 2T_1 + T_2) - \alpha T_1 + \alpha T_s = 0$$

Next, we multiply by $\Delta x^2$ and substitute the boundary condition for $T_0$. Notice that the only unknown solutions are $T_1$ and $T_2$ Rearranging the terms gives:

$$-(2+\alpha\Delta x^2)T_1 + T_2 = -\alpha T_s \Delta x^2 - T_0$$

**For $i = 2$:** The finite difference approximation for the second internal node is:

$$\frac{1}{\Delta x^2}(T_1 - 2T_2 + T_3) - \alpha T_2 + \alpha T_s = 0$$

Rearranging the terms:

$$T_1 - (2+\alpha\Delta x^2)T_2 + T_3 = -\alpha T_s \Delta x^2$$

**For $i = 3$:** The finite difference approximation for the third internal node is:

$$\frac{1}{\Delta x^2}(T_2 - 2T_3 + T_4) - \alpha T_3 + \alpha T_s = 0$$

Rearranging the terms:

$$T_2 - (2+\alpha\Delta x^2)T_3 + T_4 = -\alpha T_s \Delta x^2$$

**For $i = 4$**

$$\frac{1}{\Delta x^2}(T_3 - 2T_4 + T_5) - \alpha T_4 + \alpha Ts = 0$$

Notice that now we can plug in our other bc $T_5$ and rearrange the terms:

$$(T_3 - (2+\alpha\Delta x^2)T_4) = -\alpha Ts \Delta x^2 - T_5$$

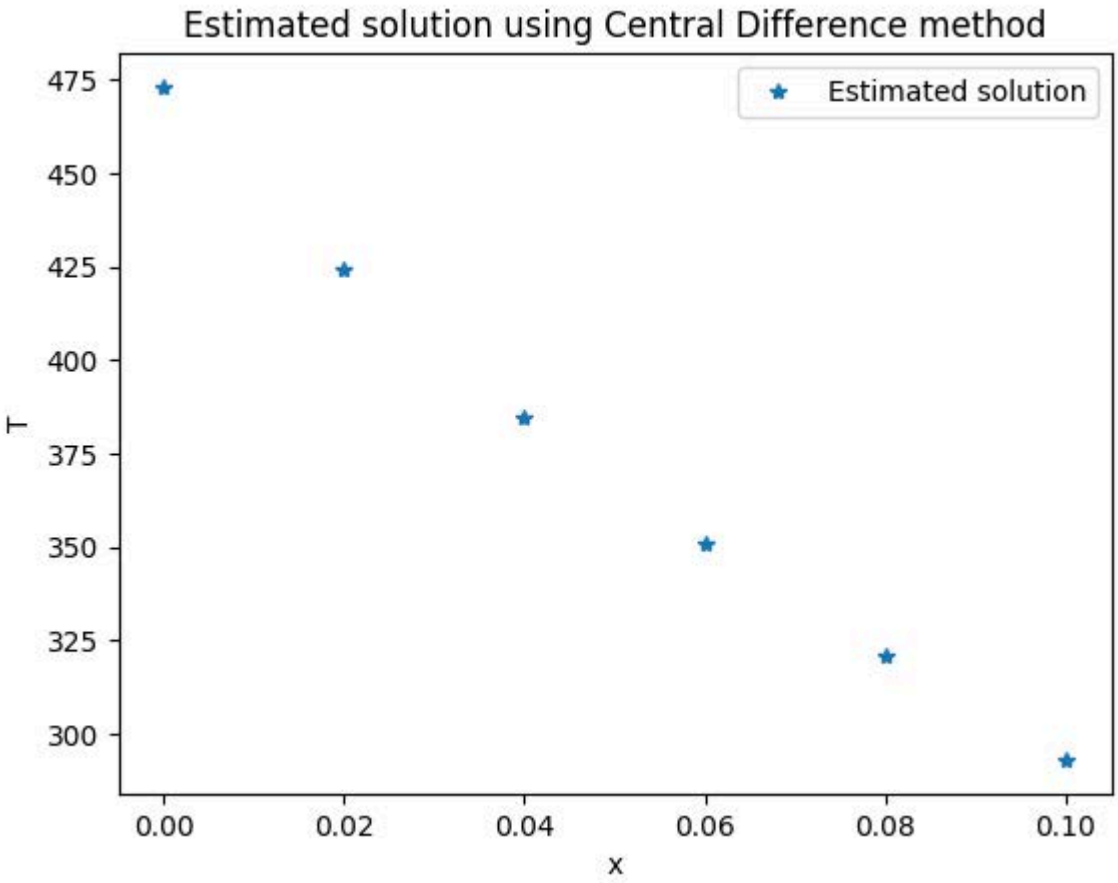
We can now move the equations in the Ax=y form:

$$\begin{bmatrix} -(2+\alpha\Delta x^2) & 1 & 0 & 0 \\ 1 & -(2+\alpha\Delta x^2) & 1 & 0 \\ 0 & 1 & -(2+\alpha\Delta x^2) & 1 \\ 0 & 0 & 1 & -(2+\alpha\Delta x^2) \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} -c \\ \\ \\ -c \end{bmatrix}$$

**Below you can see how we can put the above steps into code:**

We can now estimate the temperature T at each node.

```
import numpy as np
import matplotlib.pyplot as plt


dx=0.02
Ts = 293
alpha = 166
matrix_element = -(2+dx**2*alpha)
b_element = -dx**2*alpha*Ts

# grid creation
x = np.arange(0,0.1+dx,dx)
T = np.zeros(x.shape)

# boundary conditions
T[0] = 473
T[-1] = 293

# Building matrix A
A = np.zeros((len(x)-2,len(x)-2))
np.fill_diagonal(A, matrix_element)
A[np.arange(3), np.arange(1, 4)] = 1  # Upper diagonal
A[np.arange(1, 4), np.arange(3)] = 1  # Lower diagonal
b = np.array([ b_element - T[0] , b_element, b_element,  b_element - T[-1]])

# Solving the system
A_inv = np.linalg.inv(A)
T[1:-1] = A_inv @ b


plt.plot(x,T,'*',label='Estimated solution')
plt.xlabel('x')
plt.ylabel('T')
plt.title('Estimated solution using Central Difference method')
plt.legend()
plt.show()

print(f'The estimated temperature at the nodes are: {[f"{temp:.2f}" for temp in T]} [K]')
```




```
The estimated temperature at the nodes are: ['473.00', '424.46', '384.64', '350.91', '321
```

# Summary of Finite Difference methods

**First order derivative Finite Difference Methods**

$$\text{Backward Difference: } f' = \frac{f(x_i) - f(x_{i-1})}{\Delta x} + \mathcal{O}(\Delta x)$$

$$\text{Forward Difference: } f' = \frac{f(x_{i+1}) - f(x_1)}{\Delta x} + \mathcal{O}(\Delta x)$$

$$\text{Central Difference: } f' = \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x} + \mathcal{O}(\Delta x)^2$$

**Second order derivative Finite Difference Methods**

$$\text{Backward Difference: } f''(x_i) = \frac{f(x_i) - 2f(x_{i-1}) + f(x_{i-2})}{\Delta x^2} + \mathcal{O}(\Delta x).$$

Forward Difference: $f''(x_i) = \dfrac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{\Delta x^2} + \mathcal{O}(\Delta x).$

Central Difference: $f'' = \dfrac{f(x_{i-1}) - 2f(x_i) + f(x_{i+1})}{\Delta x^2} + \mathcal{O}(\Delta x)^2.$

For if you want a bit more practice.

**Exercise**

Here is a second order derivative BCP:

$$f'' = x^2, \quad x \in (0,1)$$

$$f(0) = 1, f(1) = 1$$

Divide the domain [0,1] into 5 equal intervals and apply the Central Finite Difference to estimate $f''$ at each grid node.

**Solution**

Discretize differential equation:

$$f_i'' \approx \frac{1}{\Delta x^2}(f_{i-1} - 2f_i + f_{1+1})$$

Define the grid:

$\Delta x$ = (1-0)/5 = 0.2 Hence $\frac{1}{\Delta x^2} = \frac{1}{0.2^2} = 25$

$$x_0 = 0, x_1 = 0.2, x_2 = 0.4, x_3 = 0.6, x_4 = 0.8, x_5 = 1$$

Define BC:

$$f_0 = 1, \quad f_5 = 1$$

Set up a system of equations for internal nodes i= 1,…,4 :

$$25(f_0 - 2f_1 + f_2) = 0.2^2$$

$$25(f_1 - 2f_2 + f_3) = 0.4^2$$

$$25(f_2 - 2f_3 + f_4) = 0.6^2$$

$$25(f_3 - 2f_4 + f_5) = 0.8^2$$

Plugging in the bc $f_0 = 1$ and $f_5 = 1$ and moving the to the right-hand side:

$$25(-2f_1 + f_2) = 0.2^2 - 25$$

$$25(f_1 - 2f_2 + f_3) = 0.4^2$$

$$25(f_2 - 2f_3 + f_4) = 0.6^2$$

$$25(f_3 - 2f_4) = 0.8^2 - 25$$

we can bring this to a system of linear equations of the form:

$$\frac{1}{\Delta x^2} A\mathbf{f} = \mathbf{y}$$

$$25 \begin{bmatrix} -2 & 1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} 0.2^2 - 25 \\ 0.4^2 \\ 0.6^2 \\ 0.8^2 - 25 \end{bmatrix}$$

### Attribution

This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Treatment of Partial Differential Equations: a start

## Contents


- Method of Lines


Partial Differential Equations (PDEs) are characterized by having more than one independent variable. They can be classified as the ODEs, in linearity or non linearity, first order, second order or higher. See the following widely used PDEs:

$$u = -k_x \frac{\partial \phi}{\partial x};\ v = -k_y \frac{\partial \phi}{\partial y};$$ Darcy's law -> PDE system of first order

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 2xy$$ Poisson's equation -> PDE second order

$$\frac{du}{dt} = \nu \frac{d^2 u}{dx^2}$$ diffusion equation -> PDE second order

$$\frac{\partial \phi}{\partial t} + c \frac{\partial \phi}{\partial x} = 0$$ Convection -> PDE first order

As you can see, there are PDEs that depend only on space variables (e.g. poisson equation) and those that depend on time and space (e.g. diffusion equation). We will now mainly treat the latter using the method of lines. Let's look at a new notation. Consider the variable $u$ at the location $x = i$ and time $t = j$. It can be written as:

$$u(x = i, t = j) = u_i^j$$

# Method of Lines

This method transforms the PDE into a system of ordinary differential equations by replacing the spatial derivatives with a numerical approximation. Let's see this transformation in the diffusion equation using the dependent variable temperature $T$ and the independent variable $x$ for space:

$$\frac{dT}{dt} = \nu \frac{d^2T}{dx^2}$$

Clearly $T(x,t)$ varies in time and space. Due to the first derivative, an initial condition is needed: the initial temperature defined in the whole space of interest. Due to the second derivative, two boundary conditions are needed. The spatial second derivative can be approximated with any method, in this example a central difference is used:

$$\left.\frac{dT}{dt}\right|_{x_i,t} = \frac{\nu}{\Delta x^2}\left(T_{i-1}^t - 2T_i^t + T_{i+1}^t\right) \text{ for } i = 1, 2, \dots n-2, n-1$$

The counter of that equation does not start at i=0 and it does not end at $n$ due to the BC. For constant Dirichlet conditions, $T_{0/n}^t = T_{left/right}$ at all times. Let's consider a problem with a grid of 5 points with only three unknowns (again due to the BC), the system of ODEs becomes:

$$\frac{dT_1^t}{dt} = \frac{\nu}{\Delta x^2}\left(T_0 - 2T_1^t + T_2^t\right)$$

$$\frac{dT_2^t}{dt} = \frac{\nu}{\Delta x^2}\left(T_1^t - 2T_2^t + T_3^t\right)$$

$$\frac{dT_3^t}{dt} = \frac{\nu}{\Delta x^2}\left(T_2^t - 2T_3^t + T_4\right)$$

# Solution with Forward Euler for the time discretization

Then, the time derivative is approximated, with any method, here we use the explicit method Forward Euler:

$$\frac{T_i^{j+1}-T_i^j}{\Delta t}=\frac{\nu}{\Delta x^2}\left(T_{i-1}^j-2T_i^j+T_{i+1}^j\right) \text{ for } i=1,2,\dots n-2,n-1; \text{ for } j=0,1,2$$

To approximate the temperature for $t=1$, the system of equations starts at $j=0$, as it is an explicit method, we can already solve for the next time step:

$$T_1^1=T_1^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_0-2T_1^0+T_2^0\right)$$

$$T_2^1=T_2^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_1^0-2T_2^0+T_3^0\right)$$

$$T_3^1=T_3^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_2^0-2T_3^0+T_4\right)$$

The solution for $T_{1,2,3}^1$ is directly found because $T_{1,2,3}^0$ are known thanks to the initial condition.

# Solution with Backward Euler for the time discretization

If instead of an explicit Forward Euler, we use an implicit scheme like Backward Euler, the system of ODEs becomes:

$$T_1^1=T_1^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_0-2T_1^1+T_2^1\right)$$

$$T_2^1=T_2^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_1^1-2T_2^1+T_3^1\right)$$

$$T_3^1=T_3^0+\frac{\nu\Delta t}{\Delta x^2}\left(T_2^1-2T_3^1+T_4\right)$$

Separating the unknowns:

$$T_1^1-\frac{\nu\Delta t}{\Delta x^2}\left(-2T_1^1+T_2^1\right)=T_1^0+\frac{\nu\Delta t}{\Delta x^2}(T_0)$$

$$T_2^1 - \frac{\nu \Delta t}{\Delta x^2}\left(T_1^1 - 2T_2^1 + T_3^1\right) = T_2^0$$

$$T_3^1 - \frac{\nu \Delta t}{\Delta x^2}\left(T_2^1 - 2T_3^1\right) = T_3^0 + \frac{\nu \Delta t}{\Delta x^2}(T_4)$$

This can be written as $Ay = b$ and can be solved in a similar way as the second order ODE of the previous chapter:

$$\begin{bmatrix} (1+2\frac{\nu\Delta t}{\Delta x^2}) & -\frac{\nu\Delta t}{\Delta x^2} & 0 \\ -\frac{\nu\Delta t}{\Delta x^2} & (1+2\frac{\nu\Delta t}{\Delta x^2}) & -\frac{\nu\Delta t}{\Delta x^2} \\ 0 & -\frac{\nu\Delta t}{\Delta x^2} & (1+2\frac{\nu\Delta t}{\Delta x^2}) \end{bmatrix} \begin{bmatrix} T_1^1 \\ T_2^1 \\ T_3^1 \end{bmatrix} = \begin{bmatrix} T_1^0 + \frac{\nu\Delta t}{\Delta x^2}(T_0) \\ T_2^0 \\ T_3^0 + \frac{\nu\Delta t}{\Delta x^2}(T_4) \end{bmatrix}$$

For the $T * 2_{1,2,3}$ the solution can be found in the same way given that now $T * 1_{1,2,3}$ are known.

# Procedure Summary

Some of the following steps can be re ordered without affecting the solution. The following structure is proposed:

1. Discretization of the spatial derivative with a numerical approximation
2. Discretization of the time derivative with a numerical approximation
3. Parameter definition
4. Grid creation
5. Define Initial conditions
6. Define Boundary conditions
7. Building a system of equations according to the discretization
8. Solve the system
9. Go back to 7 until the final time step, $m$, is reached.

> **Attribution**
>
> This chapter is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame. Find out more here.

# Univariate Continuous Distributions

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

Civil engineering and Geosciences involve working with nature and that comes with associated uncertainties. Natural phenomena have an intrinsic aleatoric uncertainty that we need to model. Moreover, we don't always have the resources to neglect the epistemic uncertainty; we typically need to make decisions in scenarios of data scarcity.

This is where probability in general and continuous distributions in particular come in indispensable as they model the uncertainty and allow to assess the risk in our decisions and incorporate it in the decision-making process.

# PDF and CDF

## Contents


- Probability Density Function (PDF)
- Cumulative Distribution Function (CDF)
- PDF and CDF of Gaussian distribution
- Probability of other intervals
- Inverse CDF


## Probability Density Function (PDF)

To mathematically describe the distribution of probability for a continuous random variable, we define the probability density function (PDF) of $X$ as $f_X(x)$, such that

$$f_X(x)dx = P(x < X \leq x + dx)$$

To qualify as a probability distribution, the function must satisfy the conditions $f_X(x) \geq 0$ and $\int_{-\infty}^{+\infty} f_X(x)dx = 1$, which can be related to the axioms. Note that in this case we use lower case $x$ as the argument of the PDF, and upper case $X$ denotes the random variable. Similarly, the function $f_Y(u)$ describes the PDF of the random variable $Y$. For notational convenience we will omit the subscript in the remainder of this chapter.

## Cumulative Distribution Function (CDF)

It's important to realize that while the PDF describes the distribution of probability across all values of the random variable, probability density is not equivalent to probability. The density allows us to quantify the probability of a certain interval of the continuous random variable, through integration. In the equation below, the mathematical relationship between the CDF (denoted here as $F(x)$) and the PDF (denoted as $f(x)$) is shown.

$$F(x) = \int_{-\infty}^{x} f(x)dx$$

The definition of the CDF includes an integral that begins at negative infinity and continues to a specific value, $x$, which defines the interval over which the probability is computing. In other words, **the CDF gives the probability that the random variable $X$ has a value less than $x$.**

Below, you find an interactive element that illustrates the relationship between the integral of the pdf and the cdf value. They grey-shaded area in the left subplot corresponds to the integral from $-\infty$ to $x$. Move your mouse over either the subplots and try to develop an intuition for how both distributions relate to each other. When is the cdf steep, when is it flat?

> **Attribution**
>
> This interactive figure is created by Max Ramgraber. Find out more here.

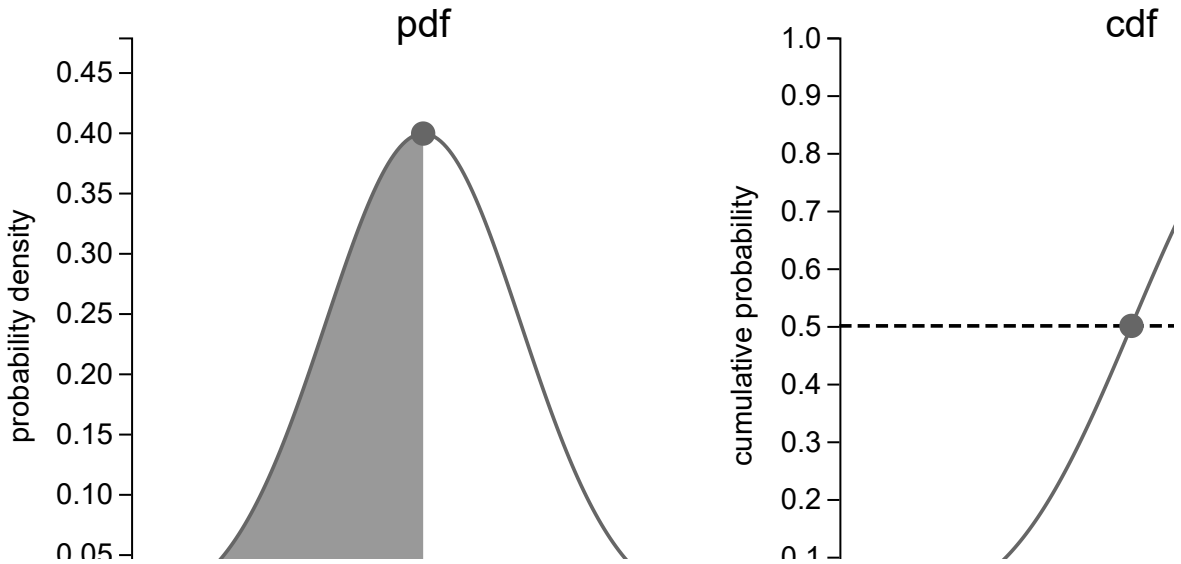



*Fig. 5.1* Interactively visualize the relationship between the PDF and the CDF (the integral of the PDF value).

It should be easy to see from the definition of the CDF that the probability of observing an exact value of a continuous random variable is exactly zero. This is an important observation, and also an important characteristic that separates continuous and discrete random variables.

# PDF and CDF of Gaussian distribution

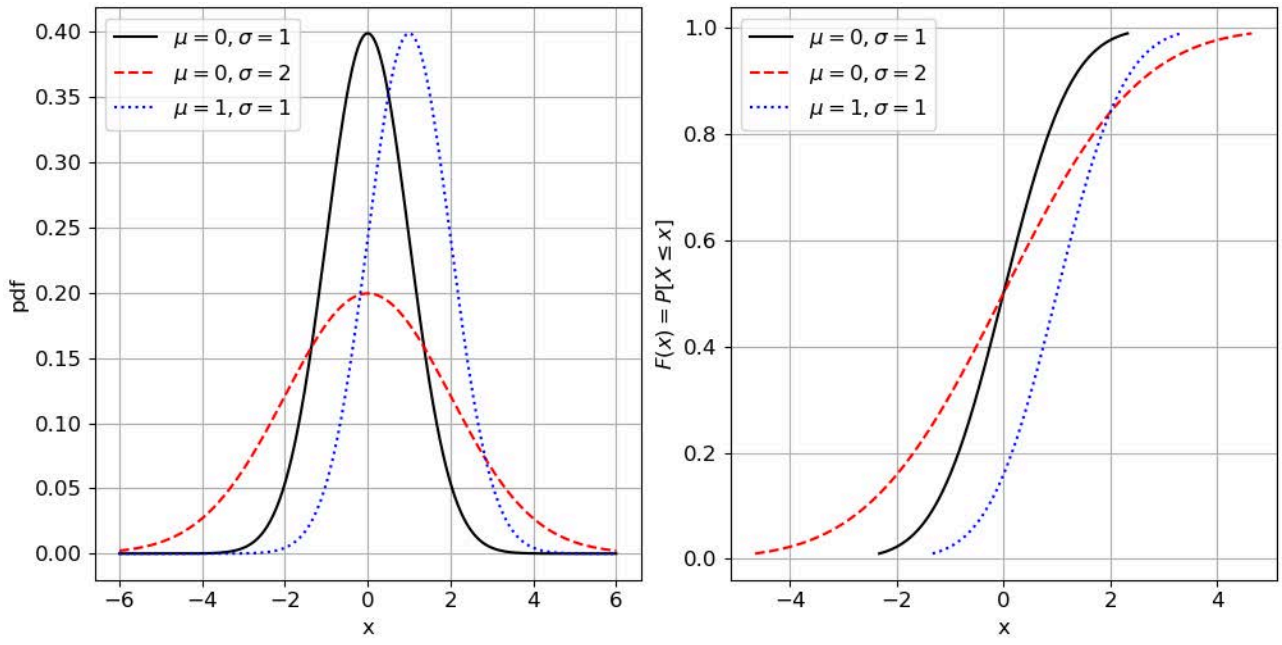
You have already extensively used one parametric distribution! Does it ring the bell ‘the bell curve’? During your BSc, you have probably used the Normal or Gaussian distribution, whose PDF presents a bell shape. The PDF of the Normal distribution is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

where $x$ is the value of the random variable and $\mu$ and $\sigma$ are the two parameters of the distribution. Thus, you can see that through the previous equation we establish a relationship between the probability densities and the values of the random variable. In the case of the Normal distribution, the parameters $\mu$ and $\sigma$ correspond to the mean and standard deviation of the random variable. However, this is not the case for all the distributions and it is also dependent on how it is parameterized.

As you have already seen, the previous expression provides us with probability densities, so we need to integrate it to obtain actual probabilities through the CDF (non-exceedance probabilities). In the case of the Normal distribution, there is no closed form of the CDF (the integral).

Let’s see how the distribution looks. In the figure below, the PDF and CDF of the Gaussian distribution are shown for different values of its parameters. In the PDF plot, you can see the bell shape that was already mentioned. You will learn more about how this distribution behaves later on.



System Message: WARNING/2 (/home/runner/work/book/book/book/probability/PDF_CDF.md, line 60)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/gaussian.png": Can only read local images.

*Fig. 5.2* Gaussian distribution function: PDF and CDF.

Below, you will find an interactive element that allows you to explore the influence of different means and standard deviations on the PDF and CDF. Experiment with both options and observe how it affects the shape of both distributions.

> **Attribution**
>
> This interactive figure is created by Max Ramgraber. Find out more here.

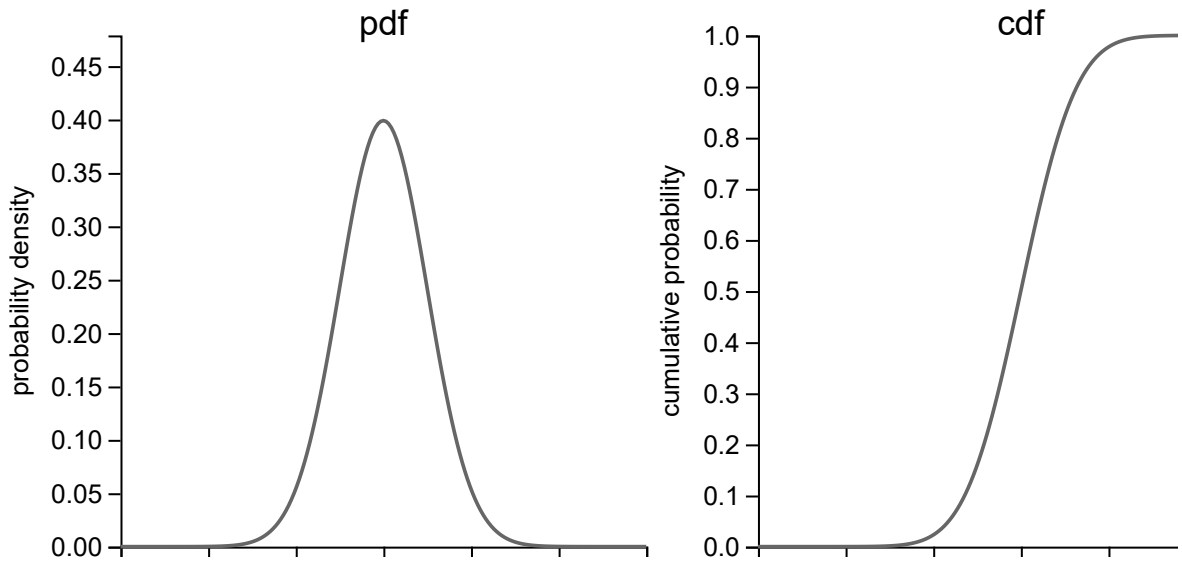



*Fig. 5.3* Interactively change the mean and standard deviation of the Gaussian distribution to visualize the effect on the PDF and CDF.

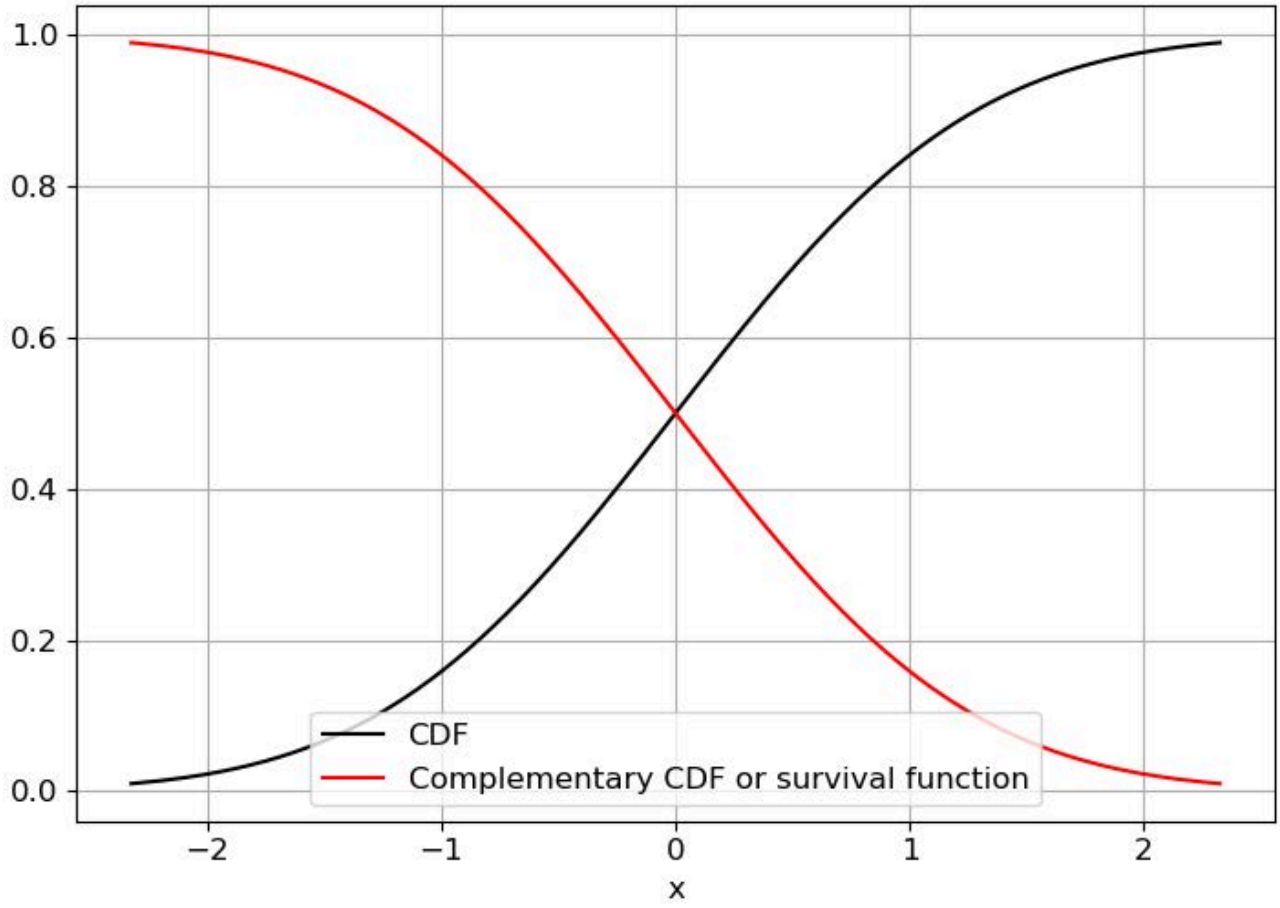
# Probability of other intervals

We saw that the CDF provides us with the non-exceedance probabilities, this is $[-\infty, x]$. But what happens if we are interested in the probabilities of another intervals? It is common to be interested in the probability of exceeding a value. For instance, wind speeds over a

value can damage an structure or concentrations of a nutrient higher than a value can lead to eutrophication. Therefore, we want to integrate from a value $x$ to $\infty$. Here the probability axioms make this easy, since the PDF integrates to 1 over the sample space of the random variable:

$$\int_x^{+\infty} f(x)dx = 1 - \int_{-\infty}^{x} f(x)dx = 1 - F(x)$$

The figure below shows both the CDF and the complementary CDF.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/PDF_CDF.md, line 98)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/survival.png": Can only read local images.

*Fig. 5.4* Gaussian distribution function: CDF and survival function or complemetary CDF.

Thus, the *exceedance probabilities* can be directly computed by substracting to 1 the non-exceedance probabilities obtained from the CDF. The result is called the *complementary CDF*. However, this function has many alternative names. The name *survival function* may

sound odd due to its positive connotation, but this is appropriate when the random variable describes, for example, the lifetime of a structure.

Another interval is that between two values, $x_1$ and $x_2$ (where $x_2 > x_1$). Using the CDF, $F(x_2)$, gives the probability of values below $x_2$ but also those below $x_1$. Then, we need to substract $F(x_2) - F(x_1)$ to obtain the probability of being in the interval $[x_1, x_2]$. In mathematical terms:

$$\int_{x_1}^{x_2} f(x)dx = \int_{-\infty}^{x_2} f(x)dx - \int_{-\infty}^{x_1} f(x)dx = F(x_2) - F(x_1)$$

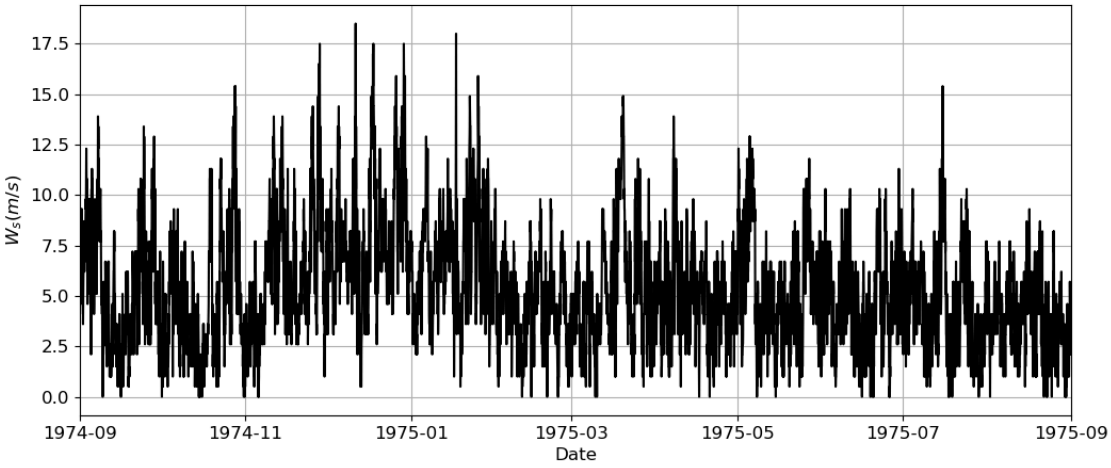
# Inverse CDF

Often, in regulations and guidelines, it is required to design our structure or system for a value which is not exceeded more than $p$ percent of the time. Thus, we are facing the opposite problem: what is the value of the random variable, $x$, whose non-exceedance probability has a specified value, $p$? The solution is simple: the inverse of the CDF, $x = F^{-1}(p)$. As previously mentioned, the CDF is just an equation which in most occasions can be solved analytically, so we just need to work through the formula and calculate $x$ given $p$.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Empirical Distributions

As you can imagine, it is possible to define a PDF and a CDF based on observations. Let's see it with an example dataset of wind speeds close to Schiphold Airport. The figure below shows the dataset which spans for 1 year.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/empirical.md, line 6)

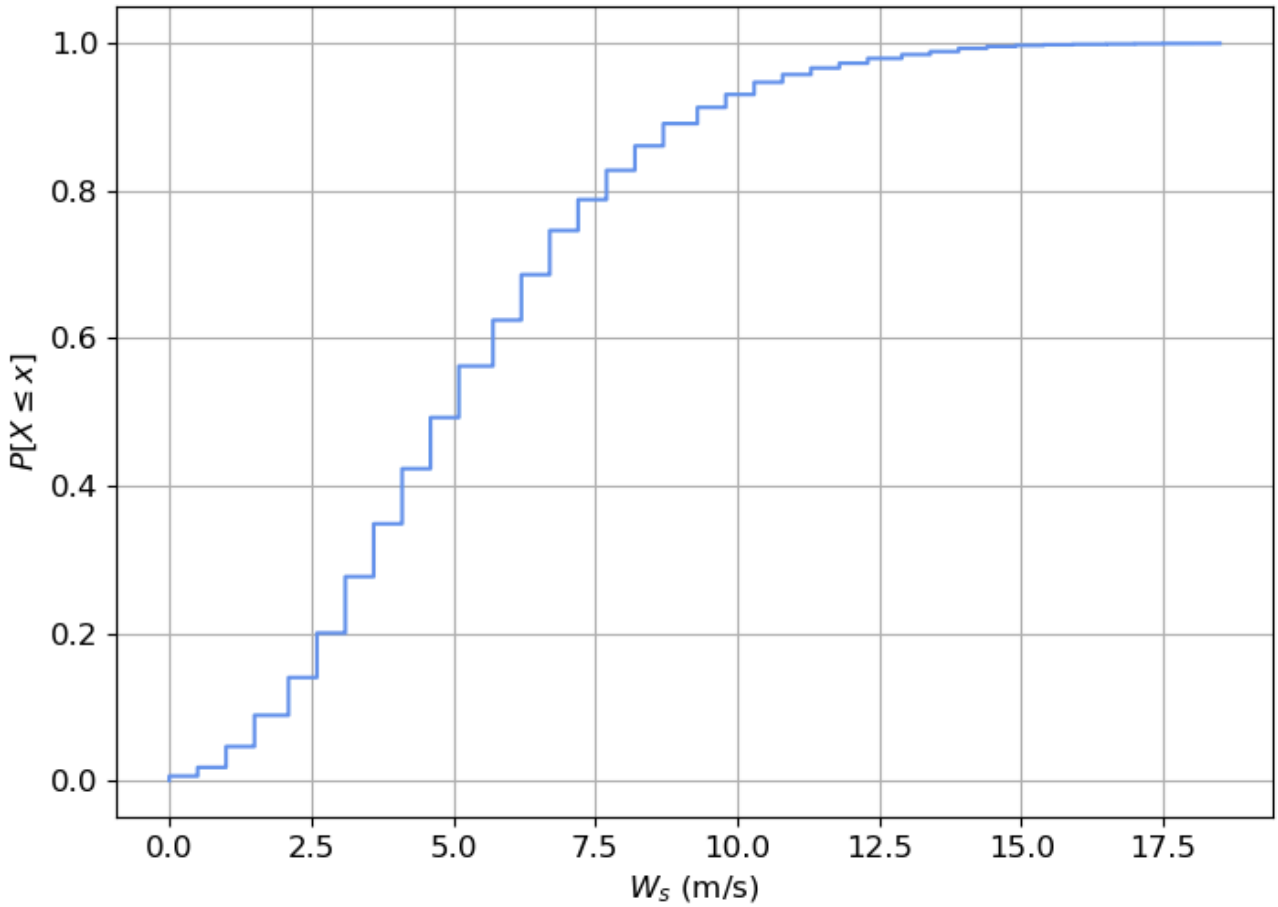
Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/data_overview.png": Can only read local images.

*Fig. 5.5* Time series of wind speed close to Schiphol Airport.

Let's start computing the empirical CDF. We need to assign to each observation a non-exceedance probability. To do so, we just need to sort the observations and compute the non-exceedance probabilities using the ranks. This is illustrated below with pseudo-code.

```
read observations

x = sort observations in ascending order

length = the number of observations
probability of not exceeding = (range of integer values from 1 \
                                to length) / length + 1

plot x versus probability of not exceeding
```

Using the above algorithm, the following figure is obtained. Note that empirical CDFs are usually plotted using a step plot.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/empirical.md, line 29)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/ecdf_wind.png": Can only read local images.

*Fig. 5.6* Empirical cumulative distribution function of the wind speed data.

It can be useful to also visualize the empirical PDF. As mentioned above, the PDF is the derivative of the CDF, leading to the following equation.

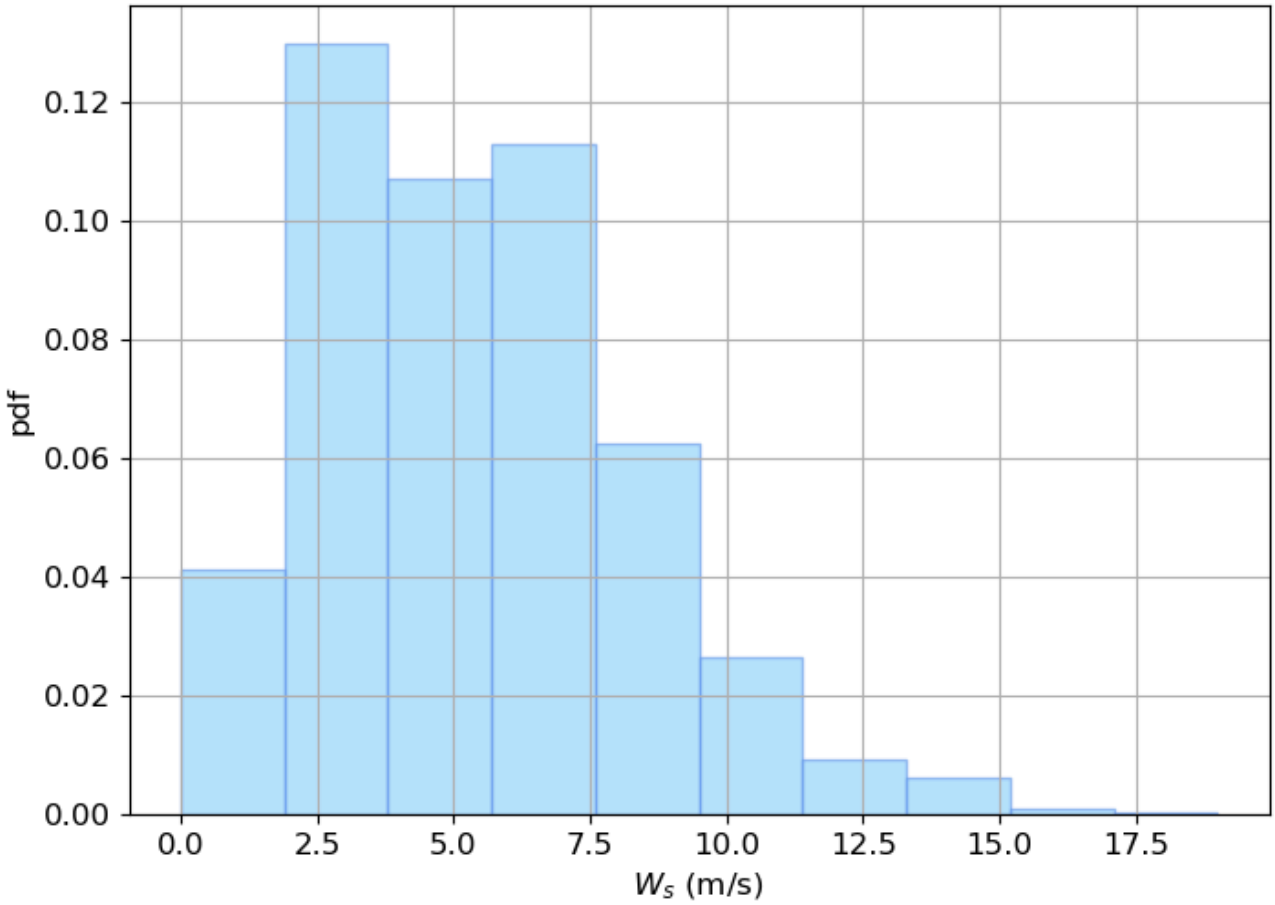
$$f(x) = F'(x) = \lim_{\Delta x \to 0} \frac{F(x + \Delta x) - F(x)}{\Delta x}$$

Thus, we can compute the empirical PDF assuming a bin size. To do so, we need to count the number of observations in each bin and calculate the relative frequency of each bin by dividing that count with the total number of observations. The density will be then those

relative frequencies divided by the bin size. This process is illustrated with the following pseudo-code [1].

```
read observations

#Assume the bin size
bin_size = 2

#Calculate the number of bins and the bin edges given the bin size
min_value = minimum value of observations
max_value = maximum value observations
n_bins = (max_value - min_value) / bin_size
bin_edges = range of n_bins + 1 values between the truncated value \
            of min_value and the ceiling value of max_value

#Count the number of observations in each bin
count = empty list
for each bin:
    append the number of observations between the bin_edges to count

#Compute relative frequencies
freq = count / number of observations

#Compute densities
densities = freq / bin_size

#plot epdf
barplot densities
```

Using the above algorithm, the following figure is obtained. We can see that most of the density is concentrated between 2 and 9 m/s.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/empirical.md, line 75)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/epdf_wind.png": Can only read local images.

*Fig. 5.7* Empirical probability density function of the wind speed data.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

Happily, in most coding languages, the algorithm to compute the pdf is already implemented and we just need to plot a histogram selecting the option to show us the densities.

# Parametric Distributions

In the previous section, you were introduced to the concepts of random variable, probability density function (PDF) and cumulative distribution function (CDF) and how to compute them using empirical data. Here, the concept of parametric distribution as a model of the observed empirical distribution is introduced.

Parametric distributions functions are mathematical models for the empirical distributions that we observe in our data. This is, a parametric CDF is just an equation which relates the non-exceedance probability with the value of the studied random variable. This equation has some parameters or coefficients that need to be fitted using our observations.

**But why do we need them?**

We typically fit a parametric distribution to our data for several reasons. The most important one is that the empirical distribution is limited to the observations we have. Using the empirical CDF, we can interpolate between the observed values, but we cannot extend it further and infer probabilities higher or lower than those we have observed.

Another good reason to fit a parametric distribution is more on the practical side: an equation allows us to use all the power of analytic solutions and it is very easy to transfer and handle. Also, we can make use of the properties of the fitted distribution to have a further insight on the random variable we are studying.

In the following sections, you will be introduced to some of the most commonly used distributions in engineering and geosciences and their properties.

**MUDE exam information**

You do not need to know the equations of the distribution functions by heart. You just need to know how the distribution looks (PDF/CDF), how it responds to changes in the parameters and some basic properties (symmetry or bounds).

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Revisiting Gaussian distribution

## Contents


- Some properties


You have already used Gaussian or Normal distribution during your BSc and this course since it is widely used in Observation Theory or to model errors in measurements. The PDF of the Normal distribution is given by

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

where $x$ is the value of the random variable and $\mu$ and $\sigma$ are the two parameters of the distribution. These parameters, $\mu$ and $\sigma$, correspond to the mean and standard deviation of the random variable.

If we integrate it, we obtain the CDF (non-exceedance probabilities). In the case of the Normal distribution, there is no closed form of the CDF (the integral), but it can be expressed as

$$F(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right)$$

where erf denotes the error function given by

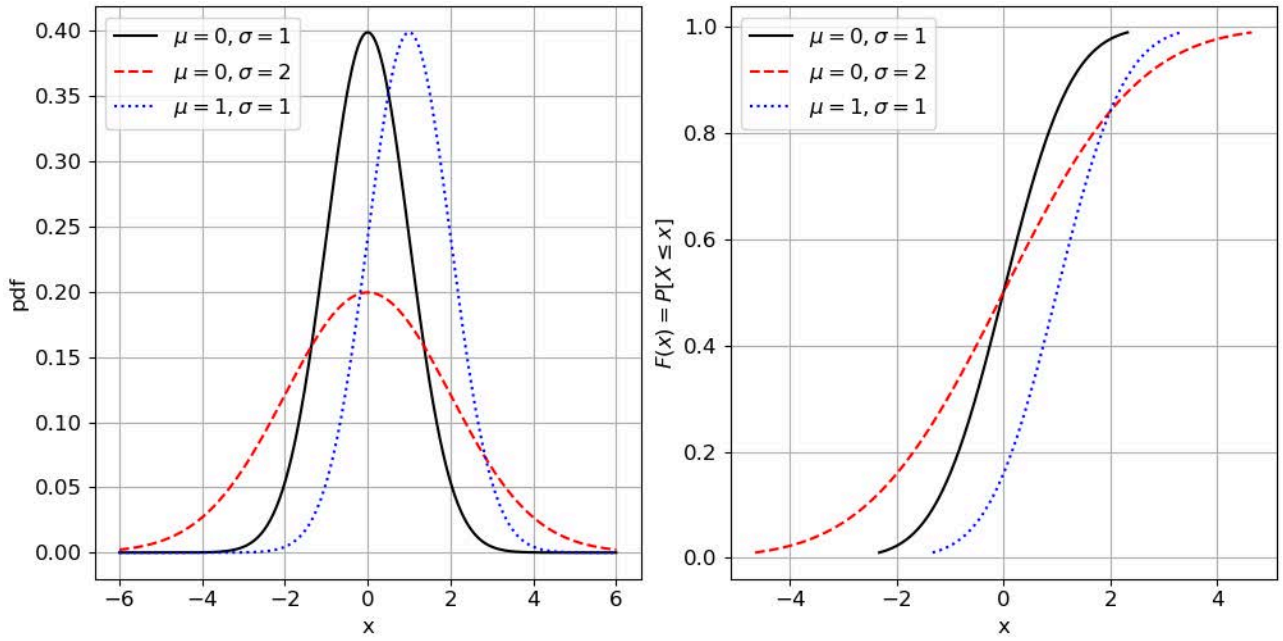
$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

Let’s see how the distribution looks.

In the figure below, the PDF and CDF of the Gaussian distribution are shown for different values of its parameters. In the left pannel, the PDF of three Gaussian distributions is

shown. The black and blue distributions present the same value of the standard deviation ($\sigma$ =1), so in the PDF plot the width of the bell is the same. However, they have different values of the mean ($\mu$), which acts like a location parameter. Thus, increasing the mean moves the distribution towards the right, making more likely higher values of the random variable. You can also see that in the CDF plot. The distribution moves towards the right so for a given value, $x = 2$, $F(x \leq 2) \approx 0.98$ for the black line and $F(x \leq 2) \approx 0.84$ for the blue line.

Regarding the standard deviation ($\sigma$), it can be interpreted as the dispersion around the mean ($\mu$). Thus, you can see in the PDF plot the the red distribution is wider that the black or blue ones, since the standard deviation is the double of the other two. You can also see the effect in the CDF plot, where the slope of the red distribution is more gentle than those of the black and blue distributions.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/Gaussian.md, line 31)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/gaussian.png": Can only read local images.

*Fig. 5.8* Gaussian distribution function: PDF and CDF.

# Some properties

The mean, median and mode of the Normal distribution is equal to $\mu$. The variance is $\sigma^2$. Note that Normal distribution is symmetric and presents 0 skewness.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Non-Gaussian distributions

## Contents


- Concepts of asymmetry and tail of the distribution


So far, you have seen that we can use a parametric distribution to model the uncertainty in our observations. Also, you have revisited one of those parametric distributions, the Gaussian distribution, which is a widely used model since it has lots of applications such as modelling natural processes or measurement errors. Here, we will discuss an interesting property of this distribution, the lack of **asymmetry**, and introduce the concept of the **tail of the distribution**.

## Concepts of asymmetry and tail of the distribution

Asymmetry is a relevant property of a distribution which analyzes its shape. In the Figure below, you have the PDF of a Gaussian distribution.

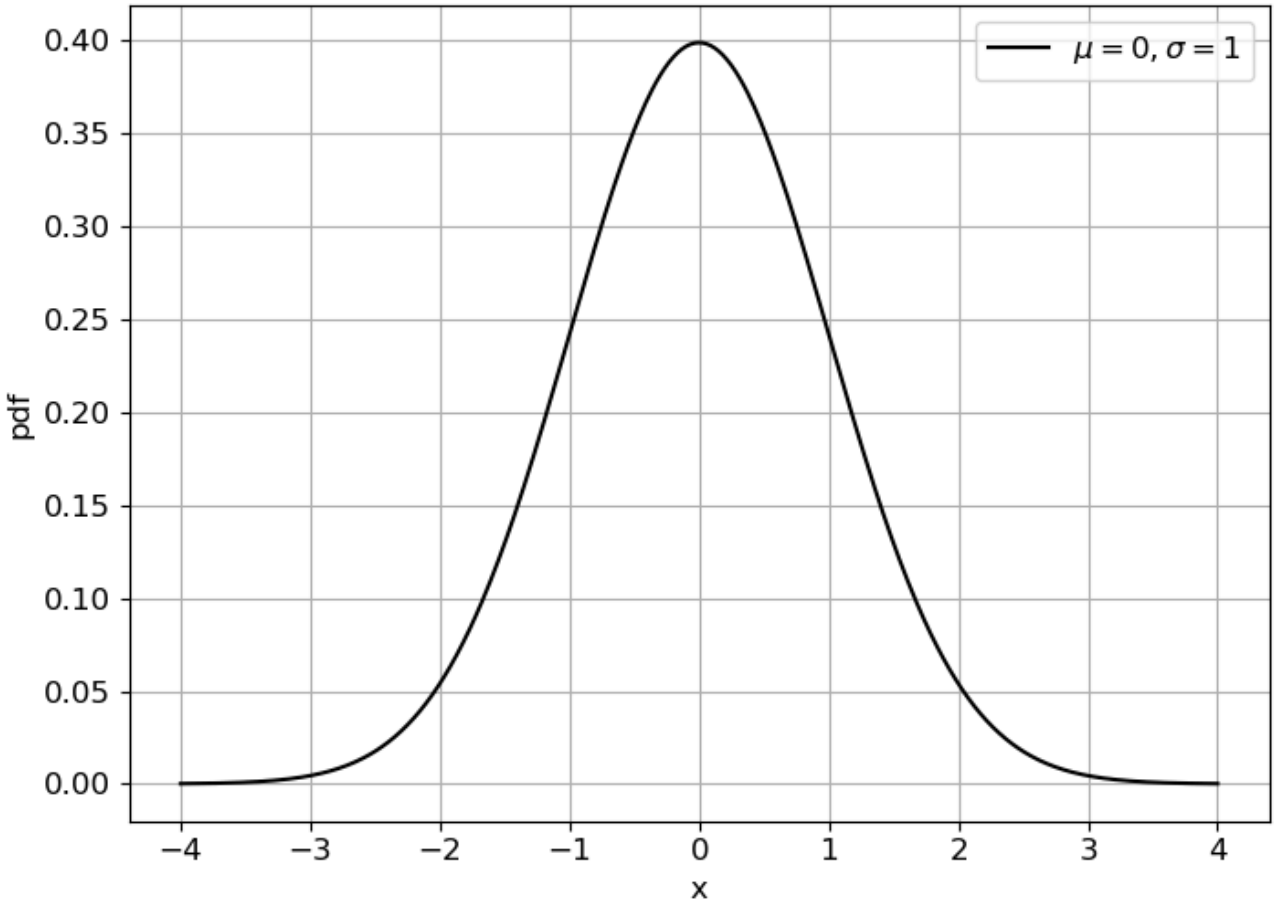



*Fig. 5.9* Gaussian PDF.

As you can see, the shape of the PDF is perfectly symmetric with respect to the mode[1] of the distribution. This leads to an interesting property: for perfectly symmetric distributions the mode, mean[2] and median[3] have the same value.

As you can imagine, it is also possible to analyze the asymmetry of an empirical distribution. To this end, we can visually inspect the empirical PDF and compute the *sample coefficient of skewness* from a set of observations as

$$g_1 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^3}{n\sigma^3}$$

where $x_i$ are the observations, $\overline{x}$ is the mean of the observations, $n$ is the number of observations, and $\sigma$ is the standard deviation.

**And what happens when my observations are not symmetric?**

In that case, Gaussian distribution may not be the most appropriate model and we need to look for a different model. Actually, it is very common to find asymmetric distributions when modelling natural processes!

If we go back to the wind data that we previously introduced and have a look at the PDF, we can easily notice that it is not symmetric. The mass in the PDF is not distributed equally at both sides of the mode. Actually, the shape is similar to a bell with the center around $W_s = 5m/s$, but we can see that the distribution expands towards the positive values creating what we call the **tail of the distribution**. Since the tail expands towards positive values, we say that the PDF has *positive skewness*.

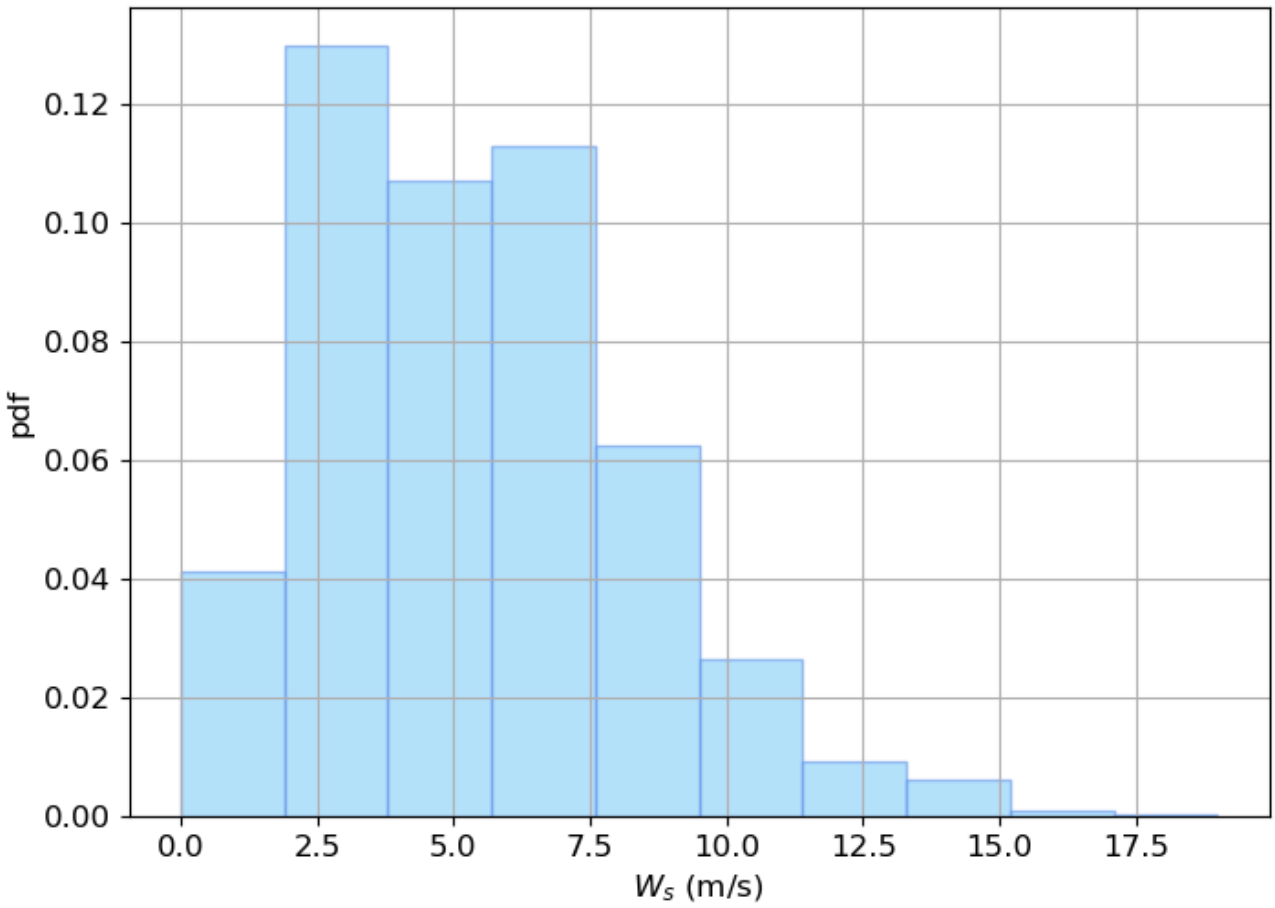



*Fig. 5.10* Empirical probability density function of the wind speed data.

Many natural phenomena present positive skewness. This means that the mode, the mean and the median do not have the same value anymore. For a positively skewed PDF: mode < median < mean. This inequality is reversed if the PDF is negatively skewed.

**And why are asymmetry and tail that important?**

As you can imagine, asymmetry is an important criteria when selecting the parametric distribution to model the uncertainty in our observations. The shape of the parametric distribution needs to fit the shape of the empirical distribution function.

Moreover, the tail of the distribution is modelling the highest and lowest values of our variable and, typically, they are the ones we use for design. For instance, if we are designing a building, we would be interested in the high wind speeds that it has to withstand (loading

conditions). Also, if we are considering the nutrients required to cultivate mussels, we would analyze the lowest values to ensure that the mussels have enough food to survive. Thus, it is important to choose a parametric distribution which is able to model the behavior in the tail when the observations are not symmetric.

In the following sections, we will introduce some common parametric distributions, as well as their characteristics, to give an overview of some possible models you may apply to model your data. But keep in mind that the list is not exhaustive, there are a lot of distributions!

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

---

The mode is the most frequently observed value and, thus, the value of the distribution with the highest value of the density in its PDF.

Visually from the PDF, we can define the mean as the value of the random variable which is the balance point of the distribution.

Visually from the PDF, we can define the median as the value of the random variable which divides the PDF in two areas of the same value.

# Uniform distribution

## Contents


- Some properties


The uniform distribution is the simplest type of continuous parametric distribution and, as it is implied in its name, the PDF has a constant value along a given interval $[a, b]$, where $a < b$. The PDF of the uniform is given by

$$f(x) = \frac{1}{b-a} \qquad \text{for } a \leq x \leq b$$

$$f(x) = 0 \qquad \text{otherwise}$$

Note that all values in the distribution are between the lower limit $a$ and the higher limit $b$ and are equally likely to occur. The left pannel of the figure below, shows an example of a uniform PDF with $a = -1$ and $b = 1$.

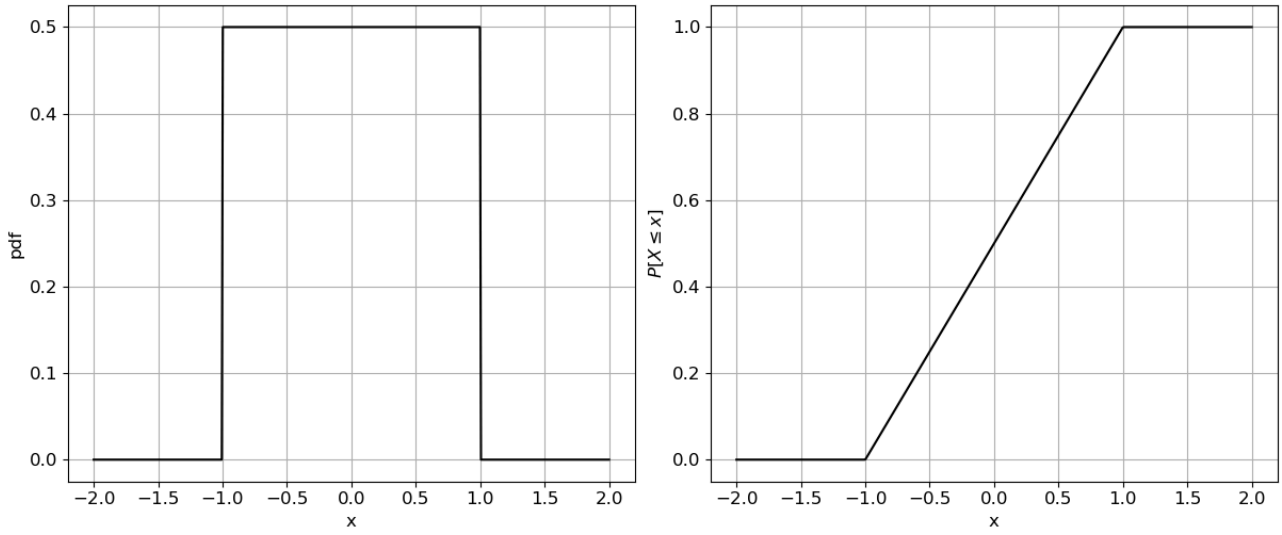



*Fig. 5.11* Uniform distribution function: (left) PDF, and (right) CDF.

As seen in previous chapters, the CDF is the integral of the PDF. Thus, as we integrate over a constant, the CDF presents a linear shape, as shown in the right pannel of the figure above. Thus, the CDF is given as

$$F(x) = 0 \qquad \text{for } x < a$$

$$F(x) = \frac{x - a}{b - a} \qquad \text{for } a \leq x \leq b$$

$$F(x) = 1 \qquad \text{for } x > b$$

If we make $a = 0$ and $b = 1$, we obtain the standard or unity uniform distribution, which is used to generate random values from other distribution functions for simulation purposes.

# Some properties

The mean of the uniform distribution can be computed based on its simple geometry as

$$E[X] = \frac{1}{2}(a + b)$$

The variance is given by

$$Var[X] = \frac{1}{12}(b - a)^2$$

Finally, note that uniform distribution is symmetric and presents 0 skewness. Thus, the median and the mean are identical. This is, it does not present any tail.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Exponential distribution

## Contents


- Some interesting properties


Another widely used distribution function is the Exponential distribution. For instance, it is applied to model the waiting time between succesive events of a Poisson process. The PDF of the Exponential distribution is given by

$$f(x) = \lambda e^{-\lambda x} \qquad for\ x \geq 0, \lambda > 0$$

$$f(x) = 0 \qquad otherwise$$

where $\lambda$ is the parameter of the distribution, which is often called *rate*. In the right pannel of the figure below, an example of two Exponential distributions with $\lambda = 1$ and $\lambda = 2$ is shown. As you can see, the maximum density in the PDF of en Exponential distribution is located at zero and it is followed by an Exponential decay. The higher the parameter $\lambda$, the higher the value of the density in $x = 0$ and the faster the decay. In other words, the higher the parameter $\lambda$, the more concentrated the values of the random variable which are likely to occur and, thus, the lower the standard deviation. This can be seen on the left pannel of the figure, where random samples of the distribution are plotted. There you can see how higher values of the random variable $x$ appear when $\lambda = 1$, presenting then a higher dispersion.

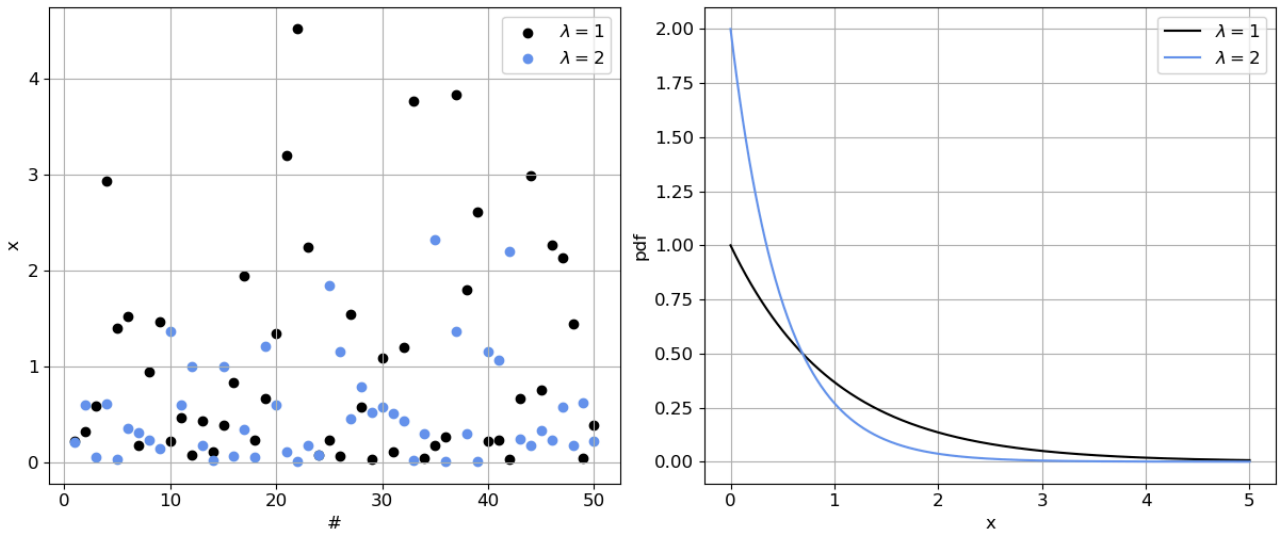


*Fig. 5.12* Exponential distribution function: (left) random samples, and (right) PDF.

Integrating the PDF, the following expression of the CDF is obtained

$$F(x) = 1 - e^{-\lambda x}$$

which is visually represented in the figure below.

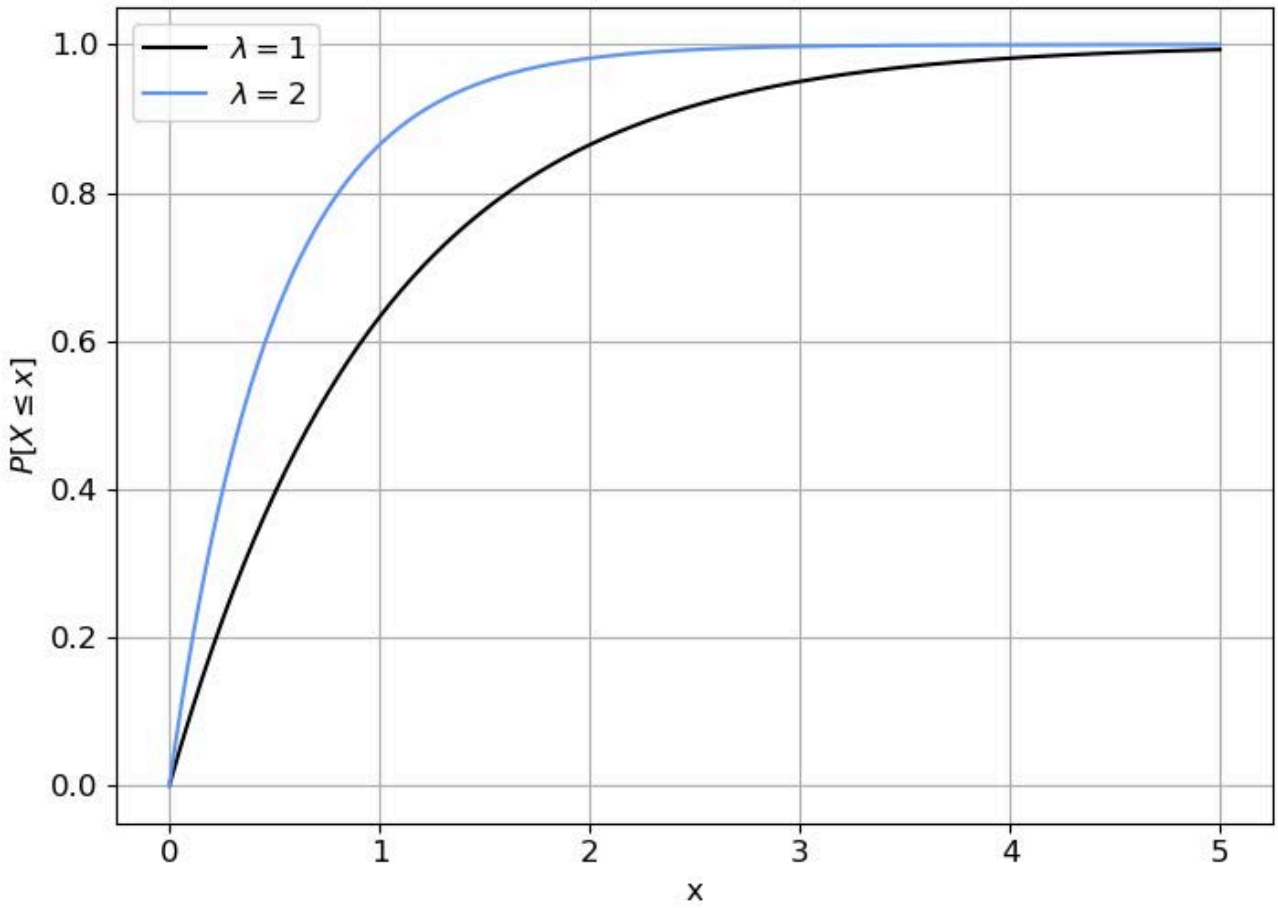



*Fig. 5.13* Exponential distribution function: CDF.

The influence of the parameter $\lambda$ can also be observed in the CDF. The higher the parameter $\lambda$, the faster the CDF reaches values of the non-exceedance probability close to 1. Thus, lower variability of the random variable is obtained.

# Some interesting properties

The mean of the exponential distribution can be obtained integrating by parts as

$$E[X] = \int_o^\infty x\lambda e^{-\lambda x} dx = [-xe^{-\lambda x}]_0^\infty + \int_0^\infty e^{-\lambda x} dx = 1/\lambda$$

As previously mentioned, the Exponential distribution models the waiting time between events (e.g.: floods or earthquakes) of a Poisson process. For the Poisson process, $\lambda$ is the rate at which the events occur. Therefore, $1/\lambda$ is the average time between those events, since it is the expectation of the associated Exponential distribution. In reliability analysis, it can be called *mean life time* or *time to failure*.

The variance is given as

$$Var[X] = E[X^2] - (E[X])^2 = 1/\lambda^2$$

It should be noted that the coefficient of variation $CV = \sqrt{Var[X]}/E[X] = 1$.

If you remember the Poisson process is characterized by independent events. Thus, the time of the next occurrence modelled by the Exponential distribution is independent of the present and past occurrences. This is called the *memoryless property of the exponential distribution*. We can show it calculating the probability of the waiting time until the next ocurrence $X$ being higher than the sum of the two previous ones $X > x_1 + x_2$, given that it is already bigger than one of them $X > x_1$ (conditional probability) as follows

$$P[X > x_1 + x_2 | X > x_1] = \frac{P[X > x_1 + x_2]}{P[X > x_1]} = \frac{e^{\lambda(x_1+x_2)}}{e^{\lambda x_1}} = e^{\lambda x_2} = P[X > x_2]$$

If the waiting time $X$ is exponential-distributed, you can see that the probability of the waiting time of one occurence ($x_2$) does not depend on the previous one ($x_1$).

Finally, note that this distribution is not bell-shaped and, thus, not symmetric. It presents a positive tail and its skewness is 2.

**Exercises**

An engineer is the manager of the construction of a bridge across a river and is concerned about large floods (floods with discharges higher than $50 m^3/s$) in the river, since they may damage the construction works. Higher floods than that occur in average every 10 years.

Assume that floods are independent and identically distributed. This means that they can be modelled using a Poisson process and, thus, the time between floods can be modelled using an Exponential distribution.

What is the rate \( \lambda \) of the Exponential distribution?

Check

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Gumbel distribution

## Contents


- Some properties
- What about a negative tail?


Gumbel distribution is widely used to model maximum and minimum values of natural phenomena, such as wave storms or floods. You will learn more about this application in the Extreme Value Analysis section. The PDF of the Gumbel distribution is given by

$$f(x) = \frac{1}{\beta}\, e^{-\left(\frac{x-\mu}{\beta}+e^{-\left(\frac{x-\mu}{\beta}\right)}\right)}$$

where $\mu$ is the location parameter and $\beta > 0$ is the scale parameter. In the figure below the influence of these parameters is presented.

In the right pannel of the figure below, an example of the PDF of three Gumbel Exponential distributions is shown. The first two distributions (black and blue continuous lines) present the same $\beta = 1$ but different $\mu_{black} = 0$ and $\mu_{blue} = 2$. This produces a shift towards the positive values of the distribution, being the mode $= \mu$. Note that the shape of the PDF is identical, so $\mu$ does not influence the dispersion of the distribution. This can also be seen in the left pannel, where random samples from the distribution are drawn. The samples coming from the distribution with $\mu_{blue} = 2$ are higher than those coming from the one with $\mu_{black} = 0$, while both present a similar dispersion.

Regarding the third distribution (red crosses in the left pannel and red dashed line in the right pannel), it presents the same $\mu = 0$ than the first distribution (black dots in the left pannel and continuous black line in the right pannel) but different $\beta$: $\beta_{black} = 1$ and $\beta_{red} = 3$. In the right pannel, it can be observed that both distributions present the same mode in $x = 0$. However, the dispersion of the distribution with $\beta_{red} = 3$ is way higher. This effect can also be seen in the random samples in the left pannel. Random samples from $\beta_{black} = 1$ range from -2 to 4 approximately, while random samples from $\beta_{red} = 3$ range from -2 to 8 approximately, having one sample a value of almost 14.

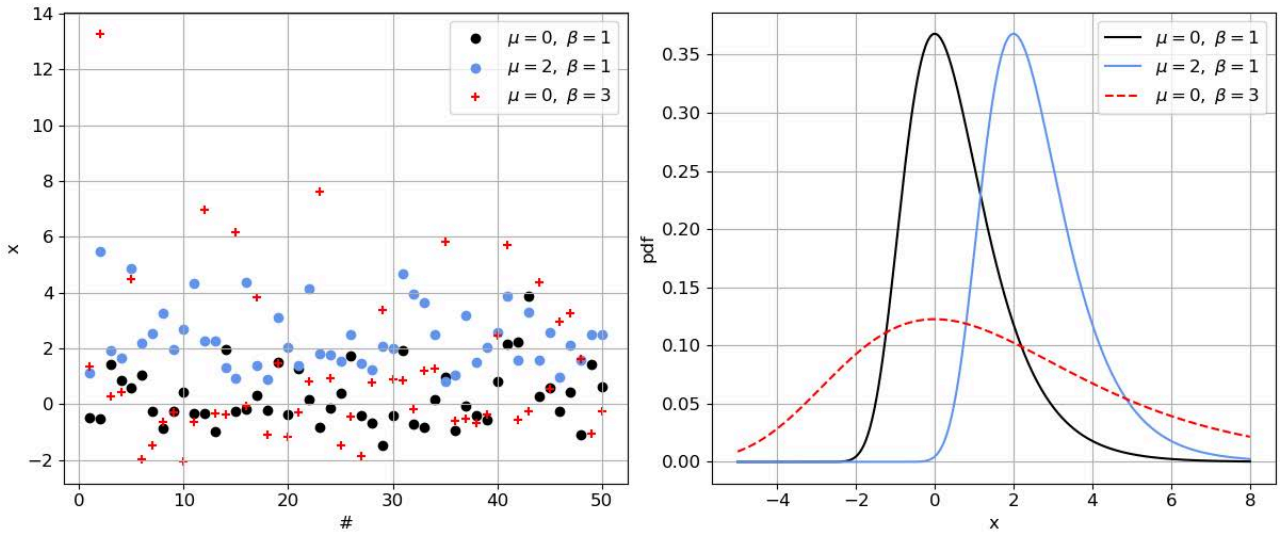



*Fig. 5.14* Gumbel distribution function: (left) random samples, and (right) PDF.

Integrating the PDF, the following expression of the CDF is derived

$$F(x) = e^{-e^{-\frac{x-\mu}{\beta}}}$$

which is displayed in the figure below.

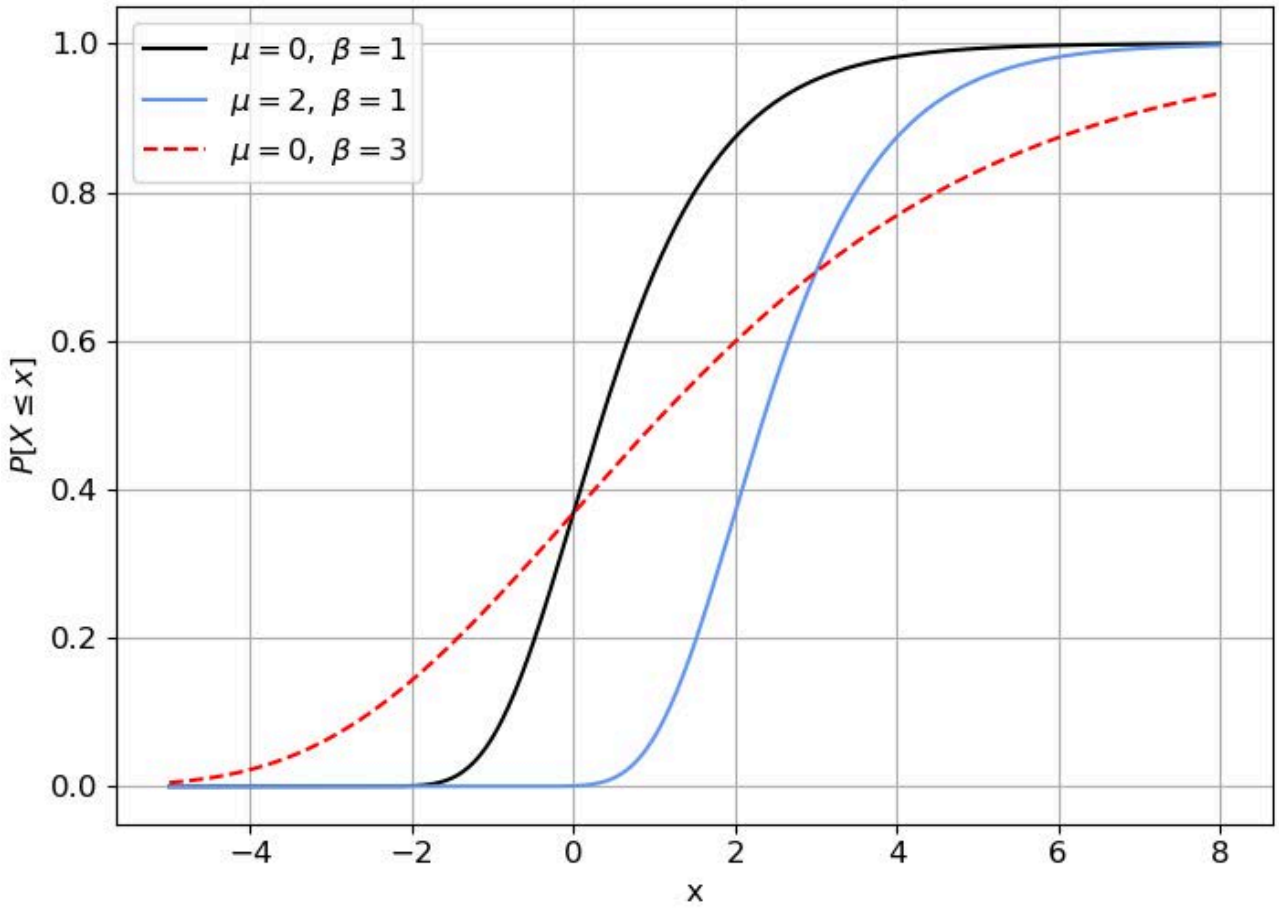



*Fig. 5.15* Gumbel distribution function: CDF.

The influence of the parameters can also be observed in the CDF. The continuous blue and black distributions present the same shape while the blue distribution is shifted towards positive values due to the higher value of $\mu$. Regarding the dashed red distribution, the slope of the CDF is way more gentle, showing a higher dispersion than the other two distributions due to the higher value of $\beta$.

# Some properties

The mean of the Gumbel distribution can be computed as

$$E[X] = \mu + \gamma\beta$$

where $\gamma \approx 0.577$ is the Euler-Mascheroni constant. The variance is given by

$$Var[X] = \frac{\pi^2}{6}\beta^2$$

Finally, note that Gumbel distribution is not symmetric and presents positive skewness. This is, it presents a tail towards positive values. Actually, the skewness of this distribution can be analytically computed and it is approximately 1.14.

# What about a negative tail?

Although less common, we can also find a variation of the Gumbel distribution with a left tail. The PDF of the left-tailed Gumbel distribution is given by

$$f(x) = \frac{1}{\beta}e^{\left(\frac{x-\mu}{\beta} - e^{\left(\frac{x-\mu}{\beta}\right)}\right)}$$

where $\mu$ is the location parameter and $\beta > 0$ is the scale parameter. In the figure below the influence of these parameters is presented. Parameter interpretation is equivalent to that for the parameters of the right-tailed one.

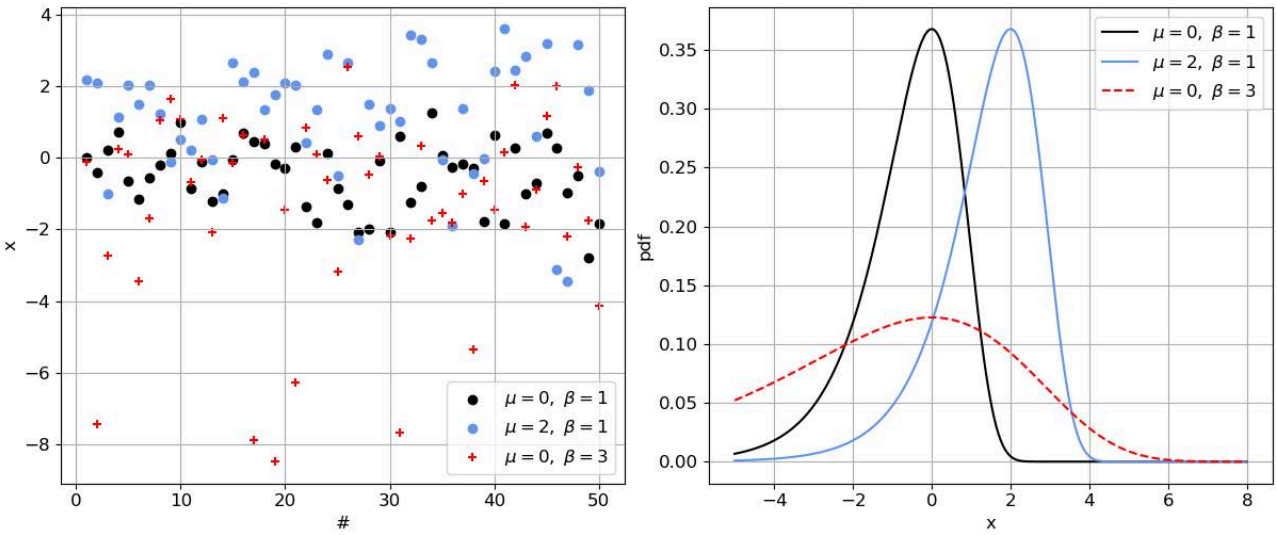



*Fig. 5.16* Left-tailed Gumbel distribution function: (left) random samples, and (right) PDF.

Integrating the PDF, the following expression of the CDF is derived

$$F(x) = e^{-e^{\frac{x-\mu}{\beta}}}$$

which is displayed in the figure below. Note how the difference in a symbol switches the distribution.

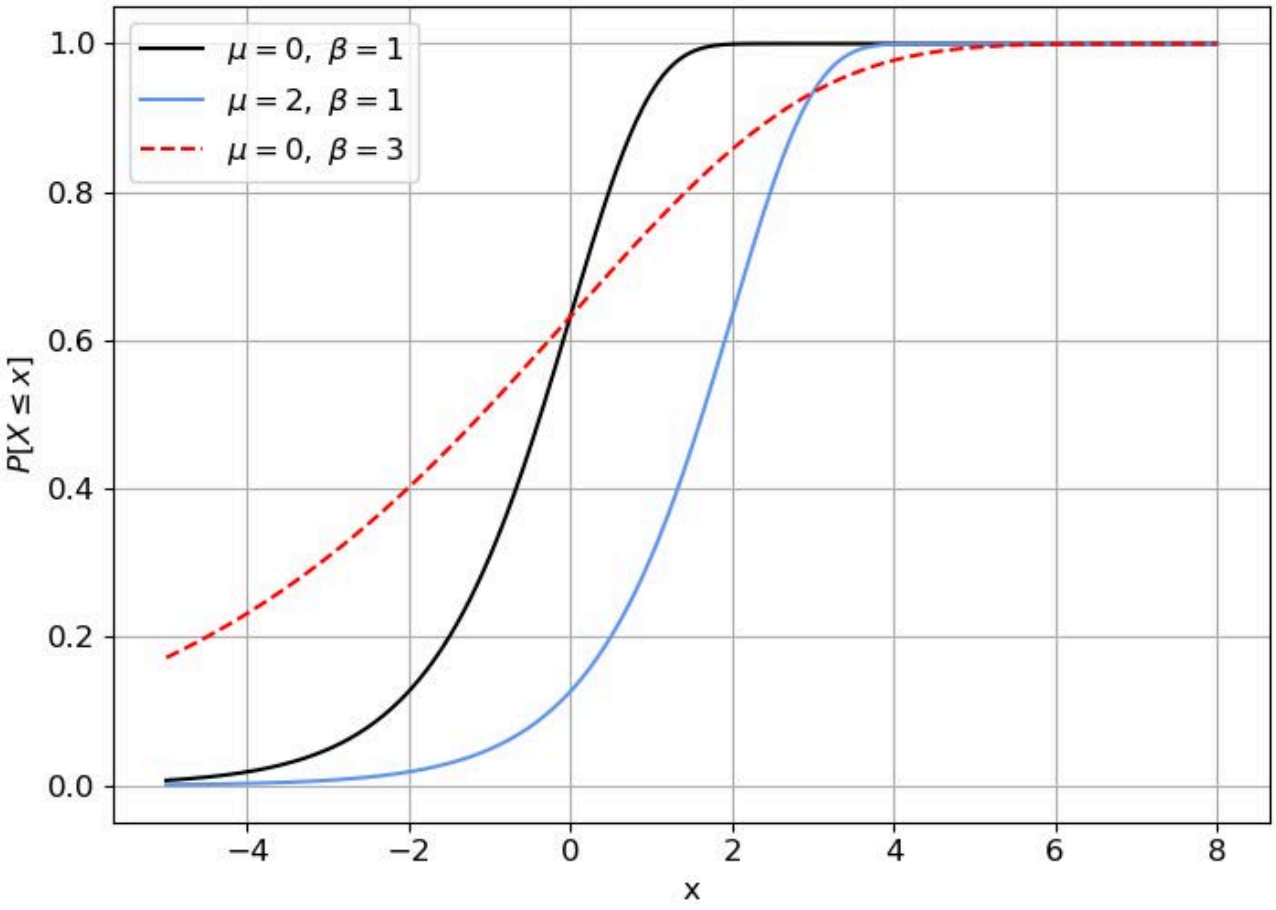


*Fig. 5.17* Left-tailed Gumbel distribution function: CDF.

Actually, if we compare the PDF of a right-tailed and left-tailed Gumbel distributions with the same parameters ($\mu$=0 and $\beta$=1 in the figure below), the mode of the distribution is the same in both distributions and one PDF is the symmetrical of the other.

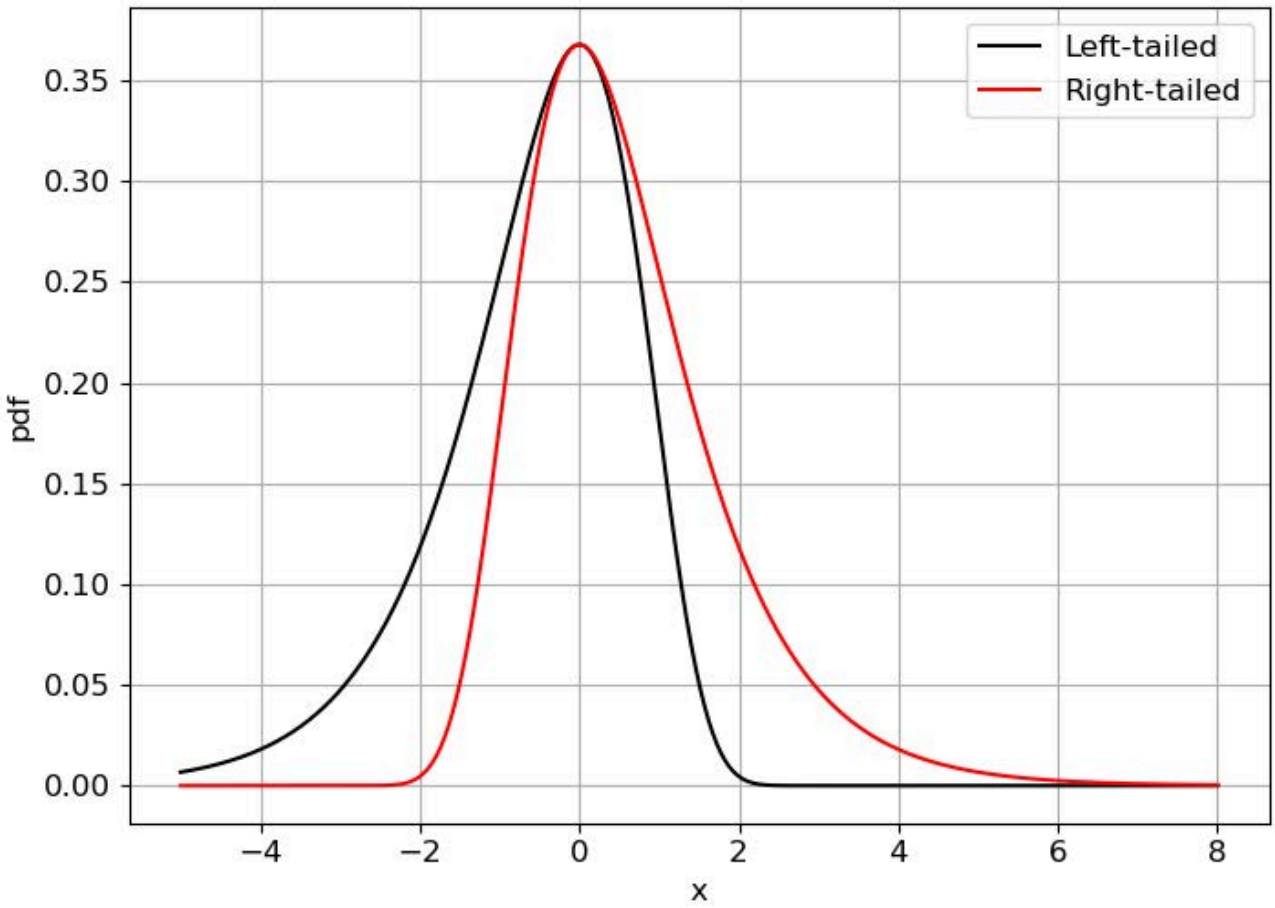



*Fig. 5.18* Comparison between the left-tailed and right-tailed Gumbel distribution.

**Exercises**

Groundwater quality of a shallow aquifer has been tracked for a year to determine if it is duable to use it or if it is getting polluted. To this end, the daily maximum concentration of chloride in miligrams per liter has been measured. These concentration has been described using a Gumbel distribution with parameters $\mu = 60$ and $\beta = 10$.

Chlorides between 1mg/L an 100mg/L are normal in freshwater. Use the Gumbel distribution to compute the probability of being in that interval.

Chlorides between 1mg/L an 100mg/L are normal in freshwater. Use the Gumbel distribution to compute the probability of being in that interval, round to 2 decimal places.

Check

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Lognormal distribution

## Contents


- Lognormal distribution
- Let's practice


Lognormal distribution is the continuous distribution of a random variable whose natural logarithm is normally distributed. This is, if the random variable $X \sim Lognormal$, then the random variable $Y = ln(X) \sim Normal$.

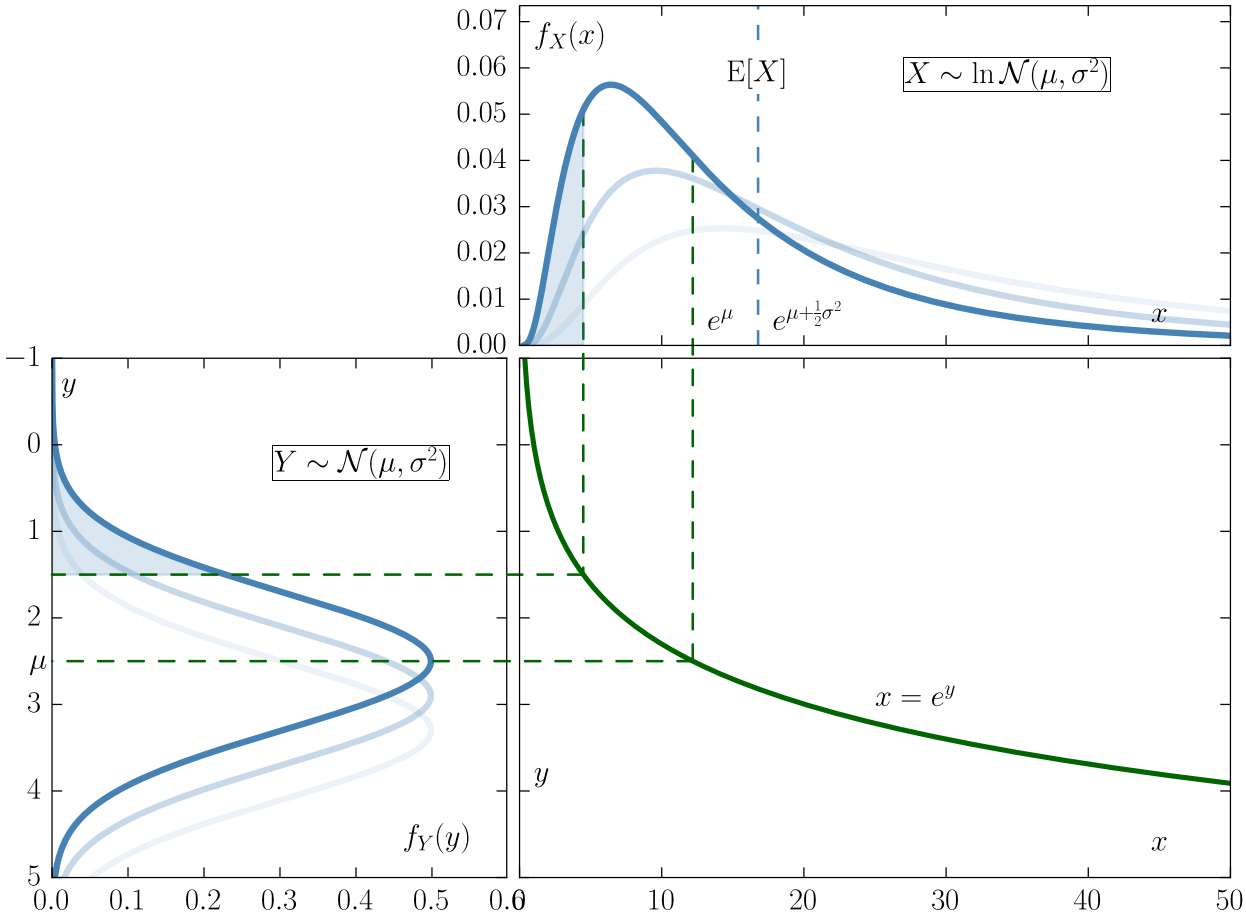



*Fig. 5.19* Relationship between Lognormal and Normal distribution[1].

It is widely used in engineering, since it models variables which only take positive real values. The PDF of the Lognormal distribution is given by

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{(ln(x)-\mu)^2}{2\sigma^2}\right)}$$

where $\mu$ and $\sigma$ are the mean and standard deviation of the variable's natural logarithm. Note that they are not the mean and standard deviation of the variable $X$ itself.

Let's see how these parameters influence the shape of the distribution. In the figure below, two sets of three PDFs are displayed. In the left pannel, the displayed distributions present the same $\sigma$ and different values of $\mu$ =0.5, 1 and 2. The lower the value of $\mu$, the more peaked is the distribution and the mode of the distribution moves towards 0. In the right pannel, all the distributions present the same value of $\mu$ and different values of $\sigma$ =0.5, 1 and 2. When $\sigma$ grows ($\sigma > 1$), the peak of the distribution moves towards 0 and increases in height. Similarly, when $\sigma$ is reduced below 1, the distribution becomes more peaked; however, it also moves towards positive values. Therefore, the influence of this parameter is different if $\sigma$ is above or below 1.

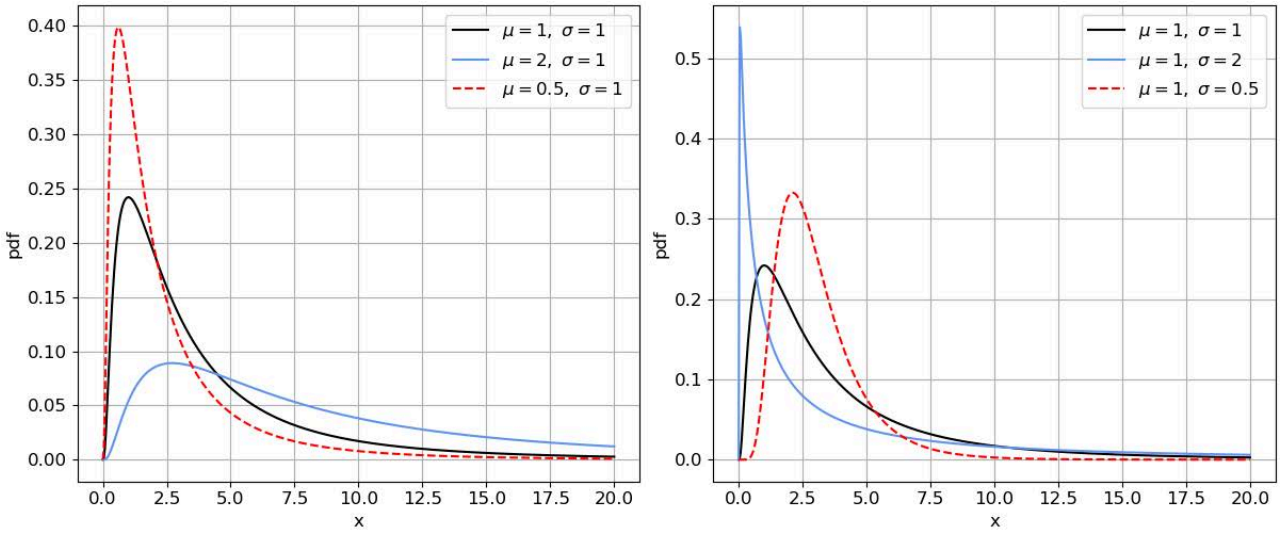



*Fig. 5.20* PDF of Lognormal distribution: (left) influence of parameter $\mu$, and (right) influence of parameter $\sigma$.

Integrating the PDF, the following expression of the CDF is derived

$$F(x) = \Phi\left(\frac{ln(x)-\mu}{\sigma}\right) = \frac{1}{2}\left[1 + erf\left(\frac{ln(x)-\mu}{\sigma\sqrt{2}}\right)\right]$$

where $\Phi$ is the cumulative distribution function of of the standard Normal distribution function ($N(0,1)$). The CDF of the Lognormal distribution is displayed in the figure below.

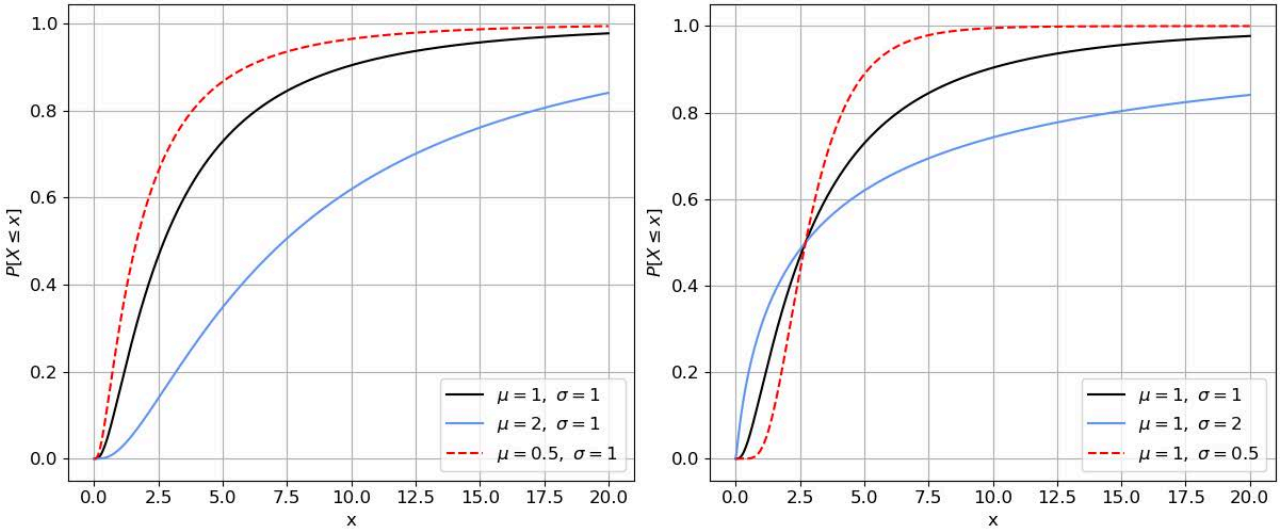



*Fig. 5.21* CDF of Lognormal distribution: (left) influence of parameter $\mu$, and (right) influence of parameter $\sigma$.

In the left pannel, the influence of the parameter $\mu$ is shown. The lower the $\mu$, the steeper becomes the CDF, reaching higher non-exceedance probabilities for increasing values of the random variable. In the right pannel, the influence of the parameter $\sigma$ is presented. Again, it can be seen that the influence of $\sigma$ depends whether it is above or below 1.

# Some properties

The mean of the Lognormal distribution can be computed as

$$E[X] = e^{\mu + \frac{\sigma^2}{2}}$$

The variance is given by

$$Var[X] = \left(e^{\sigma^2} - 1\right) e^{2\mu + \sigma^2}$$

Finally, note that Lognormal distribution is not symmetric and presents positive skewness. This is, it presents a tail towards positive values.

Also, the **Lognormal distiribution is bounded in 0**. This is, the random variable modelled with the Lognormal distribution cannot take negative values.

**Exercises**

During the design phase of a coastal structure, it is needed to assess if its height is enough to protect the sheltered area from overtopping events (sea water overpassing

the structure and reaching the lee side). The distribution of overtopping volumes is known to follow a Lognormal distribution. The engineer has already calculated the parameters of the Lognormal distribution ($\mu$=5.5 and $\sigma$=1.15) and has plotted the CDF for you.

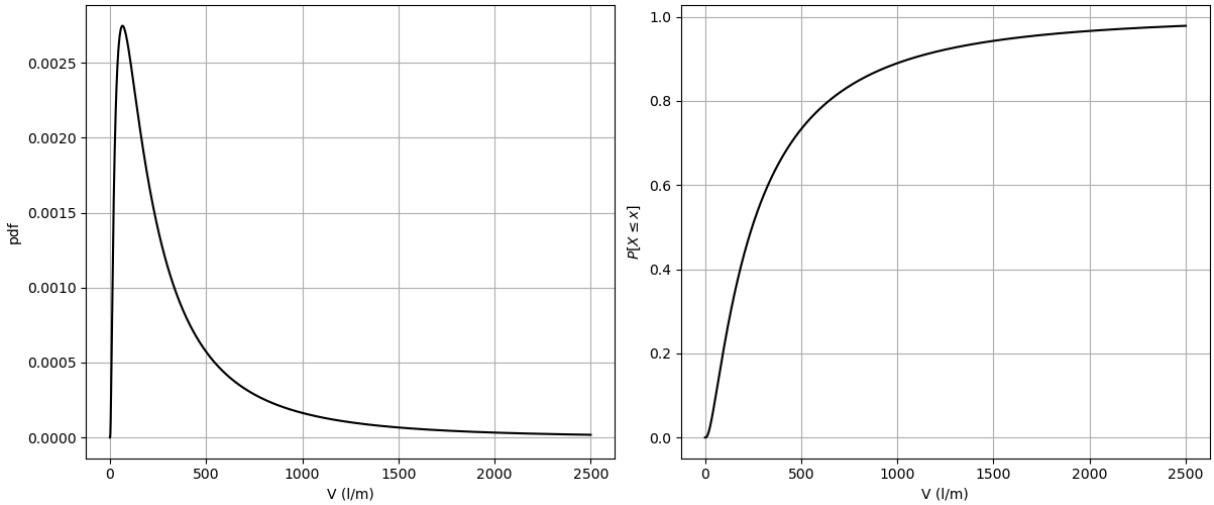



*Fig. 5.22* PDF and CDF of Lognormal distribution to describe overtopping volumes $V(l/m)$.

The maximum allowed overtopping volume is 500 l/m. What is the probability of exceeding that value? Round your answer to one decimal place.

Check

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

"Lognormal Distribution" by StijnDeVuyst is licensed under CC BY-SA 4.0. To view a copy of this license, visit https://creativecommons.org/licenses/by-sa/4.0/?ref=openverse.

# Summary of parametric distributions

Here a summary of the main equations for each of the presented distirbution functions is presented.

| Distribution | PDF | CDF |
|---|---|---|
| Normal/Gaussian | $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}$ | $F(x) = \frac{1}{2}\left(1 + \text{erf}\left(\frac{x-}{\sigma\sqrt{}}\right.\right.$ |
| Uniform | $f(x) = \begin{cases} \frac{1}{b-a} & \text{for } x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$ | $F(x) = \begin{cases} 0 & \text{for } x < \\ \frac{x-a}{b-a} & \text{for } x \in \end{cases}$ |
| Exponential | $f(x) = \lambda e^{-\lambda x}$ | $F(x) = 1 - e^{-\lambda x}$ |
| Gumbel | $f(x) = \frac{1}{\beta} e^{-(z+e^{-z})}$, where $z = \frac{x-\alpha}{\beta}$ | $F(x) = e^{-e^{-z}}$ |
| Lognormal | $f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{(ln(x)-\mu)^2}{2\sigma^2}\right)}$ | $F(x) = \Phi\left(\frac{ln(x)-\mu}{\sigma}\right)$ |

**MUDE exam information**

You do not need to know the equations of the distribution functions by heart. You just need to know how the distribution looks (PDF/CDF), how it responds to changes in the parameters and some basic properties (symmetry or bounds).

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Fitting a Distribution

In the previous sections, we introduced parametric distributions as mathematical models for the empirical distribution functions of observations. Also, some of the most widely used parametric distributions were presented, such as Exponential or Normal distribution. Those parametric distributions had some parameters which required to be fitted to the observations. For instance, Exponential distribution has a rate parameter ($\lambda$). In this section, the most commonly used methods to fit the parametric distribution functions are first presented.

- Method of moments
- Maximum loglikelihood estimator (MLE)

Note that it assumes that we know the parametric distribution we want to fit. For instance, fitting an Exponential distribution to a sample of observations. So, how do I choose between them? The choice of the appropriate distribution function needs to be based first on the physics of the random variable we are studying. Once we have accounted for those physical characteristics, we can make use of goodness of fit (GOF) techniques to support our decision. In the subsequent sections, some commonly used GOF techniques in the statistics field are also presented.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Method of moments

## Contents


- Let's see it with an example


The method of moments consists of equating the moments[1] in the observations to those of the distribution we want to fit. This is, there is a relationship between the moments and the parameters of the distribution. Then, if we equate the moments of the distribution to the moments of the observations, we can obtain the values of the parameters of the distribution. Actually, you have already done it in the exercises of the Exponential distribution!

## Let's see it with an example

An engineer is studying the intensity of earthquakes in Rome (Italy). To this end, the engineer is using *Catalogo dei terremoti italiani dall'anno 1000 al 1980* (the Catalog of Italian earthquakes from year 1000 to 1980) edited by D. Postpischl in 1985. This catalog reports the intensity of earthquakes in terms of the Mercalli-Canconi-Sieber (MCS) index. Due to the uncertainties associated to this natural phenomenom, the engineer considers them as a random one and wants to fit a Gumbel distribution to the observations found in the catalog using the method of moments. The data found in the catalog is shown in the table below [2].

*Table 5.1* MSC intensity of the recorded earthquakes in the city of Rome.

| **MSC intensity** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|
| Number | 113 | 132 | 56 | 22 | 4 | 2 |

Remember that the CDF of the Gumbel distribution is given by

$$F(x) = e^{-e^{-\frac{x-\mu}{\beta}}}$$

Therefore, the value of $\mu$ and $\beta$ needs to be determined based on the observations to fit the distribution.

The first thing the engineer needs to do is to calculate the moments of the observations in the Table. The empirical mean ($\overline{X}$) and empirical variance ($\sigma^2$) are calculated as:

$$\overline{X} = \frac{2 \cdot 113 + 3 \cdot 132 + 4 \cdot 56 + 5 \cdot 22 + 6 \cdot 4 + 7 \cdot 2}{113 + 132 + 56 + 22 + 4 + 2} \approx 3.02$$

$$\sigma^2 = \frac{\sum f x^2}{\sum f} - \left( \frac{\sum f x}{\sum f} \right)^2 \approx 0.99$$

where $x$ is the value of the intensity and $f$ is the frequency of the value $x$.

Based on the properties of the Gumbel distribution, we know

$$E[X] = \mu + \gamma\beta$$

$$Var[X] = \frac{\pi^2}{6}\beta^2$$

where $\gamma \approx 0.577$ is the Euler-Mascheroni constant.

Therefore, we can equate the expectation and variance of the distribution ($E[X]$ and $Var[X]$) to the calculated moments and obtain the value of the parameters as

$$3.02 = \mu + 0.577\beta$$

$$0.99 = \frac{\pi^2}{6}\beta^2$$

Thus, $\mu \approx 2.57$ and $\beta \approx 0.77$.

**Exercises**

As part of the quality control of the construction of a building, lab tests are performed to determine the compressive strengths of concrete. The following values in $N/mm^2$ are obtained: 60.5, 59.8, 53.4, 56.9 and 61.9.

The engineer responsible for quality assumes that the compressive strength follows a uniform distribution, whose CDF is given by

$$F(x) = 0 \qquad for\ x < a$$

$$F(x) = \frac{x-a}{b-a} \qquad for\ a \leq x \leq b$$

$$F(x) = 1 \qquad for\ x > b$$

Determine the value of \(a\) and \(b\) based on the observations using the method of moments. Round your answer to two decimal places and use \(a<b\) based on our definition of the uniform distribution.

\(a=\)

\(b=\)

Check

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

We denote as moments in statistics to the quantitative properties to characterize a distribution. The four commonly used moments are the mean, the variance, the skewness and the kurtosis.

Data extracted from Kottegoda and Rosso (2008).

# Maximum Likelihood Estimation

## Contents


- Let’s see it with an example!
- A video with further examples


Maximum Likelihood Estimation (MLE) is a statistical method used to estimate the parameters of a probability distribution based on observed data $\mathbf{x} = x_1, x_2, \ldots, x_n$. Evaluating the joint density of the data given a parametric family $\theta$ gives the likelihood function, which is a function that measures how well the observed data fits the probability distribution with the given parameters:

$$L(\theta \mid \mathbf{x}) = f(\mathbf{x} \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta)$$

The goal of MLE is to find the values of the parameters that maximize the likelihood function. The maximum likelihood estimate $\hat{\theta}$ is the set of parameters for which the observed data is the most probable with the assumed probability distribution:

$$\hat{\theta} = \arg\max_{\theta} L(\theta \mid \mathbf{x})$$

In practice, it is more common to use the log-likelihood, which is the natural logarithm of the likelihood function:

$$\ln(L(\theta \mid \mathbf{x})) = \ln(\prod_{i=1}^{n} f(x_i \mid \theta))$$

To find the maximum likelihood estimate $\hat{\theta}$, the following steps should be taken:

1. Define the likelihood function $L(\theta \mid \mathbf{x})$.

2. Take natural logarithm of the likelihood function to get the log-likelihood function $\ln(L(\theta \mid \mathbf{x}))$.
3. Differentiate the log-likelihood function and set it to zero: $\frac{\partial L(\theta \mid \mathbf{x})}{\partial \theta} = 0$.
4. Solve the equation for $\hat{\theta}$.

Note that the maximum can also occur at the boundary of the domain.

# Let's see it with an example!

The following dataset describes the time elapsed between consecutive arrivals of passengers at a bus stop (in minutes):

$$\mathbf{x} = [1.2, 0.5, 3.7, 2.3, 0.9, 1.5, 2.1, 3.0, 1.8, 2.5]$$

Assume that the observations can be described by the exponential distribution, for which the PDF is given by

$$f(x, \lambda) = \lambda e^{-\lambda x} \$ for \$ x > 0$$

with $\lambda$ being the parameter that has to be estimated. There dataset consists of $n$ observations. Find the maximum likelihood estimate $\hat{\lambda}$ following the four steps defined above:

1. Define the likelihood function $L(\lambda \mid \mathbf{x})$.
2. Take natural logarithm of the likelihood function to get the log-likelihood function $\ln(L(\lambda \mid \mathbf{x}))$.
3. Differentiate the log-likelihood function and set it to zero: $\frac{\partial L(\lambda \mid \mathbf{x})}{\partial \lambda} = 0$.
4. Solve the equation for $\hat{\lambda}$.

**1. Define the likelihood function $L(\lambda \mid \mathbf{x})$.**

According to the definition, the likelihood function is given by:

$$L(\lambda \mid \mathbf{x}) = \prod_{i=1}^{n} f(x_i \mid \lambda)$$

For the exponential distribution, $f(x_i \mid \lambda) = \lambda e^{-\lambda x}$.

Therefore, the likelihood function for the exponential distribution is given by:

$$L(\lambda \mid \mathbf{x}) = \prod_{i=1}^{n} \lambda e^{-\lambda x_i} = \lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}$$

**2. Take natural logarithm of the likelihood function to get the log-likelihood function $\ln(L(\lambda \mid \mathbf{x}))$.**

Taking the natural logarithm of the likelihood function to obtain the log-likelihood function:

$$\ln(L(\theta \mid \mathbf{x})) = \ln(\lambda^n e^{-\lambda \sum_{i=1}^{n} x_i}) = n \ln(\lambda) - \lambda \sum_{i=1}^{n} x_i$$

**3. Differentiate the log-likelihood function and set it to zero: $\frac{\partial L(\lambda \mid \mathbf{x})}{\partial \lambda} = 0$**

Taking the derivative of the log-likelihood function with respect to $\lambda$ and setting it to zero:

$$c\frac{\partial \ln(L(\lambda \mid \mathbf{x}))}{\partial \lambda} = \frac{\partial (n \ln(\lambda) - \lambda \sum_{i=1}^{n} x_i)}{\partial \lambda} = \frac{n}{\hat{\lambda}} - \sum_{i=1}^{n} x_i = 0$$

**4. Solve the equation for $\hat{\lambda}$.**

We have the following equation to solve:

$\frac{n}{\hat{\lambda}} - \sum_{i=1}^{n} x_i = 0$

which results in the maximum likelihood estimate:

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i} = \frac{1}{\bar{x}}$$

where $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$ is the sample mean of the obsrved data.

This means that the maximum likelihood estimate of the exponential distribution is the inverse of the sample mean. Therefore:

$$\hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i} = \frac{10}{19.5} = 0.513$$

The best fitting parameter of the Exponential distribution for the given data according to MLE is $\lambda = 0.513$.

# A video with further examples

In the following video, you have an explanation on MLE together with 3 examples with different continuous distribution functions.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Goodness of Fit

## Contents


- Goodness of Fit
- Let's practice


In the previous sections you have studied the different mathematical models (continuous distribution functions) that we can use to model the univariate uncertainty of a random variable and how to fit them based on observations. Also, you have been introduced to some methods to fit those models. **But how do I choose between different models?**

The choice of the appropriate distribution function needs to be based first on the **physics of the random variable** we are studying. For instance, if I am studying the concentration of a gas in the atmosphere, negative values do not have a physical meaning, so the selected distribution function should not provide with those estimations.

Once we have accounted for the physical characteristics of the random variable, we can make use of **goodness of fit (GOF) techniques** to support our decision. This is, GOF techniques are not a ground truth, but an objective way of comparing models. Different techniques may lead to different judgments and it is you as expert who has to balance those outputs and select the best model to your judgment. Thus, it is recommended to use more than one GOF technique in the decision-making process. In the subsequent sections, some commonly used GOF techniques in the statistics field are presented.

In order to illustrate these techniques, the following toy example will be used. The set of observations is represented in the plots below by its pdf and cdf. A Gaussian ( $N(5.17, 5.76)$) and an Exponential distributions ($Expon(-5.25, 10.42)$) are fitted to the data. GOF techniques will be applied to determine which one of the two models fits the data best.

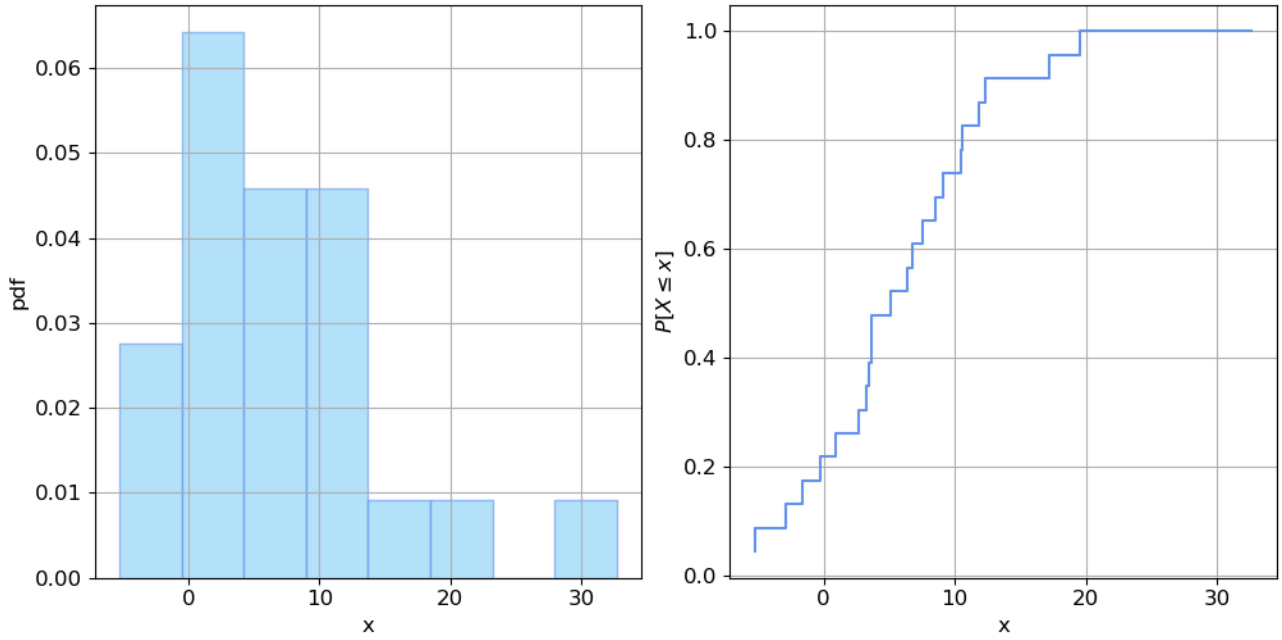



*Fig. 5.23* Data overview.

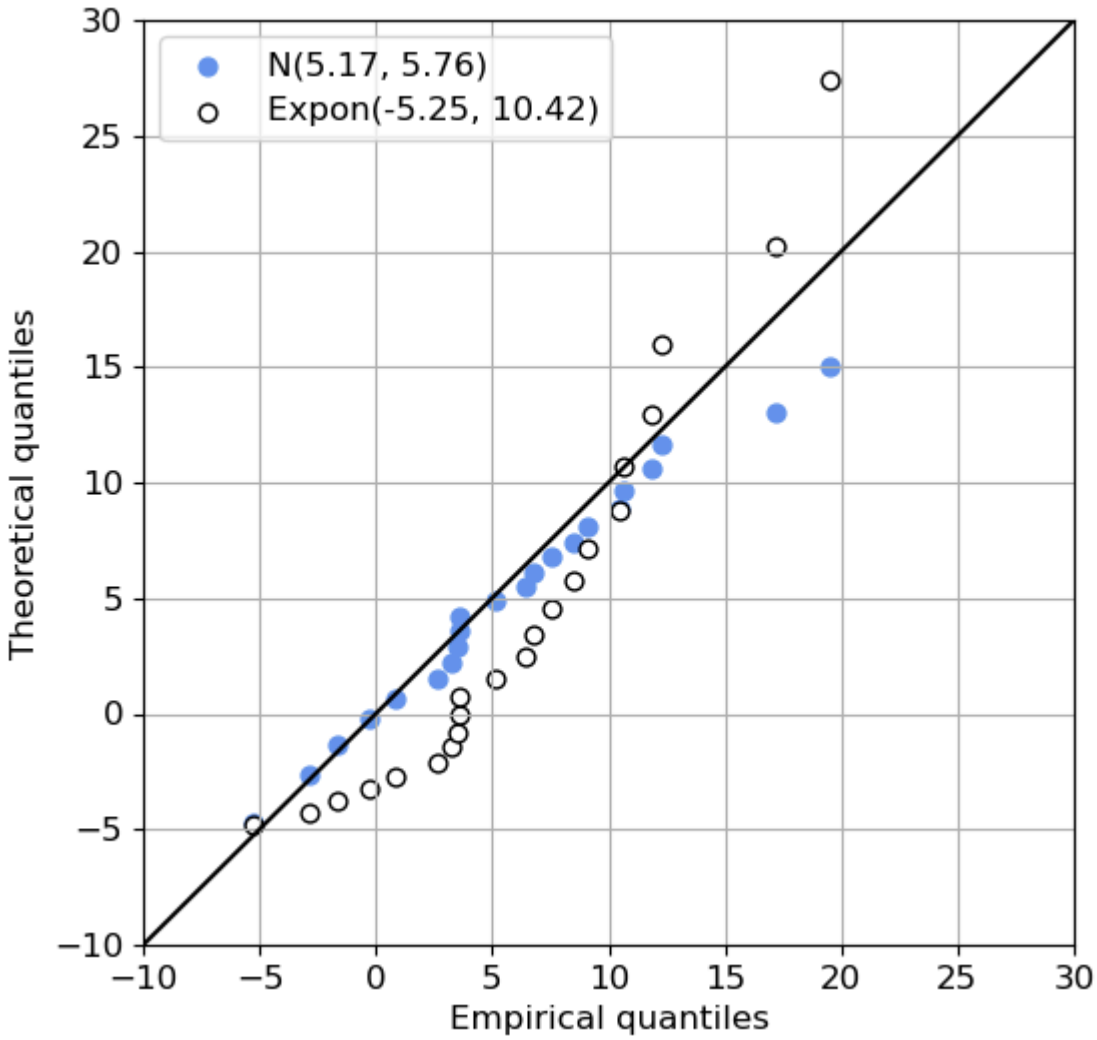
# Graphical methods

GOF graphical methods are useful tools to have a first intuition of how different models are performing and confirm the results of other quantitative analysis. Here, you are introduce to three techniques: (1) QQ-plot, (2) Log-scale, and (3) Probability plot.

## QQ-plot

This technique is as simple as comparing the observations used to fit the model with the predictions of the model. Typically, the observations are represented in the x-axis and the predictions in the y-axis. Therefore, the perfect fit would be represented by the $45^{\circ}$-line.

Let's see it applied to the example data. Note that the term *"quantile"* is used in statistics to denote the values of the random variable.




System Message: WARNING/2 (/home/runner/work/book/book/book/probability/GOF.md, line 30)

Cannot scale image! Could not get size from "https://files.mude.citg.tudelft.nl/QQplot.png": Can only read local images.

*Fig. 5.24* QQ-plot.

In the QQ-plot, it is shown how the predictions given by the Gaussian distribution (in blue) closely follow the $45^{\circ}$-line. Those provided by the Exponential distribution are way further, detaching significantly from the $45^{\circ}$-line in the upper tail. Based on this graphical technique, it is possible to conclude that Normal distribution seems to be a better model for the data.

**Let's code it!**

Pseudo code is presented to illustrate the procedure to build a QQ-plot.

```
read observations

#calculate the empirical cdf
p_emp, q_emp = empirical CDF of observations

#define the parameters of the Gaussian distribution
mean_gaussian = 5.17
sd_gaussian = 5.76

#compute the values of the random variable predicted by the Normal distribution
q_gaussian = CDF of Normal distribution evaluated in p_emp with parameters mean_gaussian a

#define the parameters of the Exponential distribution
loc_expon = -5.25
scale_expon = 10.42

#compute the values of the random variable predicted by the fitted distribution
q_exponential = CDF of Exponential distribution evaluated in p_emp with parameters loc_ex

scatterplot of q_emp versus q_gaussian
scatterplot of q_emp versys q_expon
```

# Log-scale

As previously introduced, the tails of the distributions are key to allow the inference of values which have not been observed yet. Therefore, it is important to check whether the distribution used to model the observations is performing properly in that region. A simple trick to do so is to use a logarithmic scale (log-scale) to represent the exceedance probability plot. That way, we “zoom in” on those points in the tail instead of focusing on the bulk of the data. In the figure below, the representation of the cdf in regular and log-scale is shown.

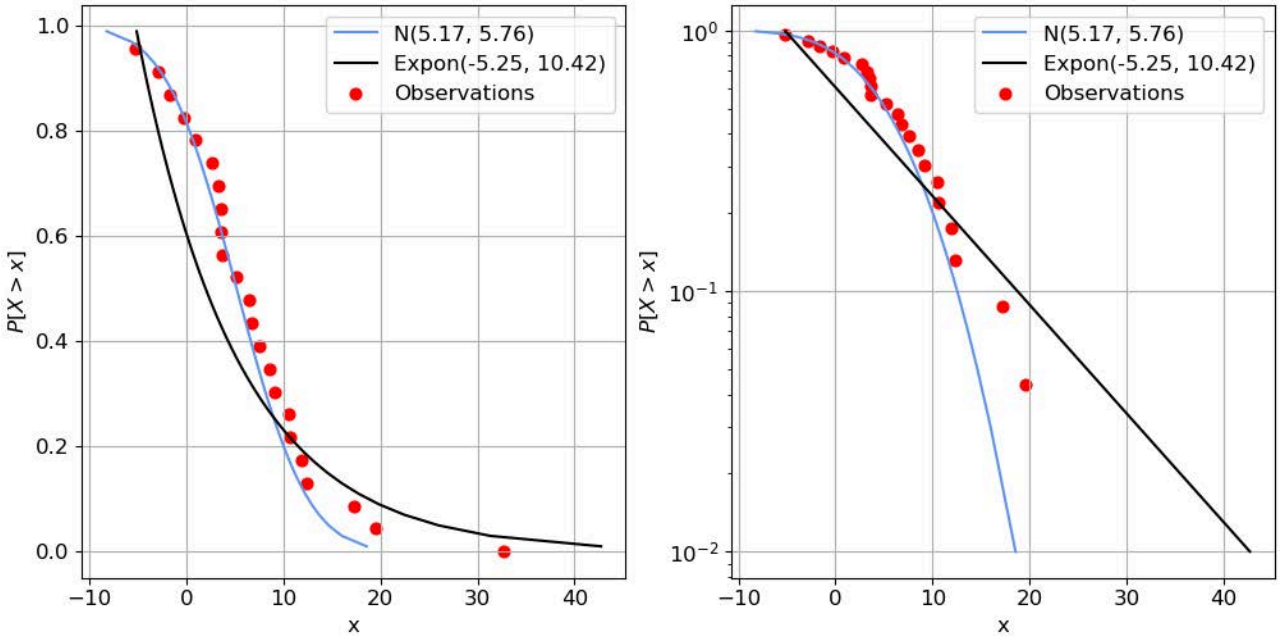


*Fig. 5.25* Exceedance probability plot represented both in regular and logarithmic scale.

Analyzing the figure on the left side, it can be seen that the observations follow better the Normal distribution. However, it is not clear which one of the two distributions is performing better in the tail. By analyzing the plot on the right side, it is possible to answer that question. Again, it is observed that the data points follow better the Gaussian distribution. However, observations in the tail are not that well represented by the Gaussian distribution, being even closer to the Exponential distribution. Thus, since none of the considered distributions performs properly in the tail, it may be needed to consider another distribution to model the asymmetry of the data, such as the Gumbel or Lognormal distributions.

# Probability plot or probability paper

This graphical technique consists on adapting the axis of the plot of the cdf accounting for the parametric distribution fitted so it is presented as a line. This is, the cdf of any Gaussian distribution will be plotted as a line in the Normal probability plot. Thus, in the x-axis a function of the values of the random variable is presented, while in the y-axis a function of the non-exceedance probabilities is shown.

Let's see it with the example of the Exponential distribution. Its cdf is given by

$$F(x) = 1 - exp(-\lambda[x - \mu])$$

where $\lambda$ is the scale parameter and $\mu$ is the location parameter. A transformation is performed on the cdf so a linear relationship is established between the value of the random variable X and the non-exceedance probabilities. In the case of the Exponential distribution, it is just a matter of calculating logarithms to both sides of the equation as

$$ln[1 - F(x)] = -\lambda[x - \mu]$$

In this manner, there is a linear relationship between $ln[1 - F(x)]$ and $x$. Note that in the case of the Exponential distribution, the probability plot is the same as the log-scale! Therefore, the Exponential distribution was shown as a straight line in the previous plot, while the Gaussian distribution was not.

# Formal hypothesis test: Kolmogorov-

# Smirnov test

Kolmogorov-Smirnov (KS) test is one of the most popular nonparametric formal hypothesis tests in statistics. It can be used with two purposes: (1) to compare a sample with a reference parametric distribution, and (2) to compare two samples. Here, the first option is considered since it is the one used for GOF purposes. Thus, this test aims to determine how likely is that a sample was drawn from the reference parametric distribution.

This test is based on the KS statistic, which is (roughly) the maximum distance between the empirical cumulative distribution and the parametric distribution fitted to those observations. This statistic is mathematically defined as

$$D_n = sup_x|\hat{F}(x) - F(x)|$$

where $D_n$ is the KS statistic, $sup_x$ is the supremum of the set distances (intuitively, the largest absolute difference between the two distribution functions across all the values of the random variable $X$), $\hat{F}(x)$ is the empirical cumulative distribution and $F(x)$ the fitted parametric cumulative distribution.

Once $D_n$ is computed, a formal hypothesis test is performed. The null hypothesis corresponds to $\hat{F}$ having the same distribution as $F$. In mathematical terms:

$$H_0 : \hat{F} \sim F$$

The distribution of $D_n$ has been already calculated and included in different statistic packages, since it depends on the considered parametric distribution. These distributions can be used to calculate the probability of the null hypothesis being true (called $p - value$). A significance level needs to be selected (typically, $\alpha = 0.05$) as a threshold to determine whether the null hypothesis is rejected or accepted. This is, if the probability of $H_0$ being true ($p - value$) is below $\alpha$, $H_0$ is rejected, so the empirical cumulative distribution is not coming from the fitted parametric cumulative distribution.

Let's see it in an example. In the figure below, both the empirical distribution (step function) and the fitted normal distribution are shown. The maximum distance between both distributions is also presented in red.

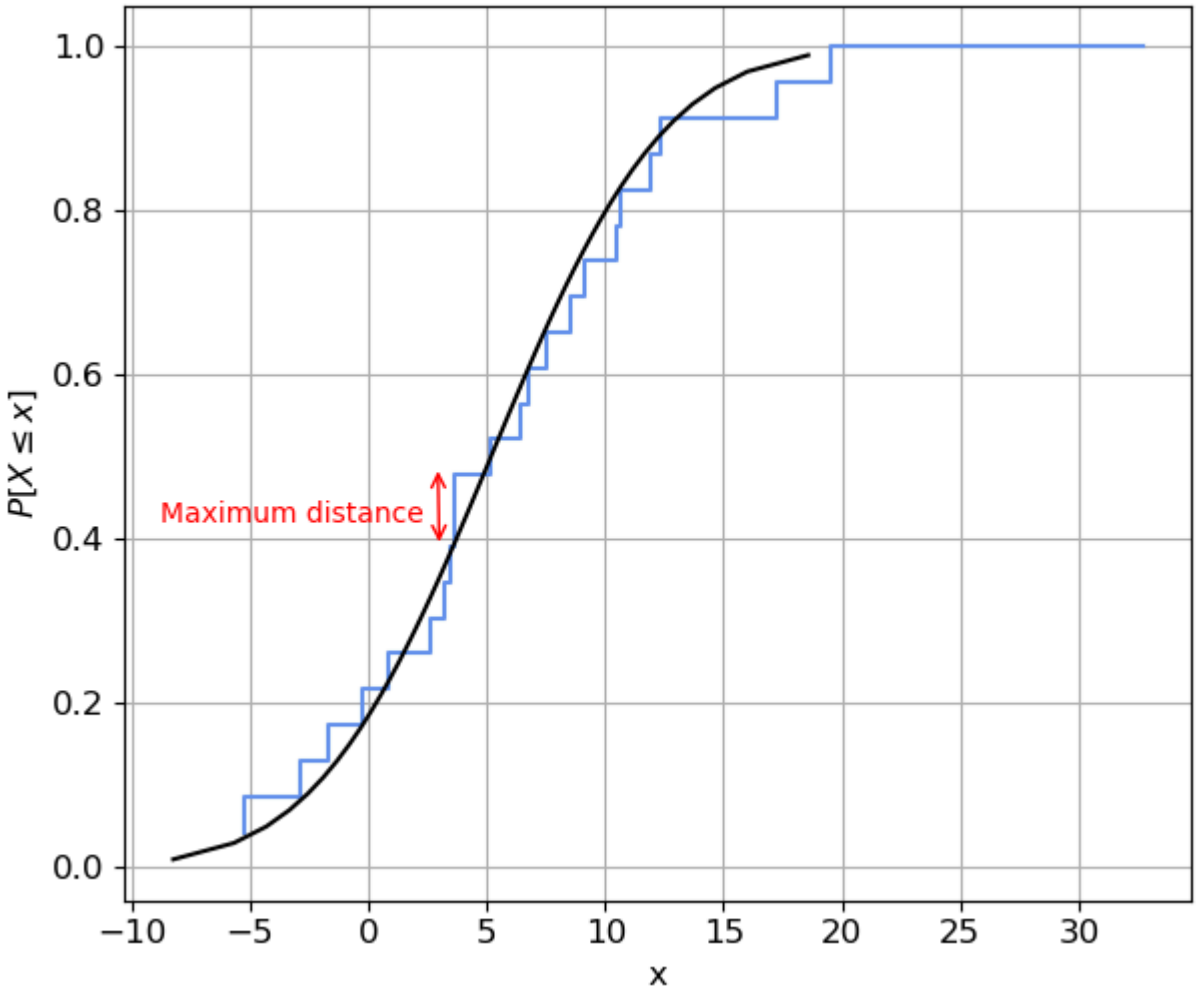



*Fig. 5.26* Maximum distance between the empirical and fitted normal distribution ($D_n$).

If we compute the KS statistic using the distribution already implemented in software (Scipy package, in this case), $D_n = 0.12$ is obtained which (roughly) corresponds to what is shown in the previous plot.

After that, the $p - value$ is also computed, obtaining $p - value = 0.93$. This means that the probability of the null hypothesis ($H_0 : \hat{F} \sim F$, the sample comes from the parametric distribution) being true is 0.93. Thus, considering a significance level $\alpha = 0.05$, $pvalue = 0.93 > \alpha = 0.05$, so I cannot reject the null hypothesis.
An engineer is characterizing the axle loads transmited by the traffic on a bridge. Based on the observations, a Normal and a Gumbel distributions are fitted, as shown in the figure below.

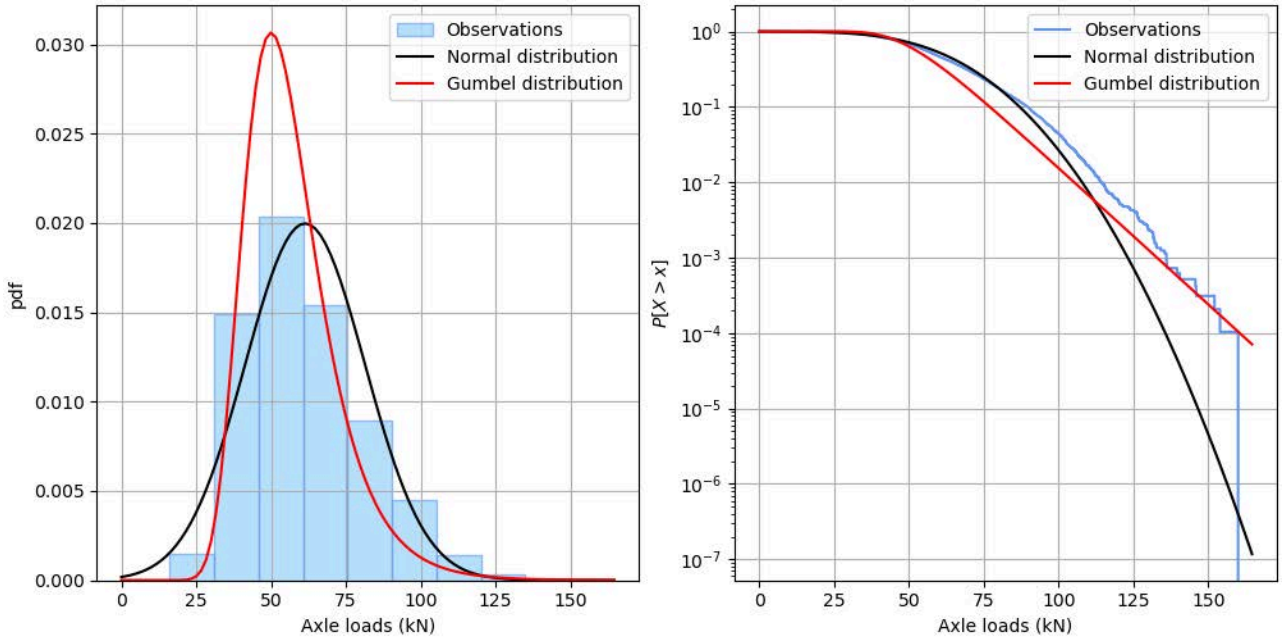



*Fig. 5.27* Comparison between the observations of axel loads and the fitted Normal and Gumbel distributions: (a) PDF, and (b) Exceedance curve.

The engineer needs to select a parametric distributions since he/she needs to infer extreme loads that have not been observed yet. In order to decide which distribution to apply for further analysis, the engineer has plotted the Exceedance Curve (1-CDF) in log-scale (previous figure) and has performed the Kolmogorov-Smirnov test. The results are:

- Normal distribution: $p - value \approx 0.07$
- Gumbel distribution: $p - value \approx 0$

Which distribution should be chosen by the engineer?

**Answer**

According to the Kolmogorov-Smirnov test, it is not possible to reject that the observations are coming from a Normal distribution, while it is possible to do so for the Gumbel distribution. Therefore, if we would only trust on the Kolmogorov-Smirnov test, Normal distribution would be chosen.

However, the plot in log-scale is also available. There, it is shown how the Gumbel distribution fits way better the tail of the empirical distribution. Since the goal of the engineer is to infer events that have not been observed yet (extrapolate), the tail is extremely important. Consequently, Gumbel distribution would be preferred in this context to model the axel loads observations.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Parameterization of continuous distributions

## Contents


- Definition of location, scale and shape
- Standardized form
- From convention to loc-scale-shape parameterization with an example
- Let's check it using code


In the previous sections, you have studied different parametric distributions that can be applied to model the univariate uncertainty in our data. Those distributions were characterized by a set of parameters (e.g.: $\lambda$ for Exponential distribution). Those parameters can be fitted to model real-world data as accurately as possible and, thus, use the distribution for predicting future events. Along the sections devoted to present a selection of distribution functions, the equations for the PDF and CDF as you can usually find them in text books were presented. However, they are just equations! That means that we can play with them and parameterize the distribution the way that fits best to our purposes.

In this section the parameterization loc-scale-shape will be addressed in the context of `scipy` Python package. This parameterization is very convenient due to the consistency it provides (all distributions with the same parameters), the ease of the interpretation of those parameters and the advantages of their implementation in computer code. That is why `scipy` package (between others) uses this parameterization for continuous distribution functions.

## Definition of location, scale and shape

The location ($\mu$) parameter shifts the distribution along the x-axis without changing its shape. The scale parameter ($\beta$) determines the width of the distribution. Finally, the shape parameter ($\xi$) is any extra parameter (if any) in the distribution function which is not $\mu$ or $\beta$ and describes the form of the distribution. Let's see it better with a couple of examples!

You have already been introduced to the (right-tailed) Gumbel distribution, whose PDF is given by

$$f(x) = \frac{1}{\beta} e^{-\left(\frac{x-\mu}{\beta} + e^{-\left(\frac{x-\mu}{\beta}\right)}\right)}$$

where $\mu$ is the location parameter and $\beta > 0$ is the scale parameter. You can see that the conventional form of the Gumbel distribution is actually the loc-shape-scale parameterization! We will see more about it later, so let's focus now on the meaning of those parameters. In the figure below, you can see the PDF of three Gumbel distributions. The blue and the black ones have the same scale parameter but different location. Thus, the distribution moves along x-axis. Note that the location parameter defined the mode of the distribution. The black and red distributions present the same location parameter but different scale. Thus, the mode is located at $x = 0$ in both distributions (since $\mu$=0) but the red distribution (with a higher scale) spreads along a wider range of values.

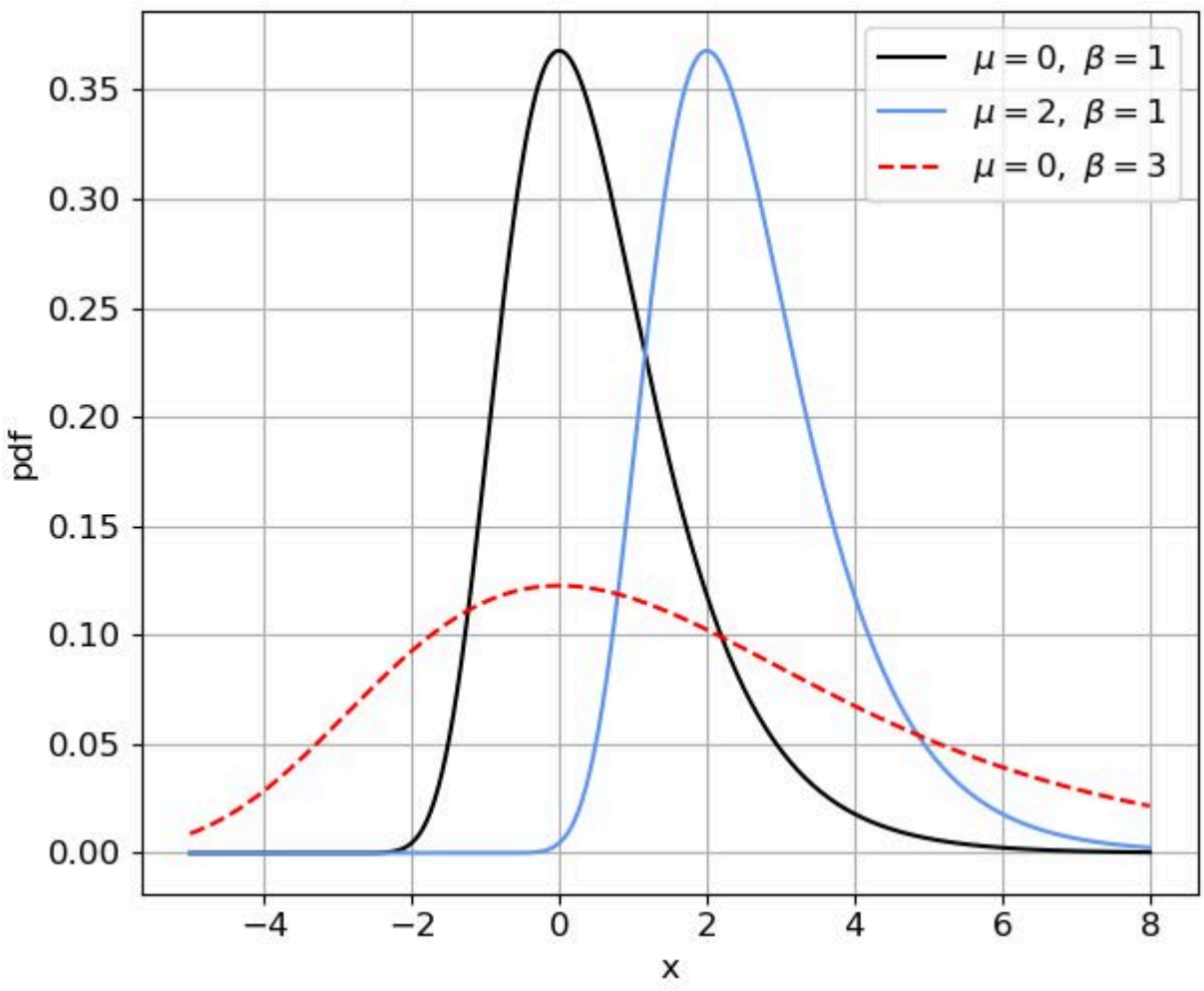



*Fig. 5.28* PDF of Gumbel distribution function.

Since the Gumbel distribution does not present a shape parameter, let's see it with a different distribution. The generalized form of Gumbel distribution is the Generalized Extreme Value (GEV) distribution. Similarly to Gumbel distribution (and as you can guess from its name), it is applied to model the probabilistic behavior of extreme observations (you will learn more about this later in the course). The PDF of the GEV is given as

$$g(x) = exp- \left[1 + \xi \frac{x-\mu}{\sigma}\right]^{\frac{-1}{\xi}} \qquad \left(1 + \xi \frac{x-\mu}{\sigma}\right) > 0$$

where $-\infty < \mu < \infty$ is the location parameter, $\sigma > 0$ is the scale parameter, and $-\infty < \xi < \infty$ is the shape parameter. Let's see the effect of the shape parameter!

In the figure below, you can see the PDF (left pannel) and the CDF (right pannel) of three GEV distributions. They have the same location ($\mu$=0) and scale ($\sigma$=1) but different shape parameters ($\xi$ = -1, 0 and 1). You can see that the form of the three distributions is totally different! Don't worry too much about this specific case, you just need to remember that the shape parameter controls the form of the distribution.

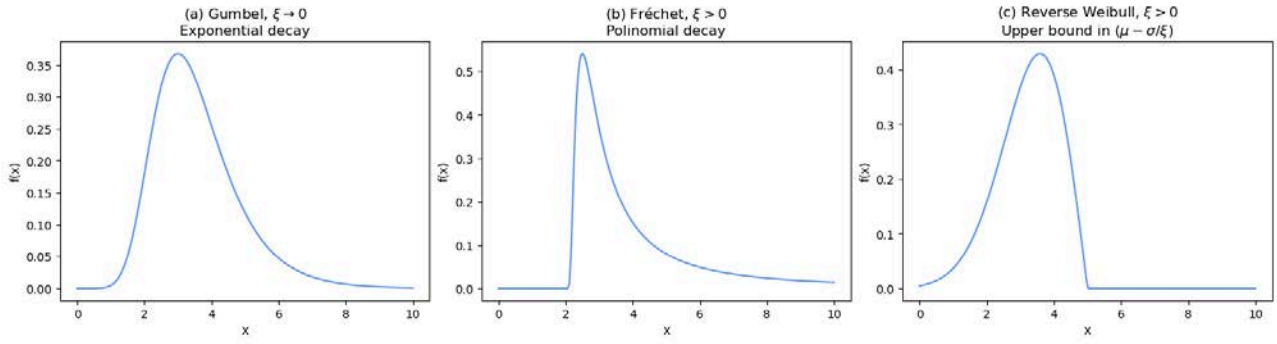



*Fig. 5.29* Effect of shape parameter on GEV distribution: (left) PDF, and (right) CDF.

# Standardized form

In order to address the loc-scale-shape parameterization, first the standardized form of a distribution needs to be defined. The standardized form is that where there are no extra parameters but shape parameters (if any); the density function is written purely as a function of the random variable $x$, without any scaling or shifting.

For the right-tailed Gumbel distribution, the standardized form of the PDF would be given by

$$f(x) = e^{-(x+e^{-x})}$$

The standardized Gumbel distribution can be useful when describing a certain dataset, however, it could be quite limiting. Think of the fact that not all random variables have these characteristics of mean and variance. Thus, we can shift and scale the standard distribution to give it the characteristics that we want.

# From convention to loc-scale-shape parameterization with an example

As mentioned, we need to be able to go from the parameters of the conventional form of the distribution to loc-shape-scale in order to use computer packages. Here, we will focus on `scipy` and will address the example of the Lognormal distribution since it is the one which creates the most confussion. You can find the documentation about the Lognormal distribution in `scipy` here.

If you follow the link, you will find the following description of the implemented Lognormal distribution:

The probability density function for `lognorm` is:

$$f(x, s) = \frac{1}{sx\sqrt{2\pi}} \exp\left(-\frac{\log^2(x)}{2s^2}\right)$$

for $x > 0, s > 0$.

`lognorm` takes `s` as a shape parameter for $s$.

The probability density above is defined in the "standardized" form. To shift and/or scale the distribution use the `loc` and `scale` parameters. Specifically, `lognorm.pdf(x, s, loc, scale)` is identically equivalent to `lognorm.pdf(y, s) / scale` with `y = (x - loc) / scale`. Note that shifting the location of a distribution does not make it a "noncentral" distribution; noncentral generalizations of some distributions are available in separate classes.

*Fig. 5.30* Documentation on the Lognormal distribution in `scipy`.

First, the PDF in standardized form is presented with a shape parameter $s$. Let's compare this PDF with the conventional one, which is given by

$$f(x) = \frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right)}$$

where $\mu$ and $\sigma$ are the mean and standard deviation of the variable's natural logarithm. If we compare both parameterizations, it is clear that the shape parameter $s$ corresponds with $\sigma$. That was an easy one!

Let's focus now on the loc and scale parameters. If we go back to the documentation in the figure, it states '… lognorm.pdf(x, s, loc, scale) is identically equivalent to lognorm.pdf(y, s) / scale with y = (x - loc) / scale.' Let's apply it and see if we can find it out.

If we apply 'lognorm.pdf(y, s) / scale with y = (x - loc) / scale', we obtain:

$$f(y) = \frac{\frac{1}{y\sigma\sqrt{2\pi}} e^{\left(-\frac{(\ln(y))^2}{2\sigma^2}\right)}}{scale} = \frac{\frac{1}{\frac{(x-loc)}{scale}\sigma\sqrt{2\pi}} e^{\left(-\frac{\left(\ln\left(\frac{x-loc}{scale}\right)\right)^2}{2\sigma^2}\right)}}{scale} = \frac{}{(x -}$$

If we compare this result with the standardized form of the Lognormal, the parameter loc does not seem to be there. Let's make loc = 0 and equate it to the standarized form.

$$\frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{\left(\ln\left(\frac{x}{scale}\right)\right)^2}{2\sigma^2}\right)} = \frac{1}{x\sigma\sqrt{2\pi}} e^{\left(-\frac{(\ln(x)-\mu)^2}{2\sigma^2}\right)}$$

which leads to

$$\ln\left(\frac{x}{scale}\right) = ln(x) - \mu$$

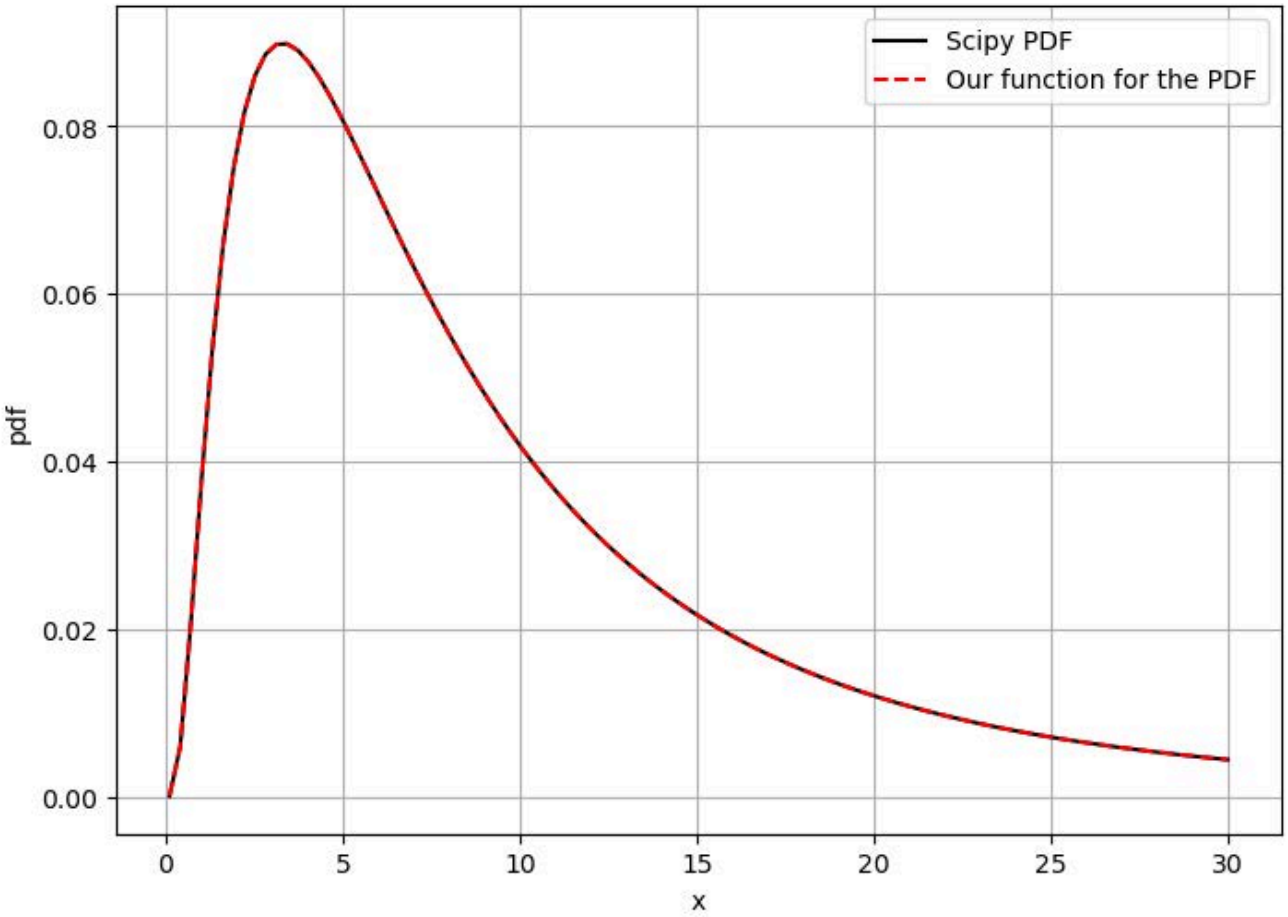
and thus, $\ln(scale) = \mu$. Therefore, in `scipy`, we would need to set loc = 0, $scale = e^{\mu}$ and $s = \sigma$ to use the Lognormal distribution.

# Let's check it using code

Here, we will define the PDF of the Lognormal distribution using a self-made function and we will compare the results with `scipy` to check whether we reached to the appropriate

solution.

```python
#Import the packages
import numpy as np
from scipy.stats import lognorm
import matplotlib.pyplot as plt

#Create a function for the PDF of the Lognormal distribution
def lognormal_PDF(x, mu, sigma):
    density = 1/(x * sigma * (2*np.pi)**0.5)*np.exp(-(np.log(x)-mu)**2/(2*sigma**2))
    return density

#Define parameters
mu = 2
sigma = 0.9

# #Transform to loc, scale, shape
loc = 0
shape = sigma
scale = np.exp(mu)

#Define a vector and evaluate the PDF using both the self-made function and scipy
x = np.linspace(0.1, 30, 100)
density_scipy = lognorm.pdf(x, shape, loc, scale)
density_manual = lognormal_PDF(x, mu, sigma)

#Compare by plotting
fig, ax = plt.subplots(1,1, figsize=(7,5), layout='constrained')
ax.plot(x, density_scipy, 'k', label = 'Scipy PDF')
ax.plot(x, density_manual, '--r', label = 'Our function for the PDF')
ax.grid()
ax.legend()
ax.set_xlabel('x')
ax.set_ylabel('pdf');
```




> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Multivariate Distributions

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

Challenges in all branches of science and engineering involve working in situations where uncertainty plays a significant role: prior chapters of this book focused a lot on *error*, but often we must also deal with data scarce scenarios (often categorized as *epistemic* uncertainty) and natural phenomena with a significant *stochastic* nature (often categorized as *aleatoric* uncertainty). As seen in the previous chapter, univariate continuous distributions can assist us in modelling uncertainty associated with a specific variable in order to quantitatively account for uncertainty in general; the distribution helps inform the decision-making process or risk analysis. However, our problems of interest are typically complex and usually involve more than one variable: a *multivariate* situation.

Consider, for example, when assessing algae blooms in a water body (e.g., a freshwater lake), different variables must be considered, such as nutrients in the water (nitrogen and phosphorus) and their concentration, dissolved oxygen and temperature of the water body. Sometimes (although not frequently), these variables are not related to each other, so we can consider them *independent.* For example, one may assume (for simplicity) that the amount of nitrogen and phosphorous that reaches the lake is not be related to the water temperature of the lake. However, truly *independent* situations rarely occur in reality, and the variables are *dependent* on each other. For example, the concentration of nitrogen and phosphorous changes with time and the reaction rates are dependent on the temperature of the water; thus if you are interested in these quantities over time, temperature is certainly a *dependent variable* of interest.

Although dependent relationships can often be quantified using deterministic relationships (e.g., mechanistic or phenomenological models), probability distributions are also capable of capturing this behavior. This is where *multivariate probability distributions* are helpful, as they allow us to model the distribution of not only one variable but several at the same time, thus accounting for their dependence.

**Revisiting correlation**

Probabilistic dependence between random variables is usually quantified through correlation coefficients. Previous, you have already been introduced to the concept of correlation and Pearson correlation coefficient, which is given by

$$\rho(X_1, X_2) = \frac{Cov(X_1, X_2)}{\sigma_{X_1}\sigma_{X_2}}$$

where $X_1$ and $X_2$ are random variables, $Cov(X_1, X_2)$ is their covariance, and $\sigma_{X_1}$ and $\sigma_{X_2}$ are the standard deviations of $X_1$ and $X_2$. $-1 \leq \rho(X_1, X_2) \leq 1$, being $\rho(X_1, X_2) = -1$ a perfect negative linear correlation, and $\rho(X_1, X_2) = 1$ a perfect positive linear correlation. If $X_1$ and $X_2$ are independent, then $\rho(X_1, X_2) \to 0$[1]. This is, that having information about $X_1$ does not provide us with information about $X_2$. The interactive element below allows you to play around with the correlation value yourself. Observe how the distribution's *density* contours, or a scatter plot of *samples,* change when you adjust the correlation.

> **Attribution**
>
> This interactive figure is created by Max Ramgraber. Find out more here.

*Fig. 6.1* Interactively change the correlation coefficient to visualize the effect on density contours or samples of the bivariate Gaussian distribution.

### Overview of this Chapter

Our ultimate goal is to construct and validate a model to quantify probability for combinations of more than one random variable of interest (i.e., to quantify various types of uncertainty).

Specifically,

$$f_X(x) \quad \text{and} \quad F_X(x)$$

where $X$ is a vector of continuous random variables and $f$ and $F$ are the multivariate probability density function (PDF) and cumulative distribution functions (CDF), respectively. Often we will use *bivariate* situations (two random variables) to illustrate key concepts, for example:

$$f_{X_1,X_2}(x_1, x_2) \quad \text{and} \quad F_{X_1,X_2}(x_1, x_2)$$

This chapter begins with a refresher on some fundamental aspects of probability theory that are typically covered in BSc courses on the subject, for example, dependence/independence, probability of binary events and conditional probability. Using the *bivariate* case, we will build a foundation on which to apply the multivariate Gaussian distribution (introduced in earlier chapters), as well as introduce alternative *multivariate distributions*. The theoretical part of the chapter ends with a brief introduction to *copulas* as a straightforward approach for evaluating two random variables that are dependent *and* described by non-Gaussian marginal distributions (the previous chapter). The last section revisits functions of random variables and takes an applied perspective by putting the univariate and multivariate probability concepts in a simple case study (design of a flood protection system).

---

> That is an intuitive definition of independence. For a more formal definition of independence, visit the next page of the chapter.

# Discrete events

## Contents


- Set Theory Review: Probability of an Event
- Intersection: AND
- Union: OR
- Definition of Independence
- Conditional probability
- Total probability theorem


Before going further into *continuous multivariate distributions,* we will start with a reminder of some basic concepts you have seen previously: independence and conditional probability, illustrated by considering the probability of *events.*

> **"Set Theory Review: Probability of an Event"**
>
> This page is review of a BSc-level probability course. If you need further practice or to revisit other concepts you should refer to the online resource Probability and Statistics for Engineers as needed.

## Set Theory Review: Probability of an Event

In probability theory, the sample space ($S$) is the collection of all possible outcomes arising from an experiment or operation which involves chance. An event is the specific outcome, or set of outcomes, from the experiment or operation.

For example, many natural disasters can be considered as binary events: either they occur or don't occur. This is analogous to flipping a coin, where the 'flip' is our experiment and heads or tails define the two possible events. Our sample space is the set of two possible events, where conventionally we will use $A$ and $B$ to represent each event, and the probability associated with each is $P(A)$ and $P(B)$.

Recall also that the complement of each event is denoted with the “bar” symbol over the top, $\bar{A}$. This is the probability of event $A$ *not* happening and the probability is:

$$P(\bar{A}) = 1 - P(A)$$

# Case Study: discrete events

Imagine two events, $A$ and $B$, where:

- $A$ represents river A flooding
- $B$ represents river B flooding

Each of these events will have a probability of occurring, denoted here as $P(A)$ and $P(B)$, respectively, and illustrated in the figure below in the Venn diagram. Recall that the axioms of probability imply that the area of the box represents probability 1.0 (the sample space, $S$), and that the shapes in the sample space correspond to probabilities of various events. Based on your understanding of the Venn diagram and probability theory, we can make the following observations from the figure (each statement is provided in layperson terms followed by probability terminology):

- *Both rivers are flooded at the same time:* the probability that $A$ and $B$ occur together is not zero (the overlapped area of the circles).
- *Either river A or B or both rivers are flooded:* the probability is the combined area of both circles.
- *Neither A nor B are flooded:* the probability outside (both) circles is not zero
- *One river is flooded while the other is not:* the probability is 1 minus the area of either circle, respectively (i.e., two probabilities, depending on the river, which are not necessarily equal).

These observations and concepts are defined in more detail below.

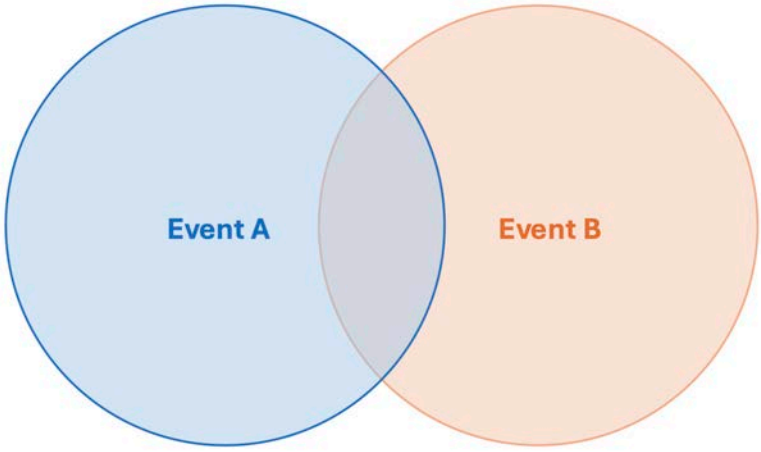



*Fig. 6.2* Venn diagram of arbitrary events $A$ and $B$.

# Intersection: AND

The *intersection* of the events $A$ and $B$, denoted $P(A \cap B)$, is defined as the probability that both events occur; thus, it would be represented in our diagram as shown in the figure below. Going back to our example, it represents the probability that both rivers get flooded at the same time.

**Definition**

The **AND probability** is the *intersection* of two events $P(A \cap B)$.

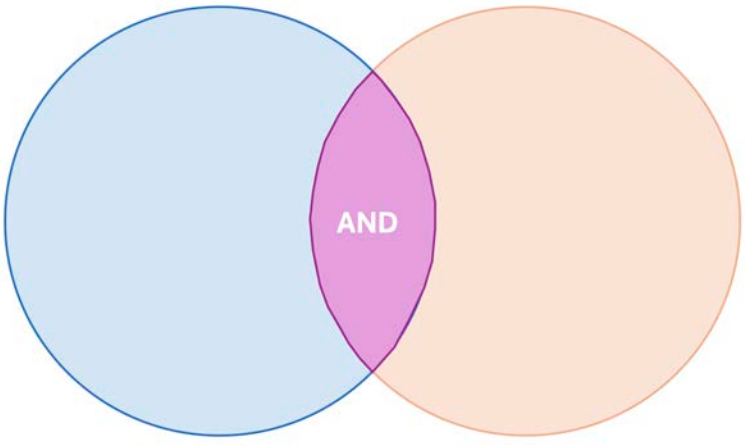



*Fig. 6.3* Venn diagram of the events $A$ and $B$, and AND probability.

# Union: OR

The *union* of events $A$ and $B$, $P(A \cup B)$, is defined as the probability that either either event occurs, or both. In our example, it would represent the probability that either river floods, or both. This probability can be computed as

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

This is, we add the probabilities of occurrence of the event $A$ and $B$ and we deduct the intersection (AND), to avoid counting this area twice. Note that the *union* is directly related to the *intersection* of the same events!

**Definition**

The **OR probability** is the union of two events $P(A \cup B)$.

# Definition of Independence

When two events, $A$ and $B$ are independent, it means that the occurrence of one does not influence the occurrence of the other. Formally, $A$ and $B$ are considered independent **if and only if** the AND probability ($P(A \cap B)$) can be factorized into the product of their probabilities. That is:

$$P(A \cap B) = P(A)P(B)$$

As long as we know the probabilities of each event the computation of the AND probability is a simple multiplication problem. This simplicity is why the assumption of independent events is often invoked.

Although the assumption of independence makes our modelling and programming efforts more simple, it is often not a good representation of reality; improving our ability to model these situations is the primary motivation for this chapter, which will provide alternative methods for computing $P(A \cap B)$.

> **Tip**
>
> The primary purpose of this chapter is to facilitate the computation of the AND probability for *dependent* events and *dependent* random variables (the next page), which can be simply described as the case:
>
> $$P(A \cap B) \neq P(A)P(B)$$

# Conditional probability

A *conditional* probability describes the probability of an event given that another event is known to have occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

where $P(A|B)$ is the probability of the event $A$ given that event $B$ is known to have occurred. Note that when reading the equation the vertical bar $|$ is read out loud as "given," and identifies the combination of events $A|B$ as "$A$ conditioned on $B$."

The concept can be illustrated graphically by recognizing that once $B$ is known to have occurred, it's probability is now 1.0 and the task at hand is to determine the probability of $A$. Using the figure above, this is equivalent to taking the area of the circle $B$ as the new sample space and noting the probability $P(A)$ is given by the fraction of the circle where $A$ is overlapped: the *intersection*, $P(A \cap B)$. As the probability of $B$ is now 1.0, we must scale $P(A \cap B)$ proportionally (normalizing by $P(B)$), thus arriving at the equation above: simple!

# Conditional Probability of Independent Events

What happens to the conditional probability if the events are *independent?*

$$P(A|B) = \frac{P(A \cap B)}{P(B)} = \frac{P(A)\cancel{P(B)}}{\cancel{P(B)}} = P(A)$$

Thus, the probability of $A$ given $B$ is simply the probability of $A$.

**Interpretation of Independence**

When $A$ and $B$ are *independent* the probability of $A$ is not influenced by the probability of $B$.

This can be interpreted graphically by recognizing that the area of the overlapped area of events $A$ and $B$ is equivalent to the product of the area of the individual circles.

**Tip**

It is common to confuse *mutually exclusive* events with *independent* events. The former states that two events cannot happen simultaneously, which implies that $P(A|B) = P(A)P(B) = 0$, whereas the latter dictates *how* the AND probability can be computed (and generally speaking $P(A|B)$ is not 0, although it can be).

Refer to the reference probability book for a refresher, if needed.

Although the definition above is illustrated with a simple case of two discrete events, it is essential to remember it when working with more complicated situations, for example multivariate probability distributions for continuous random variables.

# Total probability theorem

When the probability of an event cannot be calculated directly, but its occurrence is related to the occurrence of other events, the Total probability theorem can be applied as

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i)$$

where $B_i$ are a set of *mutually exclusive* and *collectively exhaustive* events. This result can also be derived graphically using a Venn diagram with multiple events in the sample space.

If you need further refresher of these concepts, you can further read and practice here.

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Continuous Random Variables

## Contents


- Case Study: Two Rivers
- From Discrete to Continuous
- Intersection: AND
- Union: OR
- Joint Exceedance
- Definition of Independence
- Definition of Dependence
- Conditional Probability
- Summary


This page covers fundamental concepts for *continuous random variables.* As we are interested in considering more than one variable simultaneously, the term *multivariate* is used. We will start by translating the concepts covered on the previous page from *discrete events*, allowing us to arrive at a clear understanding of the concept of probabilistic *dependence* for multivariate continuous random variables.

## Case Study: Two Rivers

Here we will illustrate probability computations using a bivariate case. Consider the discharge of two rivers that are located in the same watershed, $Q_1$ and $Q_2$. There are 34 observations each of the discharges that were taken at the same time (this is important for multivariate distributions!). The observations are illustrated in the figure below (panel (a)).

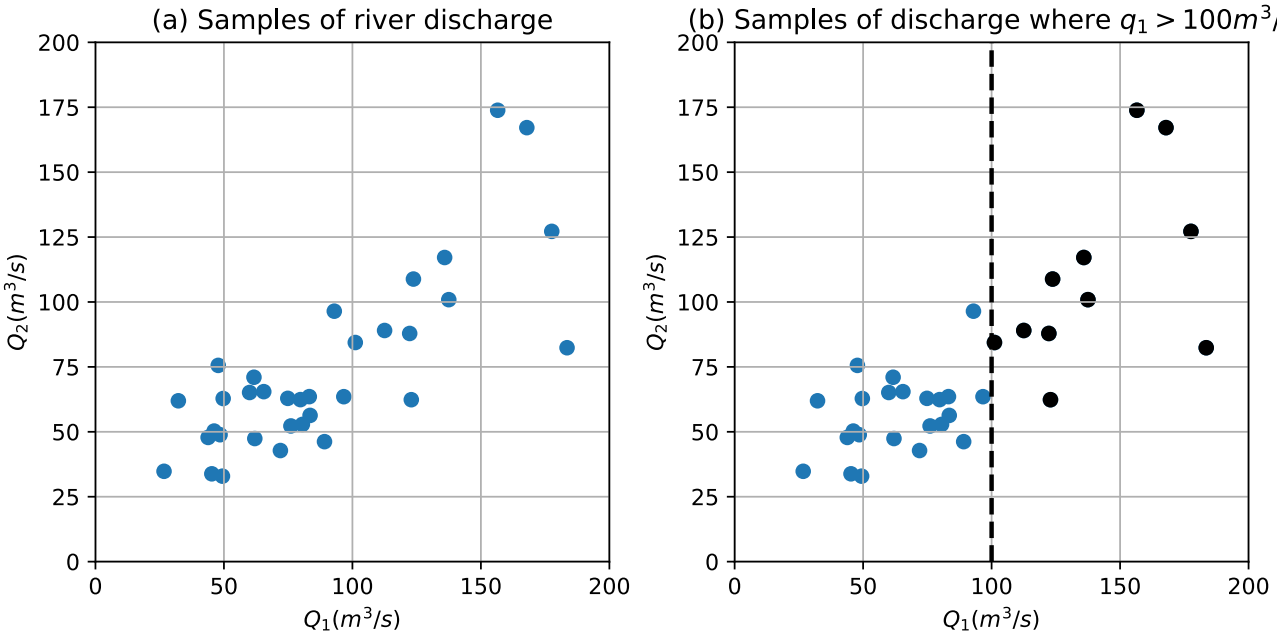



*Fig. 6.4* Samples of the discharges of two rivers ($Q_1$ and $Q_2$): (a) paired observations, and (b) paired observations highlighting data for the case $q_1 > 100$ m$^3$/s.

Panel (b) in the figure above highlights the number of events where $q_1 > 100$ m$^3$/s. We can thus compute the empirical probability:

$$P[q_1 > 100 \text{ m}^3/\text{s}] = 11/34 = 0.32$$

As covered in the univariate continuous distribution chapter, this is equivalent to evaluating the empirical distribution $F_{Q_1}(q_1)$.

**Tip**

The theory presented below applies to univariate and multivariate distributions in general (i.e., $f_X(x)$, $F_X(x)$). However, this page keeps things simple by limiting probability computations to *empirical* bivariate distributions (i.e., scatter plots of the river discharge data, as above). Following pages introduce theoretical continuous multivariate distributions.

# From Discrete to Continuous

Although the previous page considered *discrete events* $A$ and $B$, the interpretation of probabilistic concepts are directly analogous to the case of *continuous random variables.*

# Intervals

Recall that a continuous random variable $X$ can take an infinite number of values; this is because a *realization* of the random variable, $x$, can be any real number (if the distribution is unbounded):

$$x \in \mathbb{R}$$

We are typically interested in a specific combination of values of a random variable, thus it is useful to denote an *interval* as $\Omega$, for example

$$\Omega = x \in \mathbb{R} : a \leq x < b$$

defines an interval of the random variable $X$ between $x = a$ and $x = b$. Such intervals could be defined as the need arises, for example, $\Omega$ contains the set $x \in \mathbb{R}$ such that:

$$\begin{cases} x \leq a \\ x > b \end{cases}$$

However, most of the time we are interested in intervals where the random variable is greater than or less than a specific value.

**Definitions**

Given a specific value of interest $x^*$ for a random variable $X$, **exceedance** is the condition:

$$\Omega_e = x \in \mathbb{R} : x > x^*$$

Similarly, *non-exceedance* is the condition:

$$\Omega_{ne} = x \in \mathbb{R} : x \leq x^*$$

Note that $\Omega_{ne}$ is the complement of $\Omega_e$.

Having defined intervals, it should now be obvious how to translate the concepts from discrete events to continuous random variables:

$$A \rightarrow \Omega$$

A discrete event is analogous to an interval $x \subseteq \mathbb{R}$, and instead of the probability of event $A$, $P(A)$, we will refer to the probability of a realization $x$ being in the interval $\Omega$, denoted $P[\Omega]$.

In the case of *two* random variables, $\Omega$ is defined for both, and the *interval* becomes a *region:*

$$\Omega = \begin{cases} x \in \mathbb{R}: & x \subseteq \Omega_x \\ y \in \mathbb{R}: & y \subseteq \Omega_y \end{cases}$$

We can generalize these statements to define the *region of interest* over which we would like to integrate the multivariate probability density function to obtain the probability associated with that region.

**Region of Interest as a Set of Intervals**

Consider a multivariate distribution that describes a vector random variables $X = X_1, X_2, \ldots, X_n$, composed of $n$ random variables $X_i$. The region of interest $\Omega$ is defined as the set:

$$\Omega = \{\Omega_1, \Omega_2, \ldots, \Omega_n\}$$

where $\Omega_i$ is the interval over which the probability density function is integrated, for each random variable $X_i$.

Although it is difficult to construct the multivariate distribution and completely define $\Omega$ in practice (especially for dimension $n > 2$), the theoretical framework introduced in this chapter applies for all multivariate probability distributions, regardless of the model or approach used.

For now we will consider only the case where the multivariate region $\Omega$ consists of sub-intervals defined for each of the random variables individually, as in the definition above. In a later section we will describe $\Omega$ as a *function of random variables.* In addition, to easily distinguish the individual random variables, we introduce the term *marginal*:

**Definition**

A **marginal** distribution is the univariate distribution associated with a single random variable that is part of a multivariate distribution.

The probability $q_1 > 100$ m$^3$/s computed in the Case Study Introduction section is a marginal probability.

# One Random Variable

The distribution of $X$ is described with a probability *density* function (PDF), $f_X(x)$. Integration of the *density* function over a specific interval $\Omega$ gives the *probability* of the random variable $X$ taking a value $x$ within that interval, for example:

$$P[\Omega] = \int_{\Omega} f_X(x)\,\mathrm{d}x$$

A commonly used interval is $\Omega_{ne}$, the non-exceedance interval described above:

$$\Omega = \{\Omega_{ne}\} \quad \rightarrow \quad P[\Omega_{ne}] = P[X \leq x^*] = F_X(x^*) = \int_{-\infty}^{x^*} f_X(x)\,\mathrm{d}x$$

where $F_X(x^*)$ is the cumulative *distribution* function (CDF), which by definition evaluates interval $\Omega_{ne}$.

**Definitions**

The **exceedance probability**, $P[\Omega_e]$, and **non-exceedance probability**, $P[\Omega_{ne}]$ are:

$$P[\Omega_e] = P[X > x^*] = 1 - F_X(x^*)$$

$$P[\Omega_{ne}] = P[X \leq x^*] = F_X(x^*)$$

where $x^*$ and $y^*$ are values of interest for each random variable.

# Two Random Variables

Extending the notation above for the case of two random variables $X$ and $Y$, the *bivariate* distribution (i.e., the *multivariate* distribution for two random variables) has PDF $f_{X,Y}(x, y)$ and CDF $F_{X,Y}(x, y)$, respectively. Probabilities can be computed as follows:

$$P[X \leq x, Y \leq y] = F_{X,Y}(x,y) = \int_{-\infty}^{y}\int_{-\infty}^{x} f_{X,Y}(x,y)\,\mathrm{d}x\,\mathrm{d}y$$

The equation above is the the multivariate CDF and also illustrates a specific combination of two intervals (a region in the $X, Y$ plane), described here with $x^*$ and $y^*$ as arbitrary values of interest:

$$\Omega = \{\Omega_{x,ne}, \Omega_{y,ne}\} = \begin{cases} x \in \mathbb{R}: & -\infty \leq x \leq x^* \\ y \in \mathbb{R}: & -\infty \leq y \leq y^* \end{cases}$$

# Empirical Computation

Let us now apply this equation to compute a probability: returning to our case study with two rivers, consider the situation where we are interested in *low* river discharges (e.g., a dry condition that impacts water supply). Assume the values of interest are when $q_1 = 100$ m$^3$/s and $q_2 = 75$ m$^3$/s ($x^*$ and $y^*$, respectively). Noting that there are 21 points this region (see figure below), the probability of these conditions is computed as follows:

$$P[Q_1 \leq q_1, Q_2 \leq q_2] = F_{Q_1,Q_2}(\text{ m}^3\text{/s},\ \text{m}^3\text{/s}) = 21/34 = 0.62$$

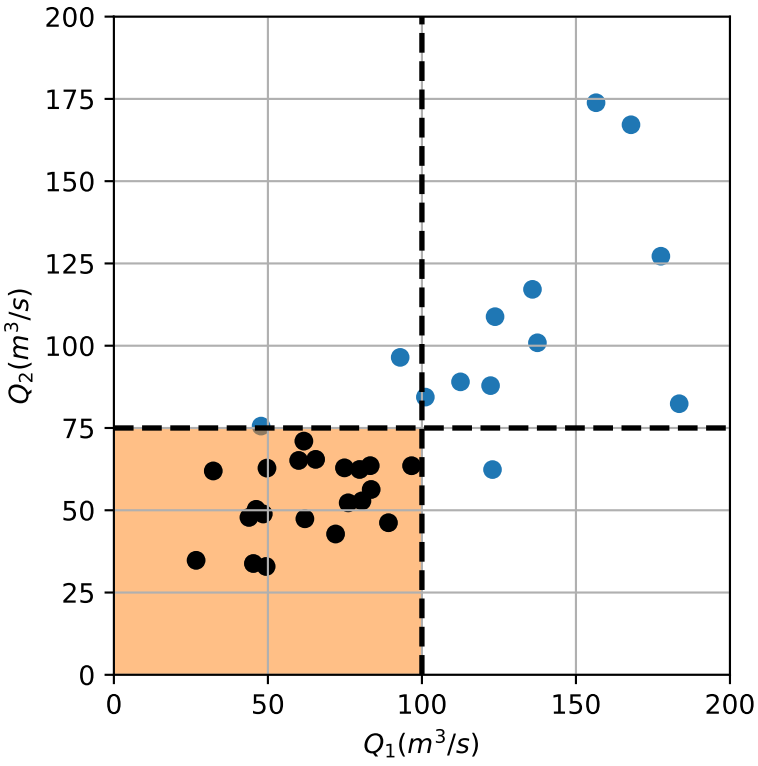



*Fig. 6.5* Samples of the discharges of two rivers ($Q_1$ and $Q_2$),highlighting the region, $q_1 \leq 100$ m$^3$/s and $q_2 \leq 75$ m$^3$/s, which contains 21 observations (34 total).

# Joint Probability

The probability computed above is the direct evaluation of the multivariate CDF and is called the *joint non-exceedance* of $X$ and $Y$. The term *joint* arises from the need to describe more than one variable in a multivariate context. This leads to additional definitions:

**Definition**

The distribution of more than one random variable described in the same probability space is a **joint distribution**. For the bivariate case the **joint probability density function** (PDF) is $f_{X,Y}(x, y)$. A **joint probability**, $P[\Omega]$, is the probability found by integrating the joint PDF over the region of interest:

$$P[\Omega] = \int_{\Omega} f_{X,Y}(x, y) \, \mathrm{d}x \, \mathrm{d}y$$

where $\Omega$ is the set:

$$\Omega = \{\Omega_1, \Omega_2\}.$$

This is shown for $n = 2$ random variables here, but the concept can easily be extended to higher dimensions.

We can now extend the univariate non-exceedance probability for the bivariate case:

**Definition**

The **joint non-exceedance probability**, $P[\Omega_{ne}]$, is:

$$P[\Omega_e] = P[X \leq x^*, Y \leq y^*] = F_{X,Y}(x^*, y^*)$$

where

$$\Omega = \{\Omega_{x,ne}, \Omega_{y,ne}\}$$

and $x^*$ and $y^*$ are values of interest for each random variable.

This definition can be generalized for more than two random variables and implies that the density functions of *all* random variables are jointly integrated in the **lower** tails ( $\Omega_{ne}$).

Similarly, for the exceedance case:

**Definition**

The **joint exceedance probability**, $P[\Omega_e]$, is:

$$P[\Omega_e] = P[X > x^*, Y > y^*]$$

where

$$\Omega = \{\Omega_{x,e}, \Omega_{y,e}\}$$

and $x^*$ and $y^*$ are values of interest for each random variable.

This definition can be generalized for more than two random variables and implies that the density functions of *all* random variables are jointly integrated in the **upper** tails ($\Omega_e$ ).

*Note that the calculation of the probability $P[\Omega_e]$ is left as an exercise for the reader at the end of this section, as it requires applying concepts introduced below.*

The definitions above generalize probability computations and illustrates the connection between the multivariate distribution and a region in the random variable space. It also indicates three essential ingredients required to find the joint probability:

1. The joint distribution (e.g., the bivariate case $f_{X,Y}(x, y)$).
2. The region of interest $\Omega$.
3. Integration of the density function over the region of interest.

The integral of the joint PDF is directly analogous to integrating a univariate *density* function (PDF) to obtain the cumulative distribution function (CDF), and thus a *probability.* However, whereas in 1D the integration region is simply an interval on the number line that can be explicitly defined with two values of $x$, in higher dimensions the region can be more difficult to define.

On this page, the region $\Omega$ over which the *joint probability* is calculated is found by combining the regions $\Omega_x$ and $\Omega_y$ as a union or intersection. In other words, working with a combination of the four rectangular subregions portions of the 2D variable space. However, for many practical applications more complex regions must be considered; for example, a parametric equations that define a specific subregion (or multiple subregions) of the multivariate sample space. For example, evaluating the discharge of a river, $Q$, that is formed by the joining of the two rivers in our example: $Q = Q_1 + Q_2$. Later in this chapter we will consider this *function of random variables.* For now, we will continue with the rectangular subregions by revisiting the AND and OR probabilities from the discrete event case.

# Intersection: AND

As in the case of discrete events, *intersection* is the probability that a specific set of events occur together, although now we consider a set of intervals on the number line for each random variable, $\Omega_i$, for all $i$. This is directly analogous to the overlapping areas of the Venn diagrams; however, in this case the overlapping area is described in the real number plane (when dealing with two random variables). It is the region where *all* sub-regions are overlapped.

**Definition**

The **AND probability** is the *intersection* of a set of $n$ events $\Omega_i$ is:

$$\Omega_{AND} = \{\Omega_1 \cap \Omega_2 \cap \ldots \cap \Omega_n\}$$

**If and only if** the random variables are statistically independent, the probability can be evaluated as a product of the marginal probabilities:

$$P[\Omega_{AND}] = \prod_{i=1}^{n} P[\Omega_i]$$

When the assumption of statistical independence is not valid, the probability $p_{AND}$ must be computed by other methods.

Within many engineering and science applications the AND probability is often conventionally associated with the *joint exceedance* case to distinguish from the joint non-exceedance case.

> **Tip**
>
> The term *AND probability* can be assumed to mean *joint exceedance*, unless explicitly stated otherwise.

Computing the AND probability is not as straightforward as the non-exceedance probability: it requires the use of conditional probability. However, it *is* possible to compute the AND probability empirically, so let's try it!

# Empirical Computation

To compute the AND probability for *joint exceedance* we can apply a similar approach as done for non-exceedance probability above, except now we consider the case where $q_1 > 100$ m$^3$/s and $q_2 > 120$ m$^3$/s is highlighted. Thus, we can compute the joint probability of exceedance $P[q_1 > 100, q_2 > 120]$. Counting the number of observations in that region results in:

$$P[q_1 > 100, q_2 > 120] = 3/34 \approx 0.09$$

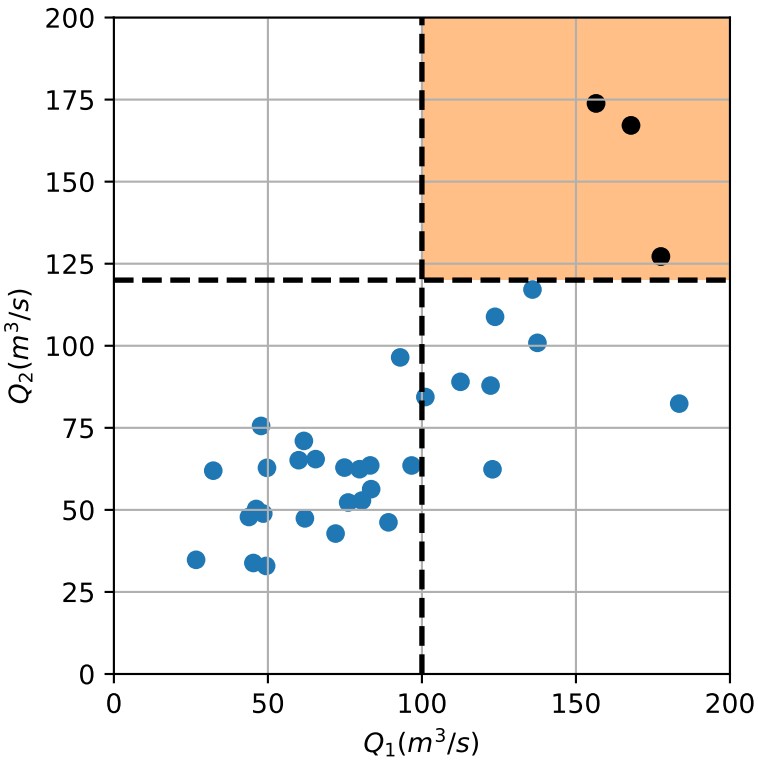



*Fig. 6.6* Samples of the discharges of two rivers ($Q_1$ and $Q_2$), highlighting the *joint exceedance* region, $\Omega_e$, where $q_1 > 100$ m$^3$/s and $q_2 > 120$ m$^3$/s. Note $P[\Omega_e]$ is *not* found by the complement of the joint CDF, $P[\Omega_e] \neq 1 - F_{Q_1,Q_2}(q_1, q_2)$ (explained below).

# Union: OR

As with _intersection, the *union* of events can be adapted from the discrete event case. For continuous random variables the region of interest becomes the total area of the real number space covered by *any* of the sub-regions of interest for each random variable.

**Definition**

The **OR probability** is the *union* of a set of $n$ events $\Omega_i$ is

$$\Omega_{OR} = \{\Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_n\}$$

For computing probability, it is easier to express the set as a combination of *intersections.* Using $\overline{\Omega}$ to denote the complement of $\overline{\Omega}$, De Morgan's laws allow us to rewrite the *union* as

$$\begin{aligned} P[\Omega_{OR}] = & \\ & = 1 - P[\,\overline{\Omega_1 \cup \Omega_2 \cup \ldots \cup \Omega_n}\,] \\ & = 1 - P[\,\overline{\Omega_1}\,] \cap P[\,\overline{\Omega_2}\,] \cap \ldots \cap P[\,\overline{\Omega_n}\,] \\ & = 1 - \Big(1 - P[\,\Omega_1\,]\Big)\Big(1 - P[\,\Omega_2\,]\Big)\ldots\Big(1 - P[\,\Omega_n\,]\Big) \\ & = 1 - \prod_{i=1}^{n}\Big(1 - P[\Omega_i]\Big) = 1 - \prod_{i=1}^{n}\Big(P[\overline{\Omega_i}]\Big) \end{aligned}$$

In other words, the OR probability is the *complement* of the probability associated with a region $\overline{\Omega}$ that overlaps with *none* of the sub-regions. For the bivariate case, this simplifies to:

$$P[\Omega_{OR}] = P[\Omega_x] + P[\Omega_y] - P[\Omega_x \cap \Omega_y]$$

**If and only if** the random variables are statistically independent, the bivariate OR probability can thus be evaluated as:

$$P[\Omega_{OR}] = P[\Omega_x] + P[\Omega_y] - P[\Omega_x]P[\Omega_y]$$

When the assumption of statistical independence is not valid, the probability $p_{OR}$ must be computed by other methods.

# Empirical Computation

Moving now to the OR case, we wish to compute the probability:

$$P\left[\Omega_{OR}\right] = P\left[\Omega_{q_1,ne}, \Omega_{q_2,ne}\right] = P(q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}]$$

The region $\Omega_{OR}$ is illustrated in panel (c) of the figure below, the area where is highlighted. We could count the samples in that area and compute the probability as before:

$$P[q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}] = 24/34 = 0.71$$

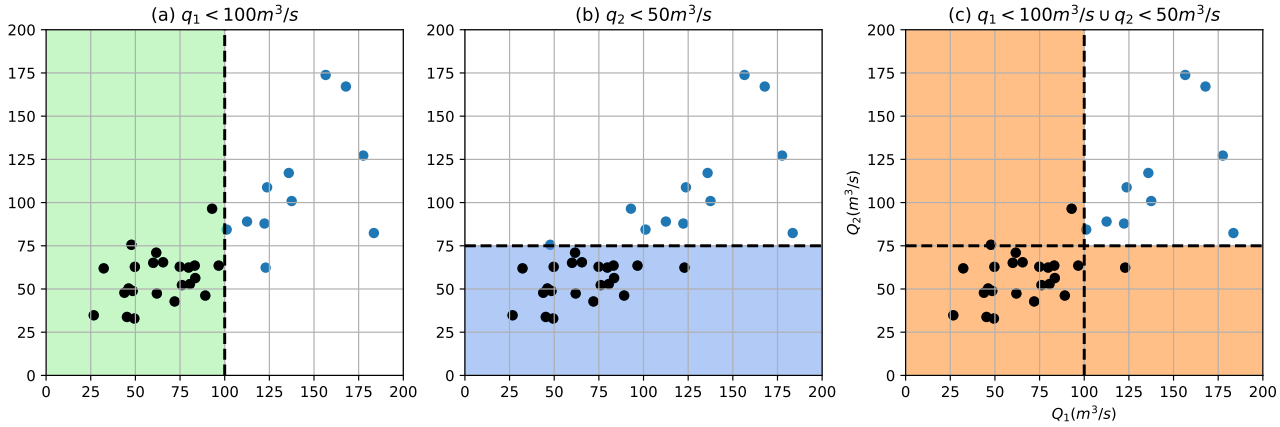



*Fig. 6.7* Samples of the discharges of two rivers ($Q_1$ and $Q_2$): (a) highlighting $\Omega_{q_1,ne}$, where $q_1 \leq 100$ m$^3$/s, (b) highlighting $\Omega_{q_2,ne}$, where $q_2 \leq 75$ m$^3$/s, and (c) highlighting $\Omega_{OR} = \left\{\Omega_{q_1,ne}, \Omega_{q_2,ne}\right\}$, where $q_1 \leq 100$ m$^3$/s $\cup$ $q_2 \leq 75$ m$^3$/s.

A graphical approach can be applied that is directly analogous to the approach with discrete events (i.e., a Venn diagram): the sum of the marginal probabilities (panels (a) and (b) in the figure above) minus the joint non-exceedance probability (already computed in this section above), which must be removed as otherwise we would be counting it twice.

$$\begin{aligned} P\left[q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}\right] = \\ = P\left[q_1 \leq 100 \text{ m}^3\text{/s}\right] + P\left[q_2 \leq 75 \text{ m}^3\text{/s}\right] - P\left[q_1 \leq 100 \text{ m}^3\text{/s} \cap q_2 \leq 75 \text{ m}^3\text{/s}\right] \\ = (23 + 22 - 21)/34 = 24/34 = 0.71 \end{aligned}$$

# Joint Exceedance

As previously mentioned, when evaluating a multivariate (here bivariate) cumulative distribution function, we obtain joint probabilities,

$F_{X,Y}(X \leq x, Y \leq y) = P[X \leq x, Y \leq y]$, also called the joint non-exceedance $P[\Omega_{ne}]$. However, we are often interested $F_{X,Y}(X > x, Y > y) = P[X > x, Y > y]$, the joint exceedance $P[\Omega_e]$. In particular, one should recognize the following:

$$F(X > x, Y > y) = P[X > x, Y > y] \neq 1 - F(X \leq x, Y \leq y)$$

Assuming that the complement of the multivariate CDF is equal to the joint exceedance probability is a common misconception.

**Exercise**

Using only empirical probabilities for the marginal random variables and the joint CDF, apply the graphical approach illustrated in the preceding section to find the joint exceedance probability $p = P[q_1 > 100 \text{ m}^3\text{/s} \cap q_2 > 120 \text{ m}^3\text{/s}]$, the area illustrated here:

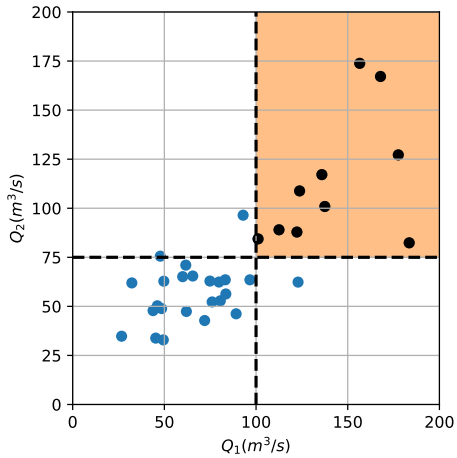



*Fig. 6.8* Illustration of the probability to be computed in the exercise: $p = P[q_1 > 100 \text{ m}^3\text{/s} \cap q_2 > 75 \text{ m}^3\text{/s}]$.

**Solution**

Observe that the bivariate region is divided into four rectangular sub-regions in the figure, and that our goal is to compute the probability associated with the upper-right sub-region. However, we can only compute the probability of the lower-left sub-region (using the joint CDF) and the marginal probabilities, each of which makes up a combination of two adjacent horizontal or vertical sub-regions (i.e., there are four possible combinations).

To compute the upper-right region, our only option is to recognize that it is the *complement* of everything else. This is actually the complement of the OR probability computed in the previous section, so we know the answer is:

$$p = 1 - 0.71 = 0.29$$

The analytic expression is thus:

$$\begin{aligned} p &= 1 - P[q_1 \leq 100 \text{ m}^3\text{/s}] - P[q_2 \leq 75 \text{ m}^3\text{/s}] \\ &\quad + P[q_1 \leq 100 \text{ m}^3\text{/s} \cap q_2 \leq 75 \text{ m}^3\text{/s}] \\ &= 1 - (23 + 22 - 21)/34 = 24/34 \\ &= 0.29 \end{aligned}$$

Where the probabilities are calculated empirically, as above. Note that we can confirm our result is correct by counting the samples in the region of interest:

$$p = 10/34 = 0.29$$

# Definition of Independence

When two random variables, $X$ and $Y$, are independent, the value of one variable does not influence the value of the other variable.

**Definition**

**Independence**: $X$ and $Y$ are considered independent **if and only if** the joint probability function (or cumulative distribution function) can be factorized into the product of their marginal probability functions (or cumulative distribution functions). That is, if:

$$F_{X,Y}(x,y) = F_X(x)F_Y(y)$$

Or, alternatively:

$$P\big[X < x \cap Y < y\big] = P\big[X < x\big]P\big[Y < y\big]$$

The relationship above highlights the connection between the joint cumulative distribution function (CDF) and the marginal CDFs of two independent random variables, $X$ and $Y$.

# Definition of Dependence

In contrast to the definition of independence, $X$ and $Y$ are considered dependent when the probability associated with one marginal variable influences the probability associated with the state of another marginal random variable and, thus, we cannot make use of the above simplification. This can be formally stated using conditional probability:

**Definition**

**Dependence**: random variables $X$ is dependent on $Y$ if

$$P\big[X|Y\big] \neq P\big[X\big]$$

# Empirical Computation

The effect of dependence can be readily seen by returning to our case study of two rivers and considering the difference between our empirical approach (i.e., an approximation of the "true" probability), and an approach where one assumes the two random variables are *independent.*

Recall from the top of this page that the non-exceedance probability $P[\Omega_{ne}]$ was computed empirically (as well as graphically):

$$P[q_1 \leq 100 \text{ m}^3\text{/s} \cap q_2 \leq 75 \text{ m}^3\text{/s}] = 21/34 = 0.62$$

If we would have computed the same probability using the definition of AND probability, we would obtain:

$$\begin{aligned} P[q_1 \leq 100 \text{ m}^3\text{/s} \cap q_2 \leq 75 \text{ m}^3\text{/s}] = \\ &= P[q_1 \leq 100 \text{ m}^3\text{/s}]P[q_2 \leq 75 \text{ m}^3\text{/s}] \\ &= (23/34)(22/34) \\ &= 0.68 \cdot 0.65 = 0.44 \end{aligned}$$

The difference between 0.62 and 0.44 is significant! Why does this difference occur? Because there is *dependence* between $Q_1$ and $Q_2$. Following the definition, we can see that the probability of observing a specific interval of $Q_1$ is influenced by the state of $Q_2$: for low values of $Q_2$, the observations of $Q_1$ are small; the converse is also true.

Another way of thinking would be to consider the probability calculated using the assumption on independence, 0.11. If this assumption were true, for the 34 samples in the case study, we would expect around 15 of them to be in the region $\Omega_e = \{Q_1 > q_1, Q_2 > q_2\}$, as $0.44 \cdot 34 \approx 15$. However, there are 21 observations in that region, implying dependence between $Q_1$ and $Q_2$ (e.g., positive correlation).

**Exercise**

Assuming $Q_1$ and $Q_2$ are independent, compute the following probability and compare it to the empirical value:

$$P[q_1 > 100 \text{ m}^3\text{/s} \cap q_2 > 75 \text{ m}^3\text{/s}]$$

**Solution**

Recall from the previous example that the following AND probability was computed empirically (as well as graphically):

$$P[q_1 > 100 \text{ m}^3\text{/s} \cap q_2 > 75 \text{ m}^3\text{/s}] = 10/34 = 0.29$$

If we would have computed the same probability using the expression from the AND probability definition, we would obtain:

$$\begin{aligned} P[q_1 > 100 \text{ m}^3\text{/s} \cap q_2 > 75 \text{ m}^3\text{/s}] & \\ &= P[q_1 > 100 \text{ m}^3\text{/s}]P[q_2 > 75 \text{ m}^3\text{/s}] \\ &= (11/34)(12/34) \\ &= 0.32 \cdot 0.35 = 0.11 \end{aligned}$$

There is a significant difference between both approaches (0.29 vs. 0.11), illustrating the importance of dependence.

**Exercise**

This is an OR probability. Assuming $Q_1$ and $Q_2$ are independent, compute the following probability and compare it to the empirical value found above:

$$P[q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}]$$

**Solution**

First, recall the empirical result:

$$P[q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}] = 24/34 = 0.71$$

If we would have computed the same probability using the expression from the OR probability definition, we would obtain:

$$\begin{aligned} &P[q_1 \leq 100 \text{ m}^3\text{/s} \cup q_2 \leq 75 \text{ m}^3\text{/s}] \\ &\quad = P[q_1 \leq 100 \text{ m}^3\text{/s}] + P[q_2 \leq 75 \text{ m}^3\text{/s}] - P[q_1 \leq 100 \text{ m}^3\text{/s}]P[q \\ &\quad = (23/34) + (22/34) - (23/34)(22/34) \\ &\quad = 0.68 + 0.65 - 0.68 \cdot 0.65 = 0.89 \end{aligned}$$

There is a significant difference between both approaches (0.71 vs. 0.89), illustrating the importance of dependence.

# Conditional Probability

A conditional probability is a measure of the probability of an event occurring given that another event has already occurred. The following relationship holds for events X and Y:

$$P[X < x | Y < y] = \frac{P[X < x \cap Y < y]}{P[Y < y]} = \frac{F_{X,Y}(x,y)}{F_Y(y)}$$

$$F_{X,Y}(x,y) = F_{X|Y}(x|y)F_Y(y) = F_{Y|X}(y|x)F_X(x)$$

Where:

1. $P[x < X | Y < y]$ represents the conditional probability that $X$ is less than a certain value $x$, given that event $Y$ is less than a certain value $y$.
2. $P[X < x \ \cap \ Y < y]$ represents the joint probability that both $X$ is less than $x$ and $Y$ is less than $y$.
3. $F_{X,Y}(x, y)$ represents the joint cumulative distribution function (CDF) of $X$ and $Y$, which gives the probability that both $X$ and $Y$ are less than or equal to their respective values $x$ and $y$.
4. $F_X(x)$ represents the marginal cumulative distribution function (CDF) of $X$, which gives the probability that $X$ is less than or equal to $x$.
5. $F_Y(y)$ represents the marginal cumulative distribution function (CDF) of $Y$, which gives the probability that $Y$ is less than or equal to $y$.

Imagine that we have a rudimentary measurement device of the discharges in one river, so we know that $q_1 > x$. If $Q_2$ is **independent** of $Q_1$, knowing the value of $Q_1$ does not provide us with any information so

$$P[q_2 > y | q_1 > x] = \frac{P[q_2 > y \ \cap \ q_1 > x]}{P[q_1 > x]} = \frac{P[q_2 > y] \cancel{P[q_1 > x]}}{\cancel{P[q_1 > x]}} = P[q_2 > y]$$

However, since these rivers are located close to each other, it is likely that they belong to the same system and their discharges are the result of similar drivers (e.g., rain). Therefore, knowing information about one discharge gives us information about the discharge of the other river and, thus, $Q_1$ and $Q_2$ are expected to be **dependent** on each other. We could compute $P[q_2 > y | q_1 > x]$ using the joint probability distribution of $Q_1$ and $Q_2$ to evaluate $P[q_2 > y \ \cap \ q_1 > x]$ accounting for the dependence.

# Empirical Computation

Imagine that we know $q_1 > 130$ m$^3$/s and want to know the probability of $q_2 > 150$ m$^3$/s. Thus, our goal is to compute

$$P[q_2 > 150 \text{ m}^3\text{/s} \mid q_1 > 130 \text{ m}^3\text{/s}]$$

As shown in panel (a) in the figure below, there are 6 samples where $q_1 > 130$ m$^3$/s. This sub-region of $Q_1, Q_2$ becomes the new sample space within which we are now evaluating the probability $q_2 > 150$ m$^3$/s. This is the effect of "conditionalizing" on $Q_1$: normalizing the probability space. It is directly analogous to the discrete event case.

In panel (b), we can see that two of those samples fulfill that $q_2 > 150$ m$^3$/s. Therefore, we can compute the aforementioned conditional probability as

$$P[q_2 > 150 \text{ m}^3\text{/s} | q_1 > 130 \text{ m}^3\text{/s}] = 2/6 = 0.33$$

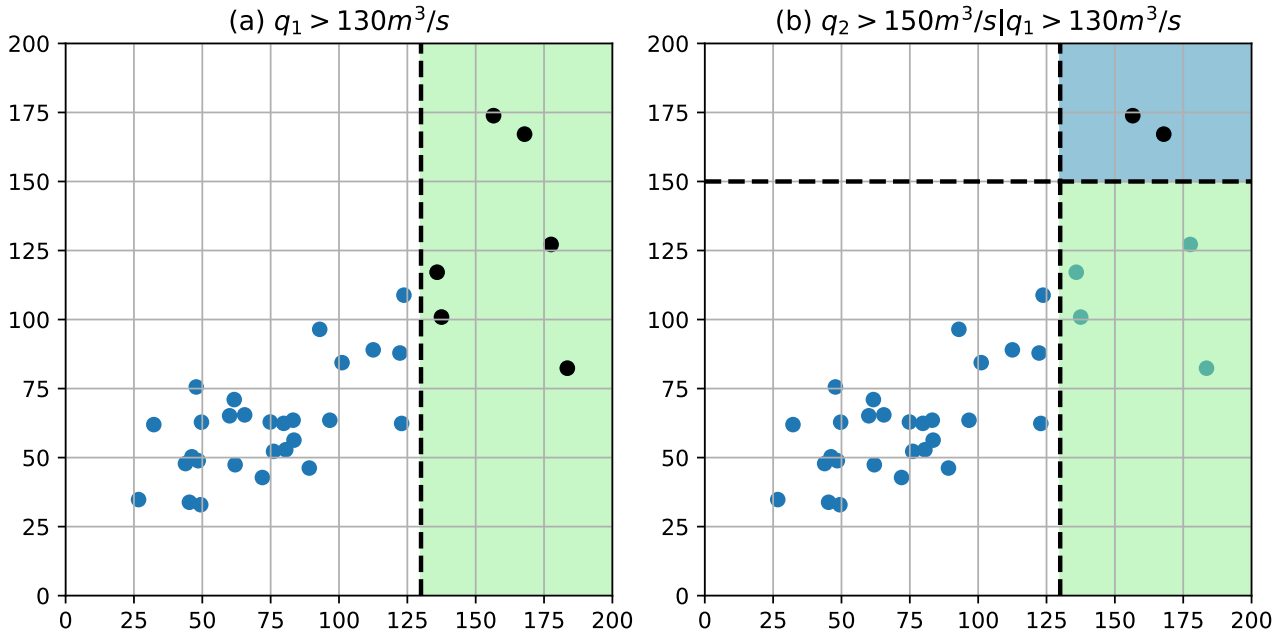



*Fig. 6.9* Samples of the discharges of two rivers ($Q_1$ and $Q_2$): (a) highlighting those where $q_1 > 130$ m$^3$/s, (b) highlighting where $q_1 > 130$ m$^3$/s, and $q_2 > 150$ m$^3$/s.

Note that if we would assume $Q_1$ and $Q_2$ are independent, we would obtain

$$P[q_2 > 150 \text{ m}^3\text{/s} | q_1 > 130 \text{ m}^3\text{/s}] = P[q_2 > 150 \text{ m}^3\text{/s}] = 2/34 = 0.06$$

The large difference between both probabilities, illustrates the role of dependence for conditional probability.

# Summary

As we have seen in the preceding examples, dependence can have a significant role in the computation of probabilities. In particular, the examples illustrate very clearly that we should no longer be satisfied with the simplifying assumption of independence, and that for many problems $P(A \cap B) \neq P(A)P(B)$. But how can we compute it?

The following pages will explore methods to describe dependence quantitatively for continuous random variables.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Gaussian distribution

## Contents


- Definition of bivariate Gaussian distribution
- Conditionalizing a bivariate Gaussian distribution
- From 2D to 3D: multivariate margins
- Extra material: a video


One of the simplest approaches we have to define a multivariate distribution, $F(x, y)$, is through the multivariate Gaussian distribution. This model assumes that both the marginals and the dependence are Gaussian. As you saw in previous chapters, the Gaussian distribution is not always the best model so the applicability of the multivariate Gaussian is limited. However, it is a convenient model since it can be manipulated analytically and we can use it as a first approach to model the dependence.

## Definition of bivariate Gaussian distribution

The bivariate Gaussian distribution for two random variables $X_1$ and $X_2$ is defined as

$$\phi_\rho(x_1, x_2) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}} \exp\left(-\frac{\left(\frac{x_1-\mu_1}{\sigma_1}\right) - \left(\frac{2\rho(x_1-\mu_1)(x_2-\mu_2)}{\sigma_1\sigma_2}\right) + \left(\frac{x_2-\mu_2}{\sigma_2}\right)}{2(1-\rho^2)}\right)$$

where $x_1$ and $x_2$ are values of the random variables, $\sigma_1$ and $\sigma_2$ are the standard deviations of the random variables, $\rho$ is the correlation coefficient between $X_1$ and $X_2$, and $\mu_1$ and $\mu_2$ are the mean values of the random variables. Therefore, it has five parameters: $\mu_1$, $\mu_2$, $\sigma_1$, $\sigma_2$ and $\rho$.

We can rewrite the above expression in matricial form as

$$\phi_\rho(x_1, x_2) = \frac{1}{\sqrt{(2\pi)^2 \begin{vmatrix} \sigma_1^2 & Cov(X_1, X_2) \\ Cov(X_1, X_2) & \sigma_2^2 \end{vmatrix}}} \exp\left( (x_1 - \mu_1\, x_2 - \mu_2) \begin{pmatrix} \sigma_1^2 & Cov(X_1, X_2 \\ Cov(X_1, X_2) & \sigma \end{pmatrix}\right.$$

where $Cov(X_1, X_2)$ is the covariance of the random variables $X_1$ and $X_2$. We can also present the above form in a compressed fashion as

$$\phi_\rho(x_1, x_2) = \frac{1}{\sqrt{(2\pi)^2 |\boldsymbol{\Sigma}|}} \exp -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{\mu})$$

where $\boldsymbol{x}$ is the vector of values of the random variable, $\boldsymbol{\mu}$ is the vector of means, and $\boldsymbol{\Sigma}$ is the covariance matrix[1].

The cumulative distribution function of the bivariate Gaussian distribution would then be defined as

$$\Phi_{X_1,X_2}(x_1, x_2) = P(X_1 \leq x_1, X_2 \leq x_2) = \int_{-\infty}^{x_1} \int_{-\infty}^{x_2} \phi_\rho(s_1, s_2) ds_1 ds_2$$

Note that when talking about the Gaussian distribution instead of using $F_{X_1,X_2}(x_1, x_2)$, we use $\Phi_{X_1,X_2}(x_1, x_2)$, but it means the same!

In the figure below, you have the PDF and CDF of a bivariate Gaussian distribution for a correlation coefficient $\rho = 0.77$.

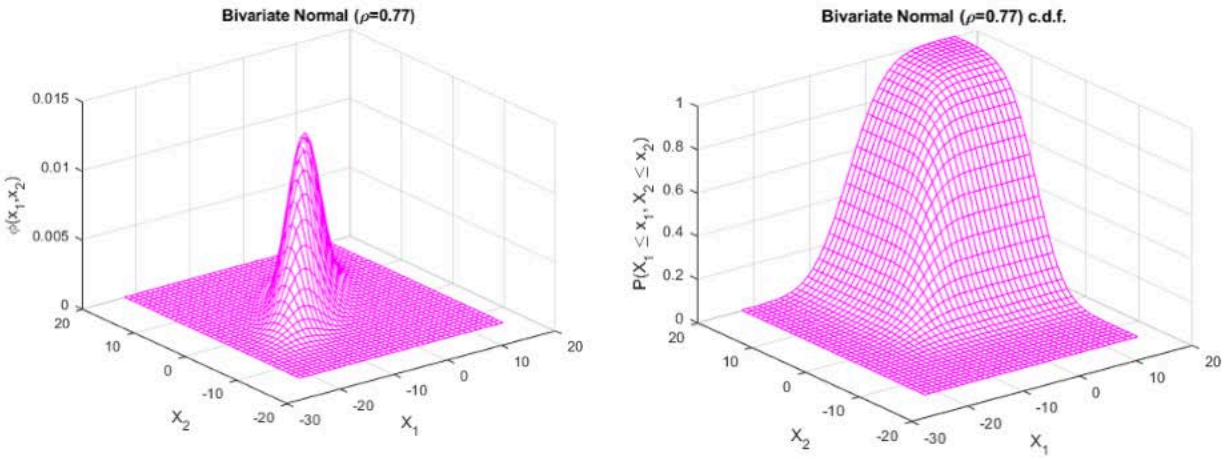



*Fig. 6.10* Bivariate Gaussian distribution: (left) probability density function, and (right) cumulative distribution function.

Also, you can plan with the interactive element below changing the correlation value yourself. Observe how the distribution's *density* contours, or a scatter plot of *samples,* change when you adjust the correlation.

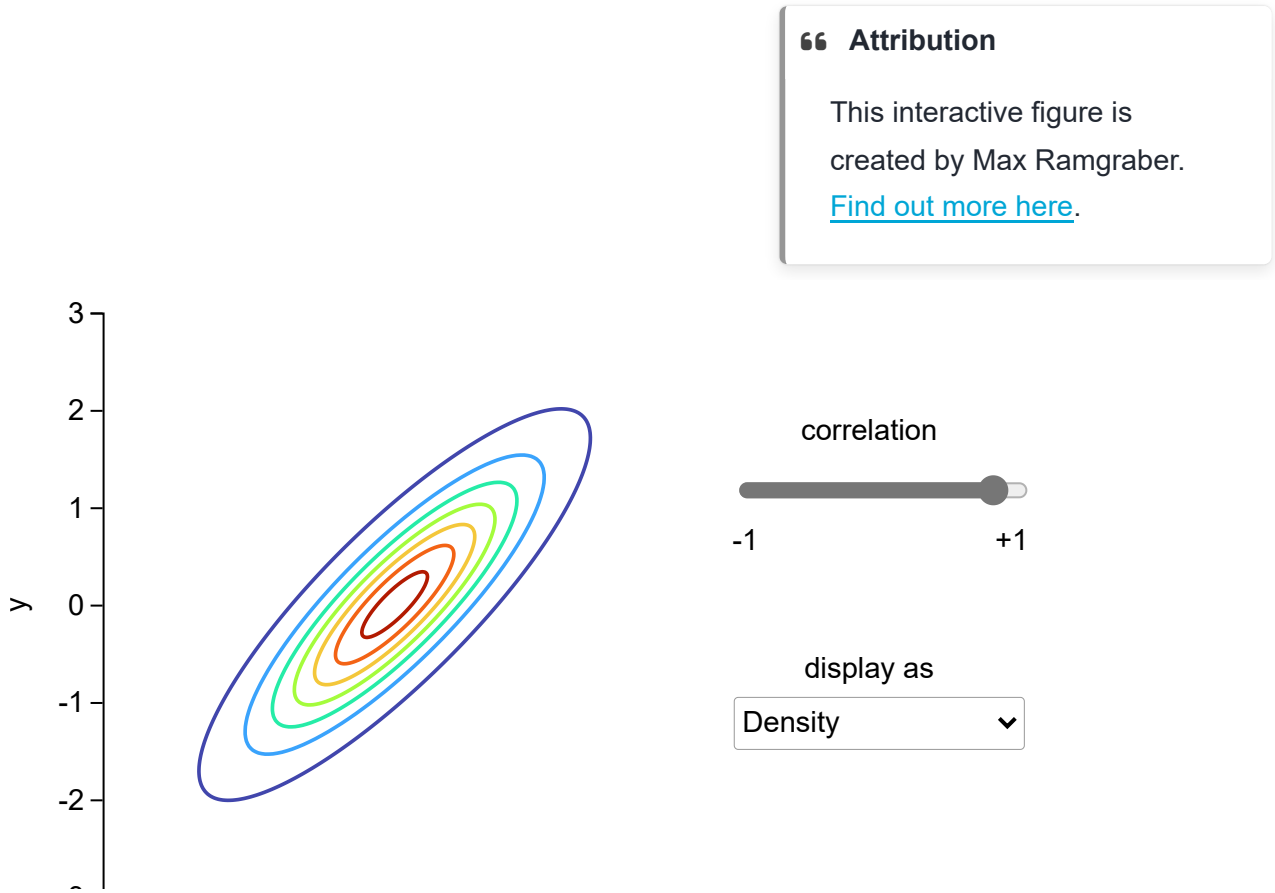



*Fig. 6.11* Interactively change the correlation coefficient to visualize the effect on density contours or samples of the bivariate Gaussian distribution.

# Conditionalizing a bivariate Gaussian distribution

Multivariate Gaussian distributions are useful because we can derive results analytically. Here, we are going to conditionalize a bivariate Gaussian distribution to exemplify it.

Given that we are modelling the joint probability distribution of $X_1$ and $X_2$ using a bivariate Gaussian distribution and we know $x_2 = a$, what is the expected distribution for $X_1$?

An important property of the multivariate Gaussian distribution is that if two sets of variables are jointly Gaussian, then the conditional distribution of one set conditioned on the other is again Gaussian. Thus, the conditional distribution of $Q_1$ given the value of $Q_2$ will be a Gaussian distribution whose mean value ($\hat{\mu}$) and standard deviation ($\hat{\Sigma}$) we need to estimate. This is, we want to compute $(x_1|x_2 = a) \sim N(\hat{\mu}, \hat{\Sigma})$. We can derive expressions

for $\hat{\mu}$ and $\hat{\Sigma}$ applying the definition of conditional density $\phi(x_1|x_2 = a) = \frac{f_{Y_1,Y_2}(y_1,a)}{f_{Y_2}(a)}$ knowing that $f_{Y_1,Y_2}$ is a bivariate Gaussian distribution and $f_{Y_2}$ is a univariate Gaussian distribution, replacing $x_2$ by $a$ and doing the (unpleasant) algebra. By doing so, we would reach the following expressions

$$\hat{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(a - \mu_2)$$

$$\hat{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

where $a$ is the known value of $X_2$ (here, $Q_2$) and $\mathbf{\Sigma} = \begin{pmatrix} \Sigma_{11} \, \Sigma_{12} \\ \Sigma_{21} \, \Sigma_{22} \end{pmatrix}$.

Let's go now back to the example of the discharges of two neighboring rivers, $Q_1$ and $Q_2$. We have historical measurements for both discharges and we want to apply a bivariate Gaussian distribution to model their joint distribution. Using the historical dataset, we can compute their mean values, $\mu_1 = 94m^3/s$ and $\mu_2 = 78m^3/s$, their standard deviations, $\sigma_1 = 41m^3/s$ and $\sigma_2 = 35m^3/s$, and the covariance between them, $Cov(Q_1, Q_2) = 1000(m^3/s)^2$. We know that $q_2 = 100m^3/s$. What is then the expected distribution for $Q_1$? This is, we want to compute $(q_1|q_2 = 100m^3/s) \sim N(\hat{\mu}, \hat{\Sigma})$.

We can summarize the above information as

$$\boldsymbol{\mu} = \begin{pmatrix} 94 \\ 78 \end{pmatrix}$$

$$\mathbf{\Sigma} = \begin{pmatrix} 41^2 \, 1000 \\ 1000 \, 35^2 \end{pmatrix}$$

Using the above expressions to our examples, we obtain

$$\hat{\mu} = 94 + 1000 \times (35^2)^{-1} \times (100 - 78) \approx 112.0$$

$$\hat{\Sigma} = 41^2 - 1000 \times (35^2)^{-1} \times 1000 \approx 864.7 \rightarrow \sigma_{x_1|x_2=a} = 29.4$$

The figure below displays the difference between the univariate distribution of $Q_1$ and the conditional distribution of $Q_1$ given $q_2 = 100m^3/s$.

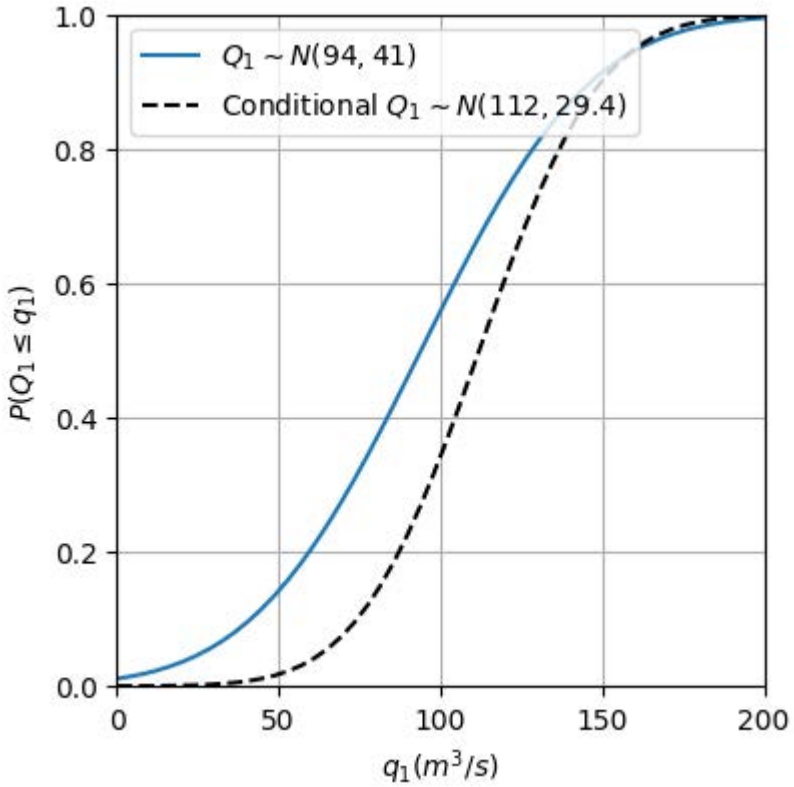



*Fig. 6.12* Distribution of $Q_1$ and conditional distribution of $Q_1$ given $Q_2$. .

# From 2D to 3D: multivariate margins

Often in the fields of Civil Engineering and Geosciences, we want to model more than two variables. Sometimes even tens or hundreds of variables. Thus, we need to find flexible models that account for the probabilistic dependence between them the best way possible. One option is to extend the bivariate Gaussian distribution to a multivariate Gaussian distribution with the desired number of random variables. Note that this implies that all the random variables are Gaussian-distributed.

We can extend the analytical equation to conditionalize the bivariate Gaussian distribution to the 3D multivariate Gaussian distribution to compute $(x_1, x_2 | x_3 = a) \sim N(\hat{\mu}, \hat{\Sigma})$ as

$$\hat{\mu} = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} + \begin{pmatrix} \Sigma_{13} \\ \Sigma_{23} \end{pmatrix} \Sigma_{33}^{-1} (a - \mu_3)$$

$$\hat{\Sigma} = \begin{pmatrix} \Sigma_{11} \Sigma_{12} \\ \Sigma_{21} \Sigma_{22} \end{pmatrix} - \begin{pmatrix} \Sigma_{13} \\ \Sigma_{23} \end{pmatrix} \Sigma_{33}^{-1} \begin{pmatrix} \Sigma_{13} & \Sigma_{23} \end{pmatrix}$$

Let's see an example with three dimensions. Imagine that we want to model the dependence between the precipitation, $P$, and the discharges of the two rivers, $Q_1$ and $Q_2$.

From a raingauge station, we could obtain the needed statistics of $P$ to model it using a Gaussian distribution, $\mu_P = 12mm/h$ and $\sigma_P = 27mm/h$, as well as the covariance with $Q_1$ and $Q_2$, $Cov(P, Q_1) = 475$ and $Cov(P, Q_2) = 520$. Assuming that we model the joint distribution of $P$, $Q_1$ and $Q_2$ using a multivariate Gaussian distribution, its parameters would be

$$\boldsymbol{\mu} = \begin{pmatrix} 94 \\ 78 \\ 12 \end{pmatrix}$$

$$\boldsymbol{\Sigma} = \begin{pmatrix} 41^2 & 1000 & 475 \\ 1000 & 35^2 & 520 \\ 475 & 520 & 27^2 \end{pmatrix}$$

Now we want to make use of the multivariate Gaussian distribution to see what is the distribution of the discharges in the river if today is raining $p = 22mm/h$. This is, we are going to conditionalize the 3D multivariate Gaussian on one variable, $P$, to obtain a conditional bivariate Gaussian distribution of $Q_1$ and $Q_2$. In mathematical terms, $(q_1, q_2|p = 22mm/h) \sim N(\hat{\mu}, \hat{\Sigma})$. We would do it as

$$\hat{\mu} = \begin{pmatrix} 94 \\ 78 \end{pmatrix} + \begin{pmatrix} 475 \\ 520 \end{pmatrix} (27^2)^{-1} (22 - 12) = \begin{pmatrix} 100.5 \\ 85.1 \end{pmatrix}$$

$$\hat{\Sigma} = \begin{pmatrix} 41^2 & 1000 \\ 1000 & 35^2 \end{pmatrix} - \begin{pmatrix} 475 \\ 520 \end{pmatrix} (27^2)^{-1} \begin{pmatrix} 475 & 520 \end{pmatrix} = \begin{pmatrix} 41^2 & 1000 \\ 1000 & 35^2 \end{pmatrix} - \begin{pmatrix} 309.5 & 338.8 \\ 338.8 & 370.9 \end{pmatrix} = \begin{pmatrix}$$

We can see that the means of the random variables $Q_1$ and $Q_2$ have increased while $Cov(Q_1, Q_2)$ has been reduced from 1000 to 661.2. The figure below displays the difference between the univariate distributions of $Q_1$ and $Q_2$ without and with conditionalizing.

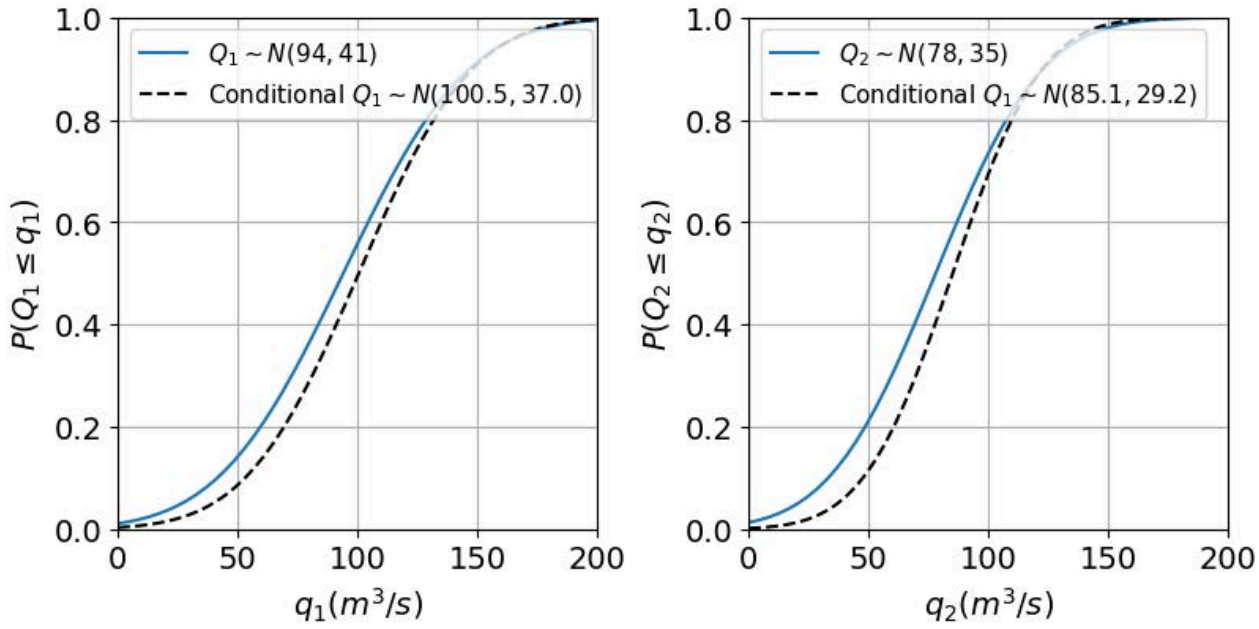



*Fig. 6.13* Unconditional and conditional Gaussian distributions given $P$: (left) $Q_1$, and (right) $Q_2$.

We can also compare the bivariate Gaussian distribution of $Q_1$ and $Q_2$ without and with the conditionalization, as shown in the Figure below. You can see how the mode of the distribution (point of maximum density) has moved towards the upper right side of the plot and become slightly narrower when conditionalizing.

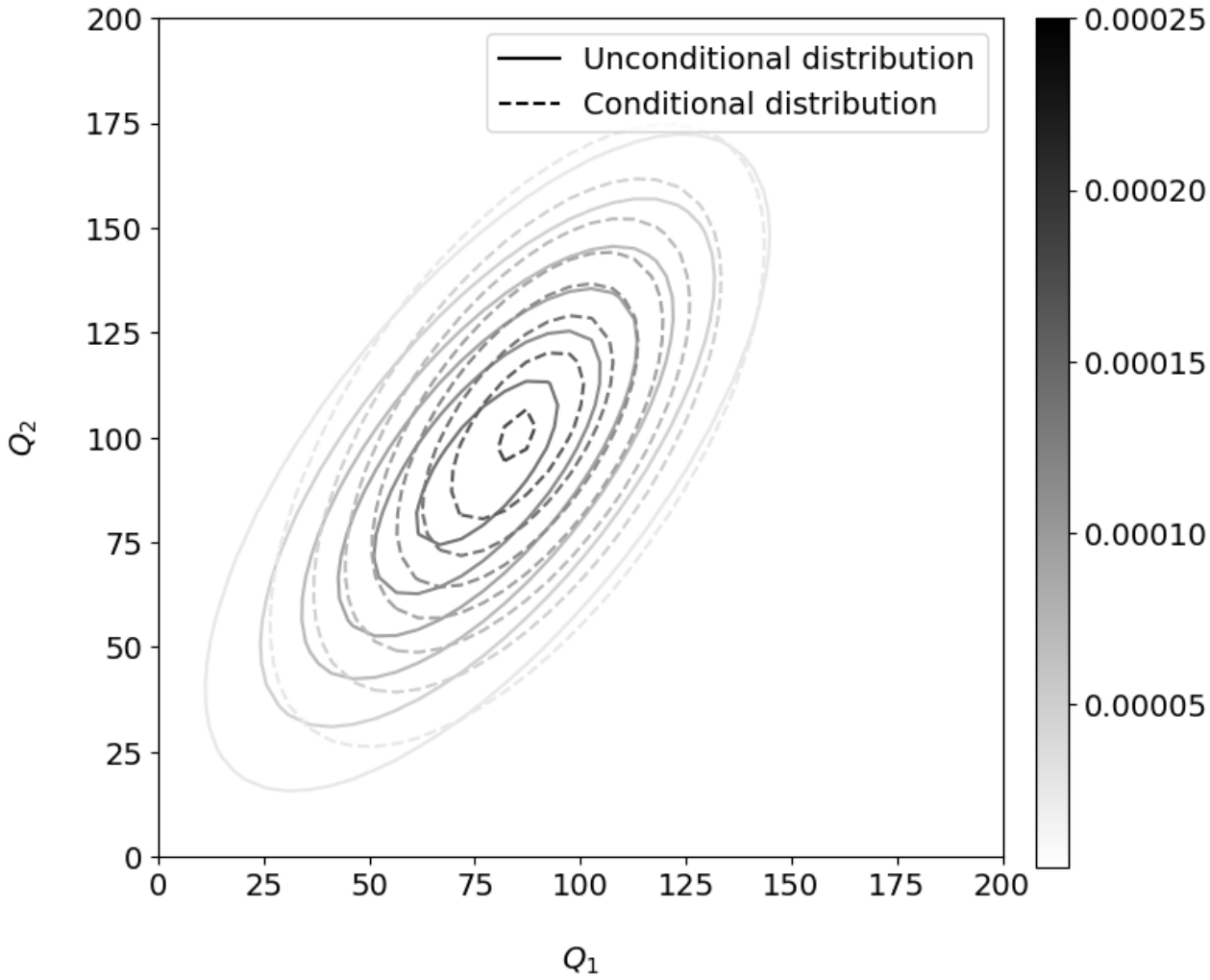



*Fig. 6.14* Unconditional and conditional bivariate Gaussian distributions for $Q_1$ and $Q_2$. Same density contours are plotted for both distributions.

# Extra material: a video

If you need to refresh the concept of covariance and correlation and want to see a short video on the multivariate Gaussian distribution, you have one here!

1_3_1a_Video

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

You already studied covariance matrices here.

# Non-Gaussian Distributions

## Contents


- Bivariate copulas


The multivariate Gaussian distribution $\Phi_X(x)$ is a simple, effective and widely used approach for capturing dependence in multivariate situations. However, for continuous random variables, it has limitations due to its inability to capture:

- non-Gaussian marginal distributions
- non-linear dependence

Fortunately there are many methods available that can take this into account, for example:

- Copulas
- Vine-Copulas
- Parametric and Non-Parametric Bayesian Networks

In this chapter we will focus on the use of a Copula, which provides a simple, yet incredibly flexible way to take into account non-Gaussian marginals *and* non-linear dependence. However, for simplicity we will stick to the Gaussian Copula, which considers *linear* dependence, just as the Multivariate Gaussian distribution does.

## Bivariate copulas

Bivariate copulas, or just copulas, are joint distributions with uniform marginal distributions in [0, 1]. According to Sklar theorem (not provided here), any multivariate joint distribution of continuous random variables can be described as a copula, $C$, that models the dependence between the variables, where the variables are defined by univariate marginal distributions.

**Definition of bivariate copula**

The definition of copula for the bivariate case is given by

$$F_{X_1,X_2}(x_1, x_2) = C\big[F_{X_1}(x_1), F_{X_2}(x_2)\big]$$

where $F_{X_1,X_2}(x_1, x_2)$ for $(x_1, x_2) \in \mathbb{R}^2$ is a joint distribution with marginals $F_{X_1}(x_1)$ and $F_{X_2}(x_2)$ in [0, 1] and $C$ is a copula in the unit square $I^2 = ([0, 1] \times [0, 1])$, being this equation satisfied for all $(x_1, x_2) \in \mathbb{R}^2$.

Therefore, the joint density is given as the product of the density of the copula, $c$, and the densities of the marginals as

$$f_{X_1,X_2}(x_1, x_2) = f_{X_1}(x_1) f_{X_2}(x_2) c(F_{X_1}(x_1), F_{X_2}(x_2))$$

**But what does that mean?**

When using copulas, we are going to model the univariate distributions for each variable and the dependence between them with different models. To do so, we select and fit a univariate distributions for each variable as seen in the previous chapter. That characterizes the univariate distribution of each variable or what call *marginal distributions*.

To deal with the dependence between the random variables using copulas, we transform the observations to what we call *unity space*, *uniform space* or *pseudo observations* by means of the cumulative distribution function. This is, we will model the dependence between the non-exceedance probabilities of the random variables and not between the values of the random variables. We can perform this transformation using either the empirical cumulative distribution function or the parametric distribution functions we have fitted.

In the figure below, in panel (a), you have observations of discharges of two rivers, $Q_1$ and $Q_2$. In panel (b) you see how we have transformed the observations to uniform space. Note that the marginal distributions in panel (b) are uniform and defined in [0,1].

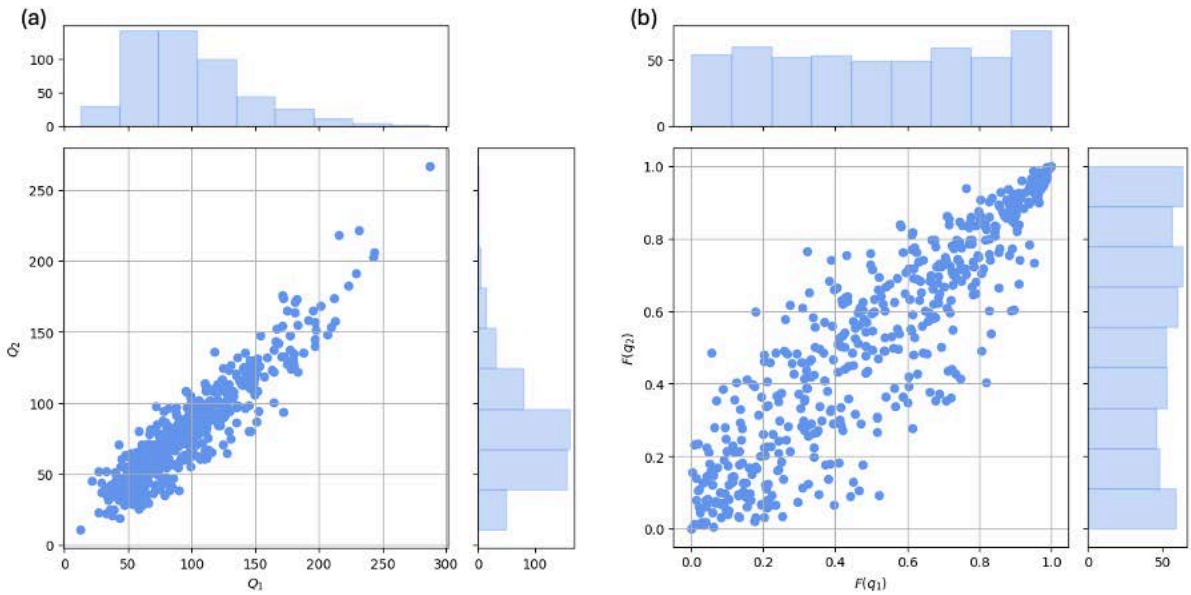



*Fig. 6.15* Scatter plot and histograms of the observations of $Q_1$ and $Q_2$: (a) in variable space, and (b) in uniform space.

This approach has several advantages when modelling joint multivariate distributions, such as:

- It improves the flexibility of the model, as we can choose different marginal distributions for each random variable, and also different to the model for the dependence.
- It helps in visualizing the dependence between the variable without the influence of the marginal distributions. This is, if you go back to the figure above and compare the shape of the scatter plot between $Q_1$ and $Q_2$, and $F_{Q_1}(q_1)$ and $F_{Q_2}(q_2)$, you can see significant differences. In panel (b), you can see how the high values of $F_{Q_1}(q_1)$ and $F_{Q_2}(q_2)$ are more correlated (the scatter is narrower) than the lower values (larger scatter). That is difficult to observe in panel (a).

When we were studying univariate distributions, we fitted parametric univariate distributions (e.g. Gaussian or Gumbel) to the empirical distribution (the observations). Here, we can do the same with copulas: we can fit a parametric copula to the observations. There are different copula families such as Gaussian, Clayton, or Gumbel, with different shapes. For instance, Gaussian copula is symmetric, while Clayton copula is not. As you can see in the Figure below, the lower values are more correlated than the higher values in Clayton copula. During the course, you will only use Gaussian copula.

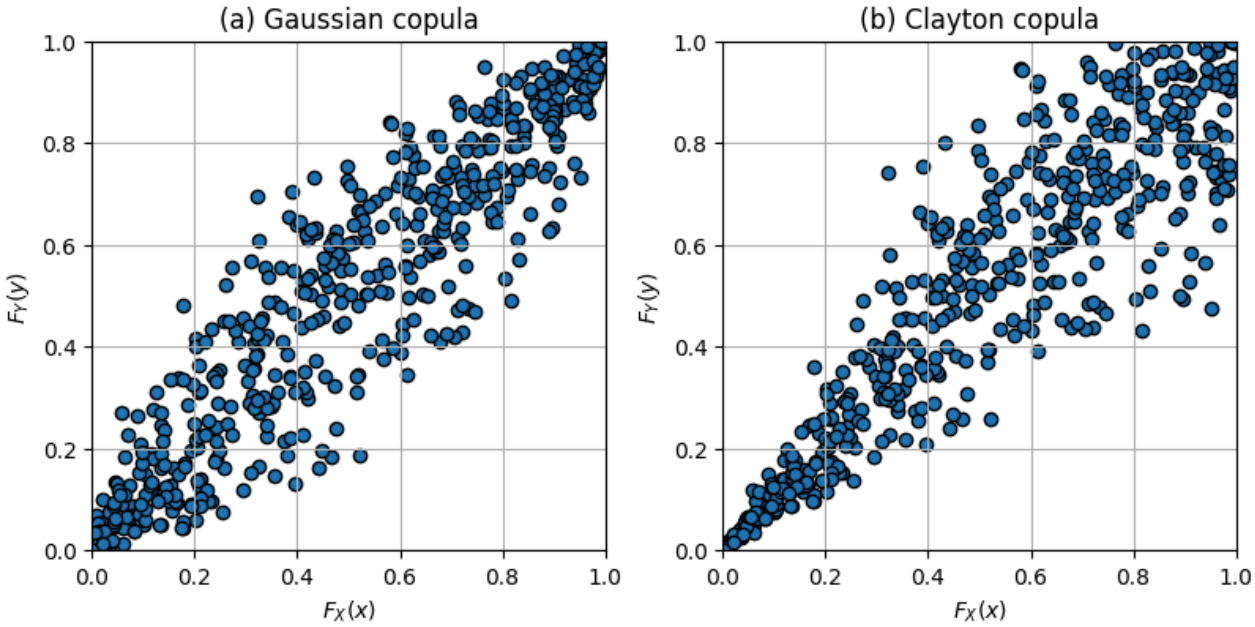



*Fig. 6.16* Scatter plot and histograms of the observations of $Q_1$ and $Q_2$: (a) in variable space, and (b) in uniform space.

**Exam tip**

The goal of this course is just to give you a first brief introduction to copulas. Thus, you just need to understand the concept of copula and how to compute the joint density.

You will be provided Python code to evaluate multivariate distributions constructed using Copulas.

If you want to learn more about bivariate copulas and other multivariate joint distributions in higher dimensions, you can study them in the Cross Over "MORE: Probabilistic Modelling of real-world phenomena through ObseRvations and Elicitation".

**Attribution**

This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Designing with Probability

## Contents


- Discrete Events
- Functions of Random Variables
- Next Steps


This chapter is closed with an illustration of how our probability models can be used in practice, specifically in the **design** of some object or system. As engineers making decisions under uncertain conditions (i.e., the unknown values of our random variables) we face a serious conundrum[1]: our design must specify a deterministic value, which will eventually be used to build the object or system of interest. In the face of uncertainty, this means there will *always* be a non-zero probability that the object or system can fail! Thus, the choice is not simply how robust to design somthing, but rather: how safe should it be designed. In other words, *what is an acceptable probability of failure for the design?*

This question can be answered by applying the concepts of risk and risk analysis, which will be covered in a later chapter. For now, we will assume that an *acceptable failure probability* has been determined and use our probability models to evaluate what the *design value* should be.

## Discrete Events

Previous sections in this chapter introduced multivariate distributions and, specifically, a few approaches for incorporating *dependence* in our probabilistic assessment of more than one random variable of interest. Until this point, examples have been illustrated using a simple paradigm: “AND” and “OR” probabilities (analogous to intersection and union in set theory), where each random variable is assumed to have a critical or threshold case with some probability of exceedance, p, for example (if $X_1$ has the lognormal distribution):

$$X_1 \sim \text{LN}(\mu, \sigma) \quad \rightarrow \quad P[X_1 \leq x_{threshold}] = 1 - F_{X_1}(x_{threshold})$$

However, for the two random variable case, this limits us to only a few relevant design situations: the various combinations where each variable is either greater than or less than a

threshold value. This is very restrictive in practice. Fortunately we already have a tool at hand for evaluating more complex situations: the concept of **functions of random variables**!

# Functions of Random Variables

Given a vector of random variables $X$ and function $q$, the output, $Y$, is a function of random variables:

$$Y = q(X)$$

While $Y$ is itself a random variable, its distribution is known only for a few special cases, for example, if $q$ is linear and $X$ is the multivariate Gaussian, then $Y$ is also Gaussian. This is no longer true if the function is non-linear or the marginal distributions of the random variables are non-Gaussian. Luckily we are usually able to empirically find the distribution of $Y$ using *Monte Carlo Sampling.*

It is important also to note that although the distribution of $Y$ is univariate, it is related to the underlying multivariate distribution of $X$ via the function $q$. As such, the model chosen for $f_X(x)$ will have an impact on the distribution of $Y$.

> **Tip**
>
> In some fields, for example structural reliability and related branches of civil engineering, it is common to formulate the function of random variables as a *limit state function,* where the *threshold* value of the function dividing a *safe* and *failed* state is found. The function is then reformulated as a new function, $g$ with output $Z$ is defined such that 0 defines the threshold state and negative values the failed state. The probability of being in the failed state, $p_f$, is found as follows:
>
> $$Z = g(X) \quad \rightarrow \quad p_f = P[Z \leq 0]$$

Regardless of the underlying multivariate distribution of the input random variables, the task at hand is to compute the design value, $y_{design}$, for a given probability of interest, $p_{design}$:

$$y_{design} = F_Y^{-1}(p_{design})$$

where $F_Y^{-1}$ is the inverse CDF of the distribution of $Y$. Although we rarely know the exact distribution of $Y$, it is straightforward to compute the design value using Monte Carlo Simulation.

# Next Steps

The following pages work through an example related to flooding on a river to illustrate the role that the random variable inputs of a function of random variables have on . Specifically, the following pages will:

1. Consider a univariate design case for the discharge capacity of a flood protection system on a river.
2. Consider a bivariate design case where two rivers contribute to flooding and make the distinction between AND and OR perspectives explicit.

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

---

a confusing and difficult problem or question (source: Oxford Languages, via Google).

# One Random Variable

> **Attribution**
>
> This page reuses material from another book. Find out more here.

Consider a river that is protected by dikes: earthen embankments with the purpose of keeping water in the river channel and preventing flooding of the hinterland. The dikes should be designed and built such that the height allows the river to safely pass the maximum discharge every year, measured in $m^3/s$. A rating curve (Fig. 5.24) gives us the relationship between discharge and water depth, $H_w$, so if we know what the maximum discharge is, we can build our dikes to the critical height, $h_{dike} \geq H_w$. But there is a problem—what is the maximum discharge in the river each year? How would we even go about determining this value? One way is to make observations and find the highest recorded data point.

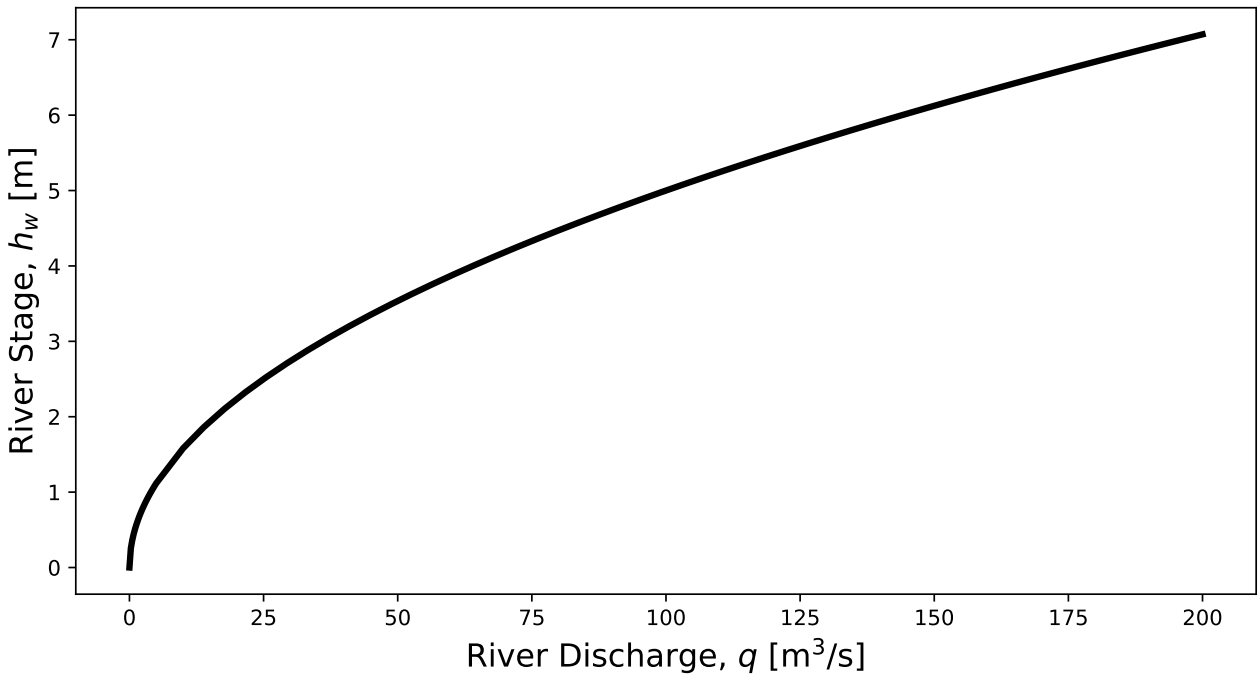



*Fig. 6.17* Rating curve for hypothetical river. Relationship is $Q = C \cdot H_w^{\beta}$ where $\beta$=2.0 and $C$ =4.0.

**Rhine River Example**

In 1995 the Rhine River reached a discharge of around 12,000 $m^3/s$, the highest recorded level to date. But even with over 100 years of measurements, the record is not complete: there could always be a bigger discharge. One approach could be to come up with a number that is, beyond a reasonable doubt, impossible to exceed. Many experts believe that 16,000 $m^3/s$ is getting close to an upper limit, and that 20,000 $m^3/s$ may be physically impossible.

*Which of these two values should we choose for the design of our dike, and why?*

**Click here for explanation**

There is no correct answer here. In an ideal world we would design for all possible scenarios, but this is often prohibitively expensive to do in practice. The question is thus what level of protection is reasonable given the possible consequences of flooding? It makes sense to protect against the more frequent flood events, and if there are a lot of people or economic resources in the flooded area, it would also make sense to choose a higher protection level, if financial resources are available. In other words, the *decision* is a function of the *probability* of the event occurring and the *consequences.*

The choice of a suitable protection level is difficult question because the actual discharge observed in a given year is random. In addition to other variables, it is largely dependent on rainfall in the drainage basin, which is impossible to predict far in advance—certainly not for the entire year. Furthermore, we don't have a way to accurately extrapolate our past observations to identify a logical maximum discharge. For this reason, we will use a continuous probability distribution to quantify the probability associated with a specific maximum discharge, $q$, that might be observed in a given year, defined by the PDF $f_Q(q)$ and CDF $F_Q(q)$.

**Tip**

The distribution in the example is Lognormal; however, we leave out the details of the distribution and its parameters here in order to focus on the exceedance probability and design value.

The only other piece of information needed is a design standard—a probability that defines the discharge to use for our design. Other chapters in this book cover the derivation of a design standard, so for this discussion we will assume that a flooding probability of 1/100 has already been determined as the acceptable level. Because there is only one random variable, we can use the CDF to determine the design discharge (Fig. 6.18).

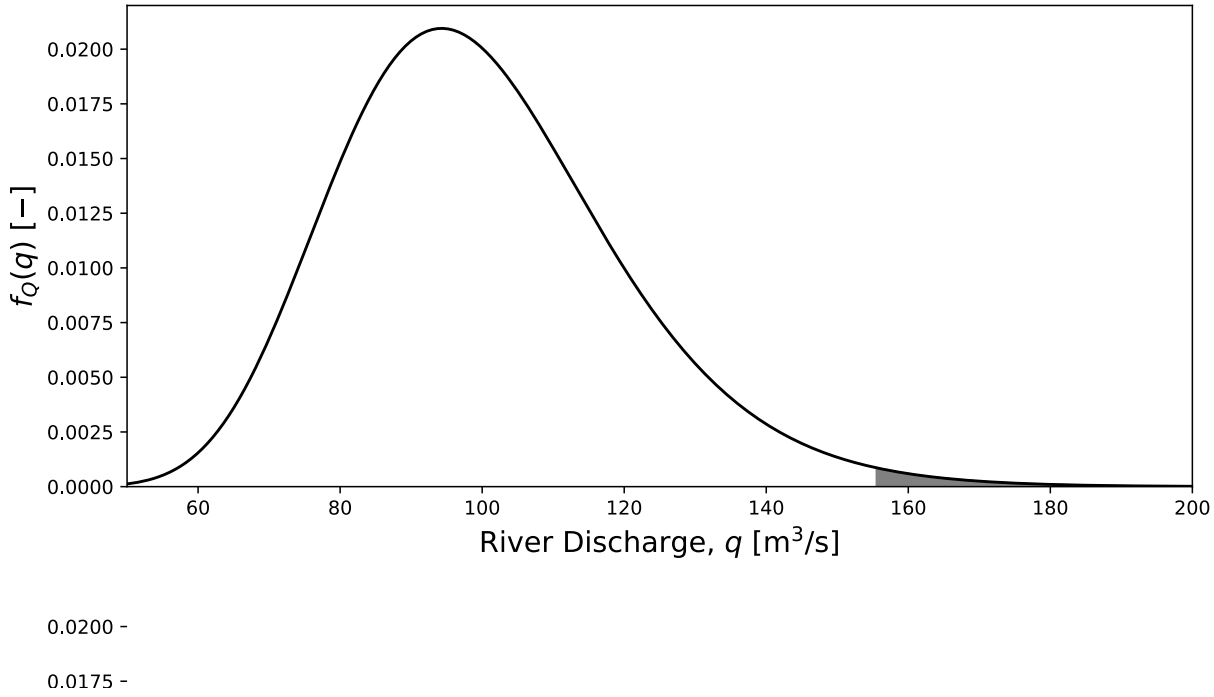



*Fig. 6.18* PDF and design value for river discharge with 0.01 probability of exceedance: $q_{design} = 155$ m$^3$/s.

In particular, we note that failure occurs for high discharge, so we must use the complementary CDF to find the design value, in this case probability of exceeding $q_{\text{design}}$:

$$1 - F_Q(q_{\text{design}}) = 0.01$$

$$q_{\text{design}} = F_Q^{-1}(1 - 0.01) = 155 \text{ m}^3/\text{s}$$

after which the water depth and minimum dike height can easily be determined from the rating curve:

$$h_{dike} = h_{w,design} = \left(\frac{q_{design}}{C}\right)^{1/\beta} = 6.23\text{m}$$

**Reflection on the Simple Example**

Is this an oversimplification of reality? Perhaps, but it gets pretty close. Many flood protection systems in the world use an exceedance proabability approach to probabilistic design: choosing a specific and making sure the component or system can handle the load. This works well when the primary source of uncertainty is the load: the river discharge in this case.

We also took for granted that a simple relationship between river discharge and water depth is available. However, this is never the case, as such a relationship depends on the cross-sectional shape and roughness of the floodplain, which not only varies along the river trajectory, but also changes due to natural phenomenon and human interventions. This introduces additional uncertainty into our assessment of whether the dikes are high enough.

Fortunately there is also extra conservatism built into this approach. For example, duration of high water plays a role: if discharge exceeds the capacity of the dike system, but only lasts for a short time (minutes or a couple hours), perhaps the dikes can withstand the overflow without eroding and causing flooding. This can be conceptualized by considering the joint probability of a high discharge *and* degradation of the dike leading to flooding:

$$P[\text{flooding}, h_w > h_{dike}] = P[\text{flooding}|h_w > h_{dike}] \cdot P[h_w > h_{dike}]$$

where the conditional term represents the probability that the dike erodes (fails) given that the water depth exceeds the height. Assuming failure when the critical water depth is exceeded is conservative because it implies $P[\text{flooding}|h_w > h_{dike}] = 1.0$, whereas in reality this is not the case, and the 'true' flooding probability is less than that computed in the equation above.

Later chapters of this book will introduce methods for taking these realities into account in the design and decision-making process. For now, we will see how our probabilistic design changes when more than one random variable must be considered.

# Two Random Variables

## Contents


- Discharge from Two Rivers
- Reflections on the Simple Example


> **Attribution**
>
> This page reuses material from another book. Find out more here.

The previous section illustrated how a decision for dike height, $h_{dike}$, can take into account uncertainty in river discharge. We used a maximum allowable probability of flooding to derive the specific value of discharge, $q_{design}$, that should be used to choose $h_{dike}$. This was necessary because there was otherwise no way of establishing what the maximum discharge should be. The only variable considered to be random was the maximum river discharge that occurs each year. This section considers how the design process becomes more complicated when *two* random variables are considered (two load variables).

**Distribution for $Q_1$ and $Q_2$**

On this page all calculations assume the same lognormal distribution for random variables $Q_1$ and $Q_2$, where the first and second moments of the distribution are:

$$E[X] = \exp\left(\mu + \frac{\sigma^2}{2}\right) = 100$$
$$VAR[X] = \left(\exp(\sigma^2) - 1\right)\exp(2\mu + \sigma^2) = 400$$

See the table in Summary of parametric distribibutions

For the `lognorm` method in `scipy.stats` we need to provide the parameters shape `s` = $\sigma$, `loc` = 0 and `scale` = $\exp(\mu)$ (see explanation here), in this case:

```
import scipy.stats as stats
s = 0.198
loc = 0.000
scale = 98.058
q_1 = stats.lognorm(s=s, loc=loc, scale=scale)
q_2 = stats.lognorm(s=s, loc=loc, scale=scale)
```

# Discharge from Two Rivers

In this scenario our objective for choosing $h_{dike}$ is still the same, except now we recognize that our location on the river is downstream of a confluence of two smaller rivers. The discharge at the location of our dike is thus the sum of the dicharge from Rivers 1 and 2:

$$Q = Q_1 + Q_2$$

where the same rating curve applies, giving $H_w$ as a function of $Q$. As before, acknowledging that the maximum discharge per year in each river is a random variable, we can approach the design problem in the same way: choose $h_{dike}$ such that it retains water when $Q$ takes a value with probability of exceedance 0.01. For now, we can assume the distribution for River 1 and River 2 are the same as in the previous Section, and that they are independent.

*Compared to the previous example, now there are two distributions—how do we find the design discharge and dike height?*

Before presenting the correct approach, three common mistakes are illustrated.

# Incorrect Approach 1

Find the discharge in each river, $Q_1$ and $q_2$, such that the probability of exceeding each discharge is 0.01. As in the previous section,

$$q_{1,\text{design}} = q_{2,\text{design}} = F_Q^{-1}(1-0.01) = 155\ \text{m}^3/\text{s} \ \rightarrow \ q_{design} = 310\ \text{m}^3/\text{s}$$

*Why is this approach incorrect?*

It considers the simultaneous occurrence of *both* rivers exceeding a design discharge with probability 0.01. This is a joint probability of occurrence, $P[Q_1 > q_{1,design}, Q_2 > q_{2,design}]$, which can be evaluated with a multivariate probability distribution (or the marginal distributions in the independent case). The scenario is equivalent to observing two sixes after tossing two dice simultaneously: if the dice are fair, the probability is $1/36$. Thus, for this incorrect case the probability $P[Q_1 > q_{1,design}, Q_2 > q_{2,design}]$ is not 0.01, it's actualy $0.01^2$=0.0001, illustrated in the figure below.

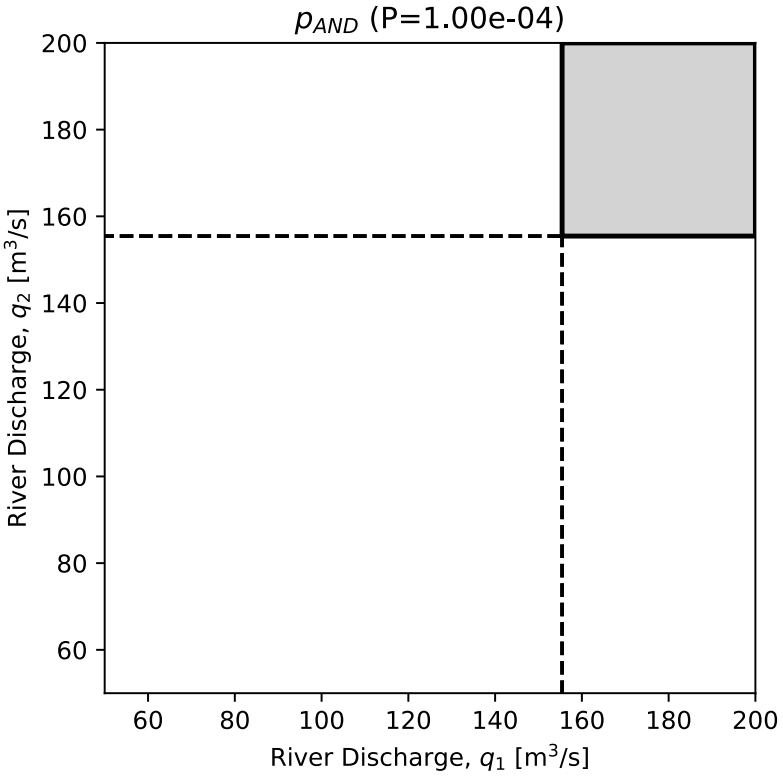



*Fig. 6.19* 'And' probability for Incorrect Approach 1.

# Incorrect Approach 2

Find the discharge in each river, $q_1$ and $q_2$, such that the joint probability of exceeding these values is 0.01. Since the two rivers are independent, we can solve for our design probability,

$p_{design}$:

$$(p_{design})^2 = 0.01 \;\rightarrow\; p_{design} = \sqrt{0.01} = 0.1$$

$$q_{1,\text{design}} = q_{2,\text{design}} = F_Q^{-1}(1-0.1) = 126\ \text{m}^3/\text{s} \;\rightarrow\; q_{design} = 252\ \text{m}^3/\text{s}$$

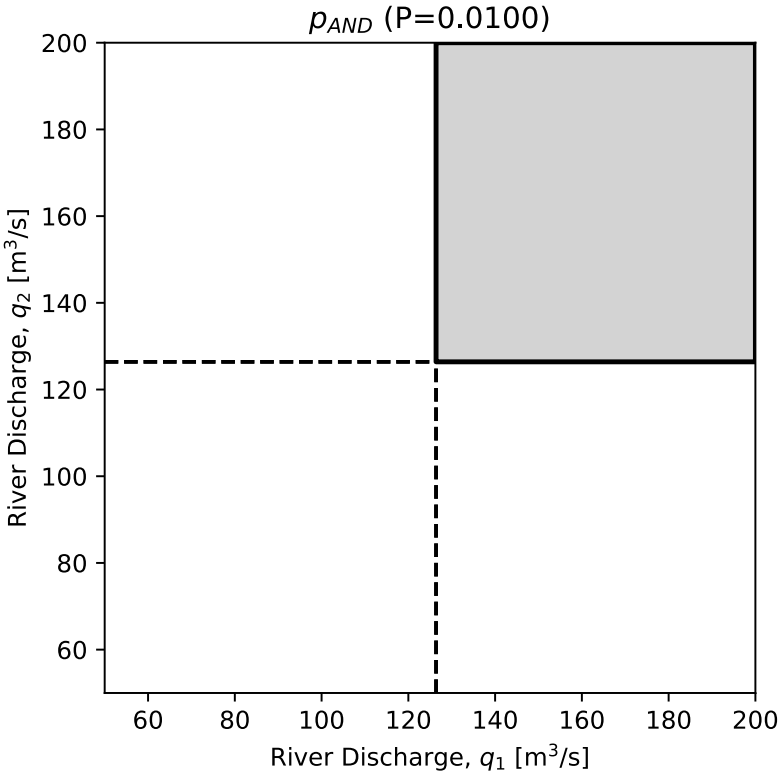



*Fig. 6.20* 'And' probability for Incorrect Approach 2.

*Why is this approach incorrect?*

It only considers one scenario!

There are infinite alternative scenarios that also result in a joint probability of exceedance of 0.01, for example, $0.2 \cdot 0.05 = 0.01$:

$$q_1 = F_Q^{-1}(1-0.20) = 116\ \text{m}^3/\text{s} \;\text{ and }\; q_2 = F_Q^{-1}(1-0.05) = 136\ \text{m}^3/\text{s}$$

which results in a totally different dike height.

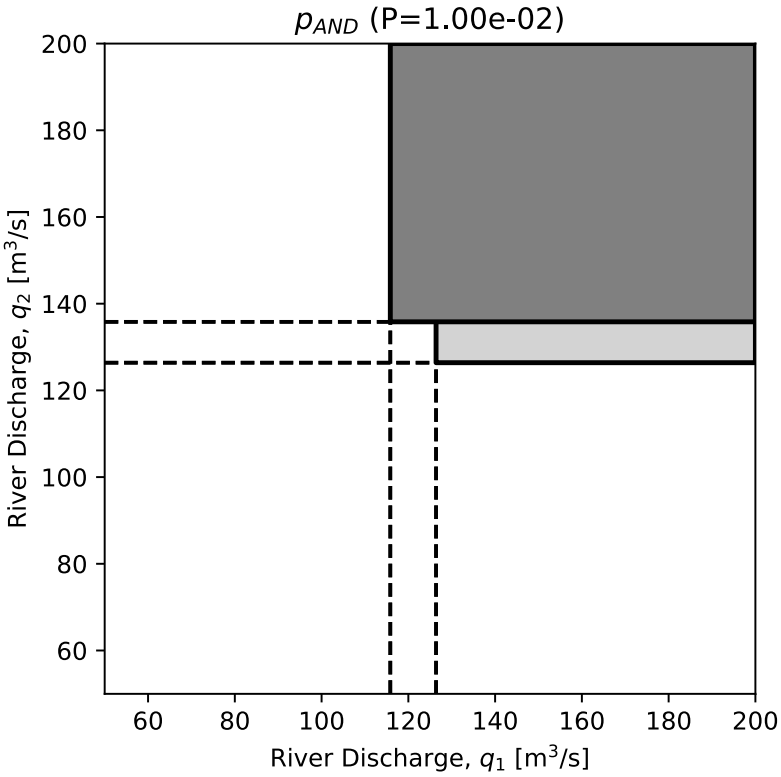



*Fig. 6.21* Comparison of two 'and' probabilities for Incorrect Approach 2.

# Incorrect Approach 3

Since the 'and' case didn't work we can use the 'or' probability, or union! In other words: find the discharge in each river, $q_1$ and $q_2$, such that the probability of either of them being exceeded is 0.01.

The union probability can be found as follows:

$$P[Q_1 > q_1 \cup Q_2 > q_2] = 1 - P[Q_1 < q_1, Q_2 < q_2]$$

given the assumtion of independence, the union probability is:

$$P[Q_1 > q_1 \cup Q_2 > q_2] = 1 - P[Q_1 < q_1] \cdot P[Q_2 < q_2]$$

Solving for the case where $P[Q_1 > q_1 \cup Q_2 > q_2] = 0.01$ implies the exceedance probability for each river discharge is $\sqrt{1 - 0.01} = \sqrt{0.99} = 0.995$, thus:

$$q_{1,\text{design}} = q_{2,\text{design}} = F_Q^{-1}(0.995) = 163 \text{ m}^3\text{/s} \ \rightarrow \ q_{design} = 326 \text{ m}^3\text{/s}$$

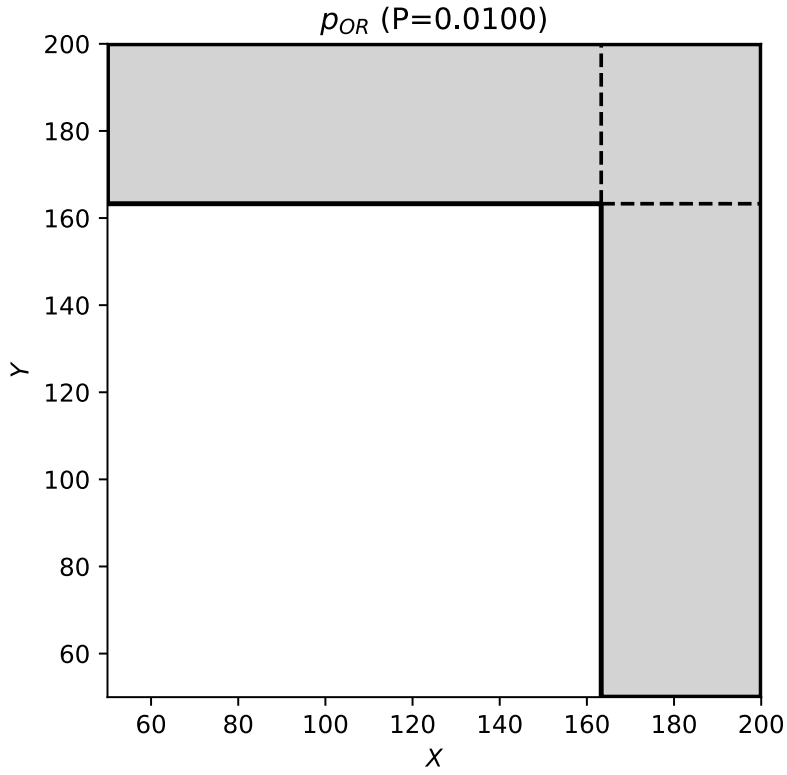



*Fig. 6.22* 'Or' probability for Incorrect Approach 3.

*Why is this approach incorrect?*

As with the first two incorrect approaches, it also only considers one specific scenario! There are infinite combinations of $Q_1$ and $Q_2$ such that the union is 0.01.

# Correct Approach

The discharge in the main river is the sum of the discharge of its tributaries, thus it is a function of random variables which has a distribution $f_Q(q)$. Thus we can use the distribution of $Q$ to find the design value $q_{design}$ directly. Unfortunately, although $q$ is a linear combination of $q_1$ and $q_2$, because they are lognormally distributed we cannot find a simple analytic expression for the distribution of $q$. However, using Monte Carlo sampling from the distributions of $Q_1$ and $Q_2$ gives:

$$q_{design} = F_Q^{-1}(1 - 0.01) = 275 \text{ m}^3\text{/s}$$

This is equivalent to integrating the joint probability distribution of $Q_1$ and $Q_2$ over the region $\Omega$ where $q > 275$ m$^3$/s, illustrated in the figure below, along with the contours of joint probability density.

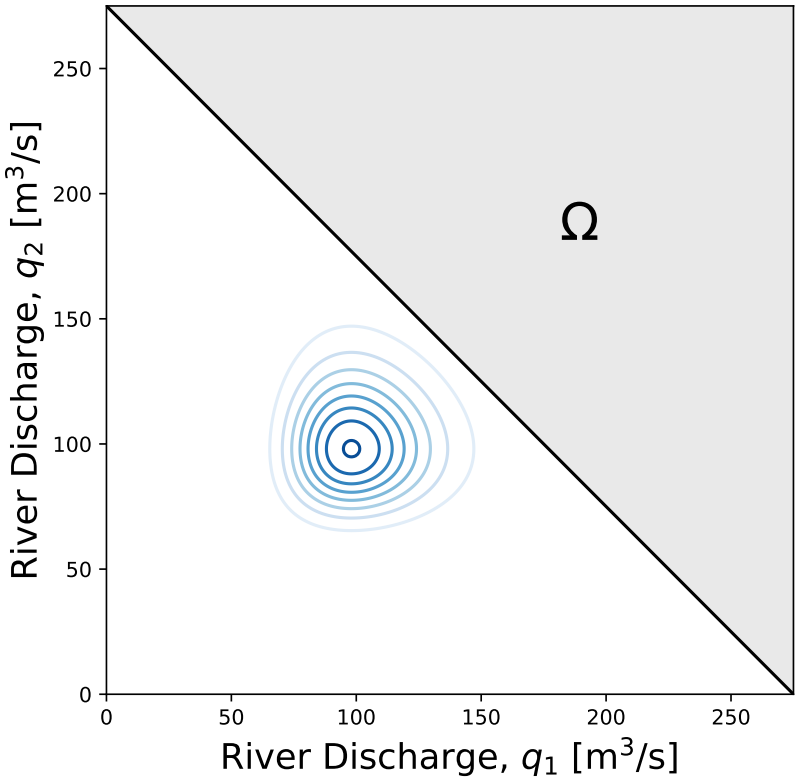



*Fig. 6.23* Correct approach: $P(q > 275\ \text{m}^3/\text{s}) = 0.01$. Also illustrated is the region $\Omega$ where $q > 275\ \text{m}^3/\text{s}$ and contours of joint probability density.

# Comparison of Approaches

All four design values of $Q$ are collected in the table below, along with the required dike height. Clearly the method of incorporating uncertainty makes a difference, as the height varies over 1 m across all options, which is a significant increase in cost for a structure that is many kilometers long.

*Table 6.1* Comparison of design values for all four approaches.

| **Approach** | **Design Value, $q_{design}$ [m$^3$/s]** | **Design Value, $h_{dike}$ [m]** |
|---|---|---|
| Incorrect 1 | 310 | 8.80 |
| Incorrect 2 | 252 | 7.94 |
| Incorrect 3 | 326 | 9.03 |
| **Correct** | **275** | 8.29 |

# Reflections on the Simple Example

We need to be explicit in what we are trying to evaluate. In this case, it is the distribution of a function of random variables, $Q$.

For non-Gaussian distributions and for non-linear functions of random variables it is often not possible to get the distribution of the function of random variables analytically. Fortunately it is easy to find it numerically through sampling techniques. This will be discussed elsewhere.

The ‘and’ and ‘or’ approaches (intersection and union) are simple, but really only limited to discrete cases. As illustrated above, they cannot be used in the case of continuous random variables, or with functions of random variables because one must integrate across all possible scenarios leading to failure. Nonetheless, these approaches are extremely useful for many situations, and are the basis of *system reliability* problems.

The situation illustrated here is often referred to as a *component reliability* problem, where the ‘component’ is defined by a function of random variables. Although nothing more than a function of random variables,is nothing more than an integration over a specific region of a multivariate probability density function. Often this region describes failure of a component, which we will try to keep below an acceptable level.

# PDEs and the Finite Volume Method

> **Attribution**
>
> This chapter is written by Robert Lanzafame and Jaime Arriaga Garcia. Find out more here.

The Finite Volume Method (FVM) is a numerical approach for solving partial differential equations (PDEs); it builds on many of the concepts covered by the Finite Difference Method (FDM), covered earlier in this textbook. The method evaluates a quantity of interest in space and time, $\phi(\mathbf{x}, t)$, over a domain of interest, $\Omega$. One main advantage of the finite volume method is that it is well-suited to flow-based problems (e.g., computational fluid dynamics), but it is also a relatively simple method that can be applied to a wide variety of physics-based conservation laws (e.g., mass, momentum and energy).

The general scheme for FV problems is as follows:

1. Convert the PDE to integral formulation, over a volume of interest
2. Apply Gauss's Theorem on the bounding surface
3. Discretize domain into finite volumes and apply the integral equations (spatial discretization)
4. Apply a numerical scheme to discretize equations in time, then solve

These topics will be discussed in more detail after the discretizing of a general problem into finite volumes is introduced below.

**Why is FVM useful?**

One of the reasons FVM is useful is because it uses a Cartesian-based reference frame that stays fixed in time to describe the time-evolution of the quantity of interest at discretized locations within a domain. Taking fluid flow as an example, FVM can be thought of as a series of containers (the "volumes"), each of which tracks the amount of fluid entering and leaving (fluxes), as well as a number of other changes in time within the container, for example, velocity or pressure.

Other numerical modelling methods for solving PDEs (e.g., finite difference or finite element methods) are often formulated to track a quantity of interest with a reference point that moves in time, such as deformation of a solid. This type of approach is generally inconvenient for fluid dynamics problems because of the complex geometry and motion that a fluid can take; for example, imagine describing the motion of a fluid subject to turbulence within a centrifugal pump. This topic is further covered in fvm_frames, but a formal derivation is generally outside the scope of this textbook.

To give an idea for how FVM is used in various civil and environmental engineering and geosciences applications, a few examples are provided here:

- **Fluid mechanics and dynamics**: simulating the flow in rivers and reservoirs, predicting flood propagation, analyzing groundwater flow, pipe and open-channel flows and conduits. FVM allows the discretization of complex spaces to estimate properties such as velocity, pressure and even concentration.
- **Heat transfer**: conduction through different materials, optimizing heating and cooling systems in buildings and analyzing the thermal behavior of structures.
- **Contaminant or pollutant transport**: simulate the impact of pollutants in the atmosphere or water bodies. This is crucial for designing pollution control systems and assessing contamination.
- **Structural analysis**: although less common, FVM can be adapted for stress and deformation in solid bodies. This technique is particularly useful in geotechnical engineering and civil infrastructure where large deformations can occur.
- **Multi-physics problems**: FVM can handle multi-physics simulations by combining fluid flow, heat transfer, chemical reactions, stress analysis, chemical transport and other processes in a single model. For example, FVM can analyze the thermal behavior of a dam holding back sediment-laden water.

# Getting Started

## Contents


- Notation
- Conservation Laws
- PDE's nature


This is a very quick overview of some fundamental knowledge that is helpful for understanding the analytic and numeric work needed for FVM. Some of it is calculus and continuum mechanics that should be mostly review (and can be used as a reference for notation), whereas the later sections give some context and insight into the derivation of commonly used conservation laws (PDE's) that can be solved using FVM. A few selected LaTeX equations are provided as well to help you use the same notation in your Markdown-formatted reports (optional).

## Notation

A generic scalar quantity $\phi$ is typically defined in 3-dimensional (3D) space and time, $\phi(x, y, z, t)$.

Bold straight lower-case letters are used for vector quantities. For example, the Cartesian coordinate system is:

$$\mathbf{x} = x\,\hat{i} + y\,\hat{j} + z\,\hat{k}$$

The scalar quantity defined above could thus be specified $\phi(\mathbf{x}, t)$. Velocity is denoted $\mathbf{u}$, with components $u$, $v$ and $w$:

$$\mathbf{u} = u\,\hat{i} + v\,\hat{j} + w\,\hat{k}$$

In the finite volume discretization scheme, volume indices are denoted with a subscript that refers to increments of each axis in $\mathbf{x}$, starting at index 1 (not 0, as in Python!). For example, subscript $i$, $j$ are used in 2D, such that the volume centered as $x$-coordinate 4 and $y$-coordinate 6 would be described with velocity $\mathbf{u}^n_{4,6}$. The time increment is identified with a

superscript, $n$; the initial condition starts at index 0. In other words, the initial condition is specified at time $t_0$.

Locations or quantities in a 1-dimensional (1D) discretized continuum in are expressed as a column vector, for example, the velocity in the $x$- or $\hat{i}$-direction at points $i = 1 : n$ is:

$$u_i \; \forall \; i = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_n \end{bmatrix}^T$$

Selected LaTeX equations:

```
$$
\mathbf{u} = u \,\hat{i} + v \,\hat{j} + w \,\hat{k}
$$
```

# Derivatives

The *total* or *direct derivative* uses $d$. For example, if $z = g(y)$ and $y = f(x)$, the chain rule is:

$$\frac{dz}{dx} = \frac{d}{dx} g(f(x)) = \frac{dg}{df} \frac{df}{dx}$$

The partial derivative uses $\partial$. For example, the time derivative of $f(\mathbf{x}, t)$ can be represented as follows:

$$\frac{\partial f}{\partial t} = \frac{\partial f(\mathbf{x}, t)}{\partial t}$$

The *material derivative* is defined using an uppercase letter $D$ to denote the direct derivative in the Lagrangian frame:

$$\frac{D\mathbf{u}}{Dt} = \frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla) \, \mathbf{u}$$

Selected LaTeX equations:

```
$$
\frac{D\mathbf{u}}{Dt}
= \frac{\partial\mathbf{u}}{\partial t}
+ (\mathbf{u}\cdot\nabla)\,\mathbf{u}
$$
```

# Nabla Operations

The nabla operator is:

$$\nabla = \frac{\partial}{\partial x}\,\hat{i} + \frac{\partial}{\partial y}\,\hat{j} + \frac{\partial}{\partial z}\,\hat{k}$$

When applied to a scalar, $\nabla$ specifies the gradient, which produces a vector quantity:

$$\nabla\phi = \frac{\partial\phi}{\partial x}\,\hat{i} + \frac{\partial\phi}{\partial y}\,\hat{j} + \frac{\partial\phi}{\partial z}\,\hat{k}$$

When applied to a vector, $\nabla$ specifies the divergence, which produces a scalar quantity:

$$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z}$$

When a vector is applied to $\nabla$ the div operator is specified, which produces a scalar quantity that is typically applied to another scalar, vector or tensor as part of a larger expression:

$$\mathbf{u} \cdot \nabla = u\frac{\partial}{\partial x} + v\frac{\partial}{\partial y} + w\frac{\partial}{\partial z}$$

The Laplacian is defined as $\nabla^2$, and is the divergence of the gradient $\nabla \cdot \nabla$. When applied to a scalar it produces a scalar quantity:

$$\nabla^2\phi = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}$$

Selected LaTeX equations:

```
$$
\nabla\
= \frac{\partial }{\partial x} \,\hat{i}
+ \frac{\partial }{\partial y} \,\hat{j}
+ \frac{\partial }{\partial z} \,\hat{k}
$$
```

## Material Derivative

The *material derivative* (also called *Lagrangian derivative*) describes the time rate of change of a scalar or vector quantity $\phi(\mathrm{x}, t)$ that is subject to a velocity field $\mathrm{u}(\mathrm{x}, t)$:

$$\frac{\mathrm{D}\phi}{\mathrm{D}t} = \frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi$$

where $\mathrm{D}$ represents the total derivative in the Lagrangian reference frame.

# Conservation Laws

Most physics-based modelling relies on the use of conservation laws. Most common are mass, momentum, energy, transport, here mas conservation and the equation of transport are described.

## Mass

Mass conservation is defined using the continuity equation, which describes the flux of a quantity per unit volume:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

where $\rho$ and $\mathbf{u}$ are the density and velocity of a material; for the FVM we will typically consider fluids, but the law is also valid for (deformable) solids.

## Equation of Transport

Many phenomena can be described by the Equation of Transport, here formulated using the material derivative. Consider the quantity to be a scalar or vector field and a function of

space and time, $\phi = \phi(\mathbf{x}, t)$. This quantity of interest can be a scalar like a Temperature or a polutant, a vector flow velocity, etc. So for a fluid, its particles are the ones transporting the quantity $\phi$, which is described by a velocity vector field, $\mathbf{u}$ (indeed, the flow velocity transports itself: momentum). The general form reads:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\phi\mathbf{u}) = \nabla \cdot (D\nabla\phi) + F_\phi$$

For incompressible flow, constant diffusion coefficient ($D$) and accounting for mass conservation, the equation can be expressed as:

$$\frac{\partial\phi}{\partial t} + \mathbf{u} \cdot \nabla\phi = D^*\nabla^2\phi + F_\phi^*$$

where $D^*$ and $F^*$ are $D$ and $F$ divided by $\rho$. From now on, the asterisk will be dropped.

From left to right, these three terms are explained as follows, relative to the quantity $\phi$:

1. Transient effect: rate of change of quantity $\phi$
2. Convection: transport due to velocity field $\mathbf{u}$
3. Diffusion: transport due to gradient of quantity $\phi$
4. Fount/Source $F_\phi$: it refers to a quantity that can be produced or removed locally. For example, a point pollution source.

Sometimes the term *convection* is used interchangeably with advection; technically the difference is that advection specifies a quantity that is carried by the fluid, rather than the fluid particles themselves (convection).

Selected LaTeX equations:

```
$$
\frac{\partial \phi}{\partial t}
+  \mathbf{u} \cdot \nabla \phi
= D \nabla^2 \phi + S_{\phi}
$$
```

# PDE's nature

In the previous numerical modelling weeks, we saw the classification of differential equations in ODEs, PDEs and their order. Now we will consider the physical nature of PDEs as it will give insight into the correct modeling approach. We dealt with the temperature

distribution along a rod when the problem was steady (in equilibrium) and when it was unsteady.

**Elliptic equations** are generally steady state problems just like the Temperature distribution in equilibrium along a rod, or stress distribution in a solid object due to a load. A relevant feature is that a disturbance in the interior points, e.g., a single one, changes the solution in the entire domain. So, the information propagates instantly in every direction. A classic elliptic equation is Laplace

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0$$

**In Parabolic equations**, the propagation speed of information is also infinite. This kind of problems are dominated by diffusion, so the solutions tend to be smooth. A classic parabolic equation is the diffusion equation (here shown in 1D):

$$\frac{\partial \phi}{\partial t} = D \frac{\partial^2 \phi}{\partial x^2}$$

**In Hyperbolic equations**, the information travels at a finite speed, i.e. the influence of a local perturbation is also local! Due to the lack of a diffusion mechanism, discontinuities (or sharp local gradients in $\phi$) persist!. A classic hyperbolic equation is the advection equation (here shown in 1D):

$$\frac{\partial \phi}{\partial t} + c \frac{\partial \phi}{\partial x} = 0$$

where the quantity $\phi$ is transported by the fluid velocity $u$.

Note that these equations can be derived from the Equation of transport, in the absence of source terms and a solid (no advection) the diffusion equation is obtained. In the absence of source terms and diffusion, the advection equation is obtained.

**Exercises**

This is the wave equation:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

What type of partial differential equation (PDE) is the wave equation?

- ○ parabolic
- ○ elliptic
- ○ None of the above
- ○ hyperbolic

Check

**Attribution**

This chapter is written by Robert Lanzafame and Jaime Arriaga Garcia. Find out more here.

# Finite Volume Method

## Contents


- Comparison with Finite Differences in a 2D regular mesh


Consider a a flow with a velocity field $\mathbf{u}$. The rate of change of $\phi$ can be evaluated casting the **transport equation**:

$$\frac{\partial(\rho\phi)}{\partial t} + \nabla \cdot (\rho\phi\mathbf{u}) = \nabla \cdot (D\nabla\phi) + F_\phi$$

In the Finite Difference Method, the region of interest would be divided in points and the derivatives would be approximated with numerical derivatives.

In the case of Finite Volume, the region of interest is divided in volumes and the **equation is integrated over those control volumes**:

$$\int_V \frac{\partial(\rho\phi)}{\partial t} dV + \int_V \nabla \cdot (\rho\phi\mathbf{u}) dV = \int_V \nabla \cdot (D\nabla\phi) dV + \int_V F_\phi dV$$

**This is the key difference between FDM and FVM**.

Now, Gauss's divergence theorem is used to rewrite the advection and diffusion terms. Gauss's theorem replaces the integral over the volume for integrals over the surface(s) as:

$$\int_V \nabla \cdot \mathbf{r}\, dV = \int_S \mathbf{n} \cdot \mathbf{r}\, dS$$

The physical interpretation of $\mathbf{n} \cdot \mathbf{r}$ is a projection of the vector $\mathbf{r}$ over the normal component of the surface. In other words, it replaces the changes over the volume of $\nabla \cdot \mathbf{r}$ for a flux over the surface! Applying it over the integral form of the transport equation:

$$\frac{\partial}{\partial t} \int_V (\rho\phi) dV + \int_S \mathbf{n} \cdot (\rho\phi\mathbf{u}) dS = \int_S \mathbf{n} \cdot (D\nabla\phi) dS + \int_V F_\phi dV$$

The first term represents the rate of change of the property $\phi$ over in the volume. The component inside the second integral represents the flux of $\phi$ (due to advection) along the normal vector of the surface, as it point outwards it represents the net rate of decrease. The component inside the third integral is the diffusive flux, as diffusion is directed towards negative gradients, the integral represents the net rate of increase due to diffusion. The fourth integral is the net rate of creation/removal of $\phi$ depending on fount/sink behavior inside the volume, e.g. a volume can coincide with a pollutant (enterococcus) source location or the pollutant can decrease due to sun exposure.

Finally, the fluxes can be approximated with a finite difference, e.g. central differences, and the integral can be approximated with any of the numerical integration techniques that were treated last quarter, e.g. midpoint rule, if evaluated at the mid point of the surface.

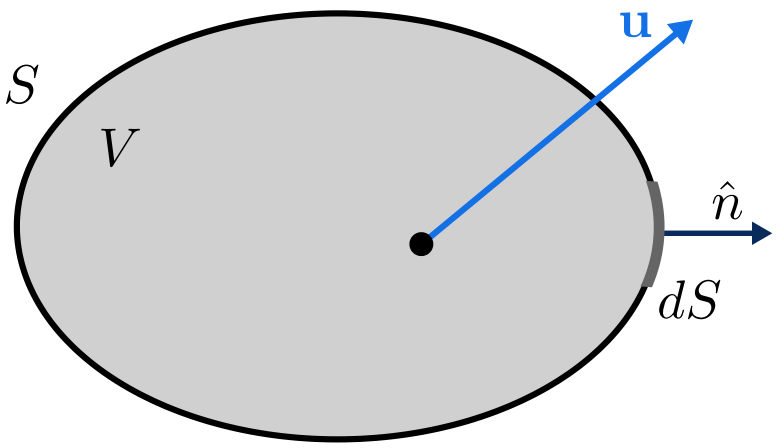



*Fig. 1.1* Generalized schematic of a 3D "volume", $V$, illustrated by a 2D ellipse, with bounding surface § and a single vector of the field $\mathbf{u}$. Flux out of the volume is through an infinitesimal part of the surface, $d$§, which has a surface normal vector $\hat{\mathbf{n}}$.

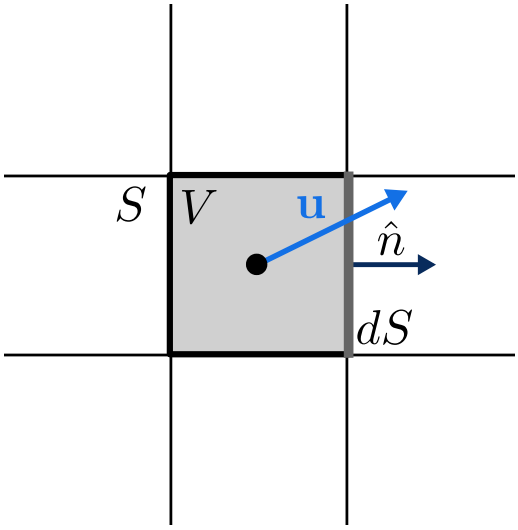



*Fig. 1.2* Illustration of a finite volume. The quantity of interest ($\mathbf{u}$ in this case) and conservation laws are identical to the generic case illustrated in Fig. 1.1, and computed at the center of the finite volume.

For an domain of interest, many finite volumes can be defined. For a Cartesian coordinate system, $\mathbf{x}$, such that the center point of each volume $\mathbf{x}_{i,j,k}$ is the point where $\mathbf{u}_{i,j,k}$ is computed. This is illustrated for a 2D example illustrated in Fig. 1.3, for volume $i$, $j$. In this case the $x$ and $y$ axes are sub-divided into increments $\Delta x$ and $\Delta y$. Although only cubic volumes are considered here (square in 2D), in practice the finite volumes can employ a variety of different shapes to make the geometric discretization into finite volumes and numerical computations more efficient.

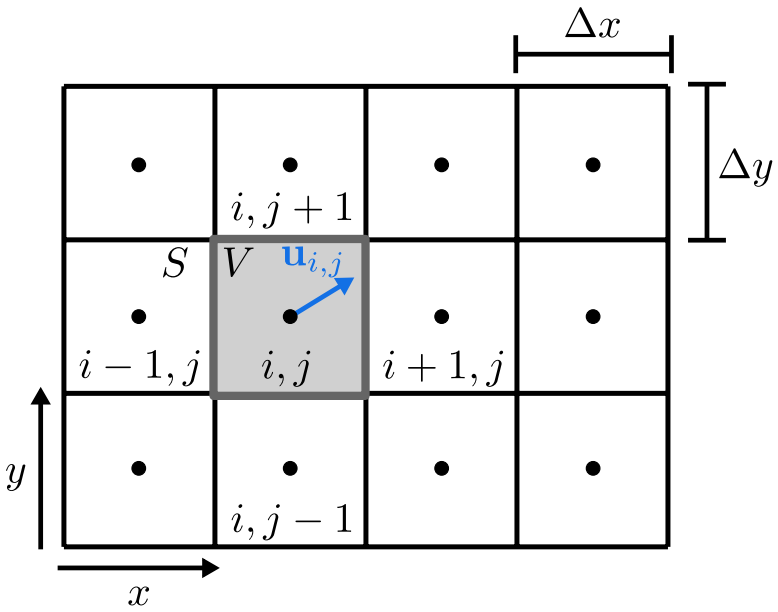



*Fig. 1.3* Discretization of a generic domain of interest into finite volumes $V$ in the 2D Cartesian coordinate system. Volume $i$, $j$ is shown alongside neighboring volumes that will be used in flux calculations. The quantity of interest ($\mathbf{u}$ in this case) are computed at the center of each finite volume.

As seen in other chapters of this textbook, to solve PDE's numerically, values of the function of interest at several discrete points in $\mathbf{x}$ are required to formulate and computing numerical differentiation and integration schemes (in space *and* time). However, as illustrated in Fig. 1.2 and Fig. 1.3, the finite volume discretization uses only a single central point to represent the quantity of interest. Therefore, the finite volume method requires use of neighboring volumes in the numerical schemes. As the conservation laws require $\mathbf{u}$ to be balanced at the boundaries of every finite volume, this forms an ideal point at which to make computations. The boundary of a standard finite volume is thus divided into four faces, North, South, East, West (abbreviated N, S, E, W), as illustrated in Fig. 1.4, and $\mathbf{u}$ is computed at each face, typically as an average using the neighboring volumes.

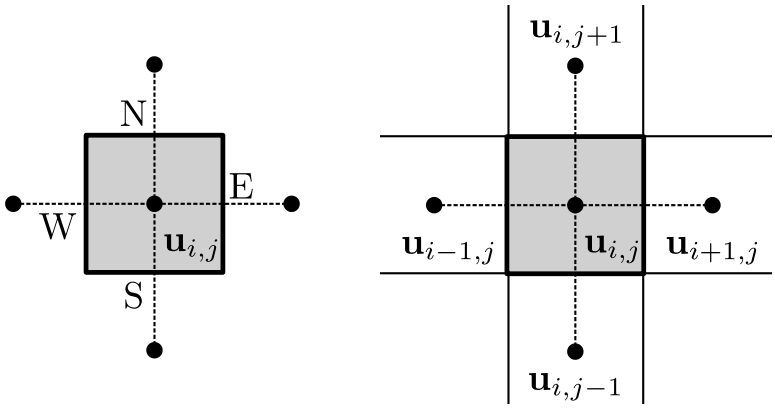



*Fig. 1.4* Left: specification of North, South, East and West faces (N, S, E, W) for a standard finite volume. Right: center points of neighboring finite volumes that can be used to compute $\mathbf{u}$ at each face, as well as $\mathbf{u}$.

Subsequent chapters will illustrate the specific discretization for certain cases of the transport equation.

# Comparison with Finite Differences in a 2D regular mesh

Both methods can be used to solve the same conservation laws over a discretized domain of interest $V$; in fact, the points at which the quantity of interest is calculated are exactly the same. However, whereas the FVM uses each point to represent the *center* of a finite volume, FDM uses the points as corners of square units in a grid. Unlike FDM, FVM uses surface fluxes, computed at the boundary of each finite volume; these points are located between the evaluation points of the FDM grid. In addition, FVM integrates over the volumes prior to applying numerical integration schemes, whereas FDM uses numerical differentiation to discretize the PDE's. This gives a very important advantage to the FVM: quantities are conserved! This is not ensured while using FDM.

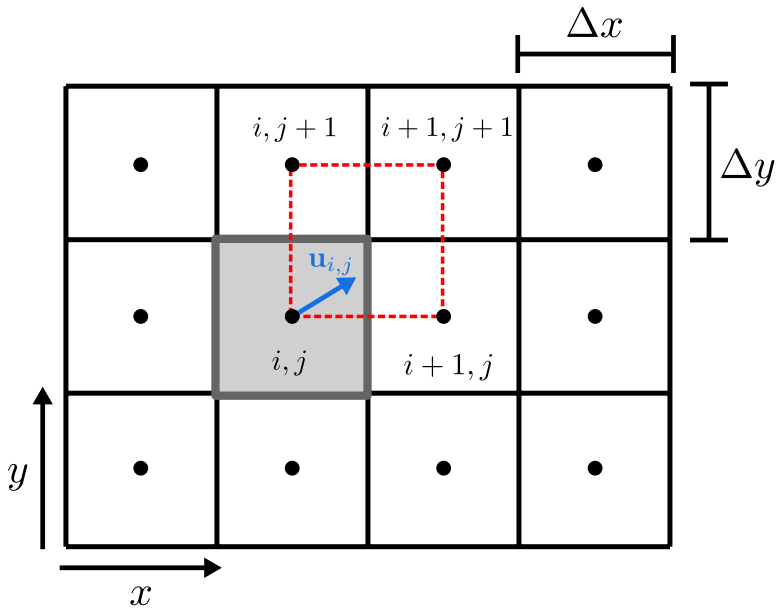



*Fig. 1.5* A finite difference unit compared to a finite volume using the same grid of points $(x_i, y_j)$ for computation of the quantity of interest $\mathbf{u}_{i,j}$ in both methods. Boundary surface fluxes of FVM volumes are computed at points located between evaluation points of the FDM.

**Exercises**

This is to practice with the concept conservation laws and Gauss' divergence theorem. The divergence theorem can be used to connect a PDE(which applies at every point in space) to a global integral form of the conservation laws over a region of space.

Let take for example Gauss's law for Electric Fields, this law describes the relationship between electric flux through a surface and the charge within that surface:

$$\epsilon_0 \nabla \cdot E = \rho$$

From this deferential equations integrate over the control volumes and derive the global conservation law

**Solution**

$$\epsilon_0 \nabla \cdot \mathbf{E} = \rho$$

By integrating both sides over a volume $V$:

$$\int_V \epsilon_0 \nabla \cdot \mathbf{E} dV = \int_V \rho dV$$

$$\epsilon_0 \int_V \nabla \cdot \mathbf{E} dV = \int_V \rho dV$$

apply the Gauss' divergence theorem

$$\int_V \nabla \cdot \mathbf{r}\, dV = \int_S \mathbf{n} \cdot \mathbf{r}, dS$$

After applying the Divergence Theorem, we arrive at:

$$\epsilon_0 \int_S \mathbf{n} \cdot \mathbf{E} dS = \int_V \rho dV$$

**Exercises**

Suppose we have a certain function $\phi(\vec{x})$ which solves the following problem:

**Differential Equation:**

$$-\nabla \cdot (\kappa \nabla \phi) = f, \quad \mathbf{x} \in \Omega;$$

2. **Boundary Condition:**

$$-\kappa(\nabla \phi \cdot \mathbf{n}) = g, \quad \mathbf{x} \in \partial\Omega.$$

Show that the following compatibility condition must be satisfied (hint: use Divergence theorem):

$$\int_S g\,dS = \int_V f\,dV.$$

**Solution**

substitute f and g in Equation $\int_S g\,dS = \int_V f dV$ we get:

$$\int_S -\kappa(\nabla\phi \cdot \mathbf{n})\,dS = \int_V -\nabla\cdot(\kappa\nabla\phi)\,dV$$

now apply the Divergence theorem:

$$\int_S -\kappa(\nabla\phi \cdot \mathbf{n})\,dS = \int_S -(\kappa\nabla\phi)\cdot\mathbf{n})\,dS$$

rearrange and because $\kappa$ is a scalar:

$$\int_S -\kappa(\nabla\phi \cdot \mathbf{n})\,dS = \int_S -\kappa(\nabla\phi\cdot\mathbf{n})\,dS$$

This means that the compatibility condition is satisfied, as required

**Attribution**

This chapter is written by Robert Lanzafame and Jaime Arriaga Garcia. Find out more here.

# Advection

## Contents


- Application in 1D
- Stability
- Diffusion
- Stability


In the first chapter, it was shown that for incompressible flow, constant diffusion coefficient ( $D$) and accounting for mass conservation, the equation of transport can be expressed as:

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = D \nabla^2 \phi + F_\phi$$

A pure advection problem can be used to illustrate the finite volume method and to isolate the problems that may arise while treating this term. Consider the idealized problem without diffusion and source terms, a quantity $\phi(\mathbf{x}, t)$ traveling in a fluid with velocity field $\mathbf{u}(\mathbf{x}, t)$:

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} + \mathbf{u}(\mathbf{x}, t) \cdot \nabla \phi(\mathbf{x}, t) = 0$$

This relationship describes phenomenon that can be thought of as a “pulse” of quantity $\phi$ traveling through the fluid, for example a single wave ($\phi$ being the wave amplitude), or a pollutant ($\phi$ as concentration). In the example presented here, we will consider the pulse being carried (advected) through a fluid with constant velocity $\mathbf{c}$, where the components in each direction are:

$$\mathbf{c} = c_u \, \hat{i} + c_v \, \hat{j} + c_w \, \hat{k}$$

As the fluid velocity is constant in time and space, we can re-write the conservation equation as follows:

$$\frac{\partial \phi(\mathbf{x}, t)}{\partial t} + \mathbf{c} \cdot \nabla \phi(\mathbf{x}, t) = 0$$

# Application in 1D

The finite volume method will be applied in 1D for simplicity following the steps:

1. Integrate PDE over volume of interest $V$
2. Apply Gauss's Theorem on the bounding surface $S$
3. Discretize the volume and integral equations over finite volumes (in space): meshing
4. Apply a numerical scheme to discretize equations in time, then solve

# Integral Form

Consider only the $x$-direction to represent a 1D problem:

$$\frac{\partial \phi}{\partial t} + c \frac{\partial \phi}{\partial x} = 0$$

where the subscript $u$ (denoting the $u$-component of $\mathbf{c}$) has been dropped. The situation represents a "pulse" traveling along the $x$-axis with speed $c$, and is illustrated in Fig. 1.6 for initial conditions such that a pulse with width $(b-a)$ has amplitude $\phi_0$ between the interval $(a, b)$ in $x$.

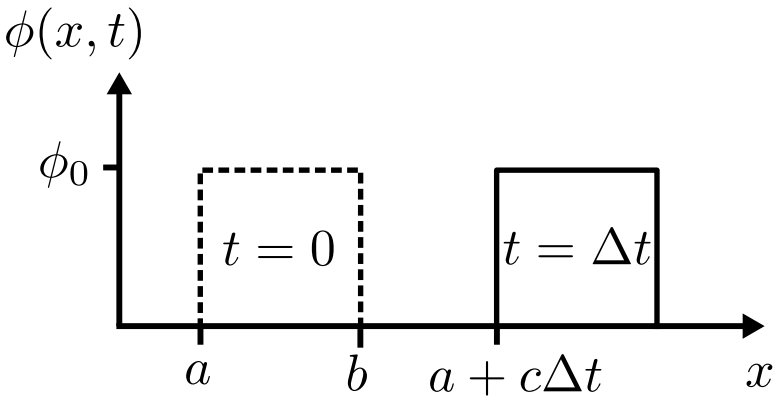



*Fig. 1.6* Propagation of wave in 1D with constant speed $c$.

Integrate the linear convection equation in $x$ over an arbitrary volume $V$ with surface $S$:

$$\int_V \frac{\partial \phi}{\partial t} dV + \int_V c \frac{\partial \phi}{\partial x} \, dV = 0$$

# Applying Gauss's theorem

Recognizing $\frac{\partial \phi}{\partial x}$ as the 1D divergence $\nabla \cdot u$ and applying Gauss's Theorem with $\mathbf{n}$ being the normal surface vector:

$$\int_V \frac{\partial \phi}{\partial t} dV + \int_S c\, \phi \cdot \mathbf{n}\, dS = 0$$

# Finite Volume Mesh

The $x$-domain will be divided into six finite volumes, shown in Fig. 1.7, numbered 1 through 6 from left to right, such that the volume number refers to the $x$-coordinate of the geometric center. For example, volume 3 is located with center at $x_3$. The quantity $\phi$ will only being calculated at the center of the finite volume, $\phi(x_i)$.

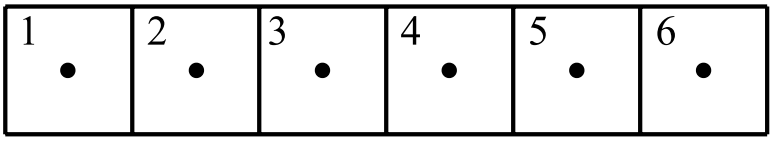



*Fig. 1.7* Schematic of 6 finite volumes aligned with $\hat{i}$ ($x$-direction).

# Discretization

The integral equations may now be discretized over each finite volumes with the "unknown" at the center, $\phi_i$, and with dimensions $\Delta x$, $\Delta y$ and $\Delta z$. .

For the first term: velocity $c$ is constant over the domain, the integral is applied over the finite volume $i$ and a forward Euler scheme is applied with time-step $\Delta t$:

$$\int_V \frac{\partial \phi}{\partial t} dV = \frac{d\phi_i}{dt} \int dV = \frac{d\phi_i}{dt} \Delta x \Delta y \Delta z$$

$$= \frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} \Delta x \Delta y \Delta z$$

Note that even though it is a 1D problem, the volume is still represented with the three dimension $\Delta$ to represent $\Delta V$

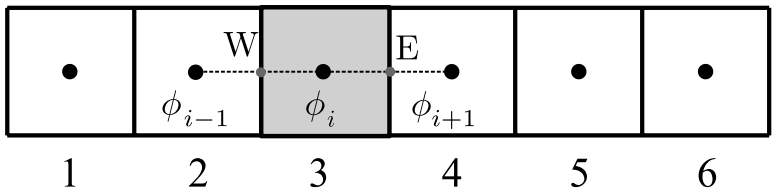



*Fig. 1.8* 6 finite volumes showing discretization around $\phi_i$ ($x$-direction) and the Western and Eastern boundary .

For the second term: the general surface integral can be represented as the sum of integrals over the active volume surfaces (east and west) as visualized in the figure above:

$$\int_S c\,\phi \cdot \mathbf{n}\, dS = \int_{S_w} c\,\phi_w \cdot \mathbf{n}_w\, dS + \int_{S_e} c\,\phi_e \cdot \mathbf{n}_e\, dS$$

In the previous equation the west and east faces are explicitly written as subscript in the normal vector $\mathbf{n}$ to emphasize that it is the normal to those surfaces pointing outwards the volume, so the normal vector of the west surface is $-i$ (pointing outwards) and the normal vector of the east surface is $i$, i.e. one integral has a negative sign and the other a positive one:

$$\int_S c\,\phi \cdot \mathbf{n}\, dS = -\int_{S_w} c\,\phi_w\, dS + \int_{S_e} c\,\phi_e\, dS$$

For a 1D problem there wont be any variations of $\phi$ along the surface, if there were we can already approximate that value with a numerical integration technique (Left Riemman, midpoint, etc). Recalling that the midpoint evaluation is second order accurate and convenient, we obtain:

$$\int_S c\,\phi \cdot \mathbf{n}\, dS = -c\,\phi_w\,\Delta y \Delta z + c\phi_e\,\Delta y \Delta z = c\,(\phi_e - \phi_w)\Delta y \Delta z$$

where $\phi_{e,w}$ are evaluated at the midpoint of the east and west surfaces. As those values are not yet known at the surface (they are known at the volume centroid), an approximation is needed. A linear interpolation can be used, thus only two points per surface value are needed:

$$\phi_e = \frac{\phi_{i+1} + \phi_i}{2} \qquad \phi_w = \frac{\phi_{i-1} + \phi_i}{2} \quad \rightarrow \quad \phi_e - \phi_w = \frac{\phi_{i+1} + \phi_i - \phi_{i-1} - \phi_i}{2}$$

Putting both integral terms together:

$$\frac{d\phi_i}{dt}\Delta x\Delta y\Delta z + c\left(\frac{\phi_{i+1}-\phi_{i-1}}{2}\right)\Delta y\Delta z = 0$$

This is an ODE! We can use one of the methods of the previous quarter to solve it. For example, using Explicit Euler and dividing by $\Delta x\Delta y\Delta z$:

$$\frac{\phi_i^{n+1}-\phi_i^n}{\Delta t} + \frac{c}{2\Delta x}\left(\phi_{i+1}^n - \phi_{i-1}^n\right) = 0$$

Rearranging, the algebraic equation for the unknown term at the next time step:

$$\phi_i^{n+1} = \phi_i^n - \frac{c\Delta t}{2\Delta x}\left(\phi_{i+1}^n - \phi_{i-1}^n\right)$$

**Turns out that the 1D problem final algebraic expression is the same as for the finite difference method using central differences to approximate the spatial derivative! The differences become evident once non-rectangular volumes are used.**

Unfortunately, the use of Explicit Euler in this case is useless as it will make the problem unstable for any time step.

# Stability

As shown with ODE's, the existance of a stability criteria only applies for explicit schemes. Unfortunately, using Explicit Euler in this case is useless as it will make the problem unstable for any time step (to be tested in the workshop). The physical reason why this combination worked fine with the diffusion equation is that diffusivity occurs in all directions (elliptic, parabolic problem depending on unsteadiness), but for advection (hyperbolic equation) information mainly comes from upstream (the direction of information propagation), so it is not logical to have equal weights from contiguous volumes to define the flux of $\phi$! It should be more relevant the quantity coming from upstream.

This can be circumvented in two ways. One is by using central differences in time:

$$\phi_i^{n+1} = \phi_i^{n-1} - \frac{c\Delta t}{\Delta x}\left(\phi_{i+1}^n - \phi_{i-1}^n\right)$$

Indirectly, there is more relevance from the upstream direction by using $\phi^{n-1}$ rather than $\phi^n$ .

The other and more easily appreciated is by applying an upwind scheme. This, if $c$ is positive:

$$\phi_e = \phi_i \qquad \phi_w = \phi_{i-1}$$

if $c$ is negative :

$$\phi_e = \phi_{i+1} \qquad \phi_w = \phi_i$$

.

The fist option scheme is named leapfrog because the value $\phi_i^n$ is skipped. The resulting stability criteria, here shown without proof, is the *Courant-Friedrich-Lewy criteria* (CFL):

$$\frac{|u|\Delta t}{\Delta x} < 1$$

The physical meaning of this criteria is that time step should be less than the time that takes the information to cross to the next volume. Thus, if a very fine spatial resolution is used, the time-set must be very small to maintain the CFL criteria.

# Solution as System of Equations

We will now assemble a system of equations for the entire problem and formulate it as a matrix. This is instructive not only for understanding the solution and how the individual finite volumes relate to each other geometrically, as well as to the boundary conditions, but it also facilitates a comparison with other numerical methods (e.g., finite difference and finite element methods).

Observe that the general forward Euler solution above for a non-boundary volume can be expressed in vector/matrix form as:

$$\phi_i^{n+1} = \phi_i^{n-1} - \frac{c\Delta t}{\Delta x}\begin{bmatrix}-\frac{1}{2} & 0 & \frac{1}{2}\end{bmatrix}\begin{bmatrix}\phi_{i-1}\\ \phi_i\\ \phi_{i+1}\end{bmatrix}^n$$

Recognizing that the vector of coefficients $\begin{bmatrix}-\frac{1}{2} & 0 & \frac{1}{2}\end{bmatrix}$ can be written $\begin{bmatrix}a_{i,i-1} & a_{i,i} & a_{i,i+1}\end{bmatrix}$ and is the $i^{\text{th}}$ equation in the system allows generalizing to the global matrix formulation:

$$[\,\phi\,]^{n+1} = [\,\phi\,]^{n-1} + [\,A\,]^n [\,\phi\,]^n$$

Where the coefficients of $A$, $a_{ij}$, are defined by the finite volume discretization scheme:

$$\begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^{n+1} = [\,I\,] \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^{n-1} + \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16}\\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26}\\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36}\\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46}\\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56}\\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66}\end{bmatrix}^n \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^n$$

Applying and extending the pattern found above:

$$\begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^{n+1} = [\,I\,] \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^{n-1} - \frac{c\Delta t}{\Delta x} \begin{bmatrix} 0 & \frac{1}{2} & 0 & 0 & 0 & 0\\ -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0\\ 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0\\ 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2} & 0\\ 0 & 0 & 0 & -\frac{1}{2} & 0 & \frac{1}{2}\\ 0 & 0 & 0 & 0 & -\frac{1}{2} & 0\end{bmatrix} \begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\\ \phi_4\\ \phi_5\\ \phi_6\end{bmatrix}^n$$

The initialization of the problem is delicate as only after the second time step the algebraic equation is fully satisfied without additional requirements.

The coefficient matrix $A$ only has non-zero elements along the upper and lower diagonal due to the regular geometry incorporated in the 1D discretization. In the derivation of the system of equations, the boundary conditions of end volumes were not presented. Actually, the advection equation has only one boundary condition due to the first spatial derivative! It is defined at the boundary from where $\phi$ is being advected. The formulation of the coefficients in the matrix implies that the values of the $0^{\text{th}}$ and 7th volume ($\phi_0$ and $\phi_7$) were taken as 0, the latter is an artifice to vanish the quantity $\phi$. Rigorously speaking, it should not be a boundary condition.

# Solution Techniques

This system can be solved without the need for building the matrix as each value of the next time step can be computed from known values of previous steps. So, it can be solved with a for loop as well!

Note that the matrix in the above example contains a lot of zeros: 10 out of 36 elements are non-zero (28%). This is called a *sparse* matrix, and the proportion of non-zero entries grows with the size of the problem. For example, $N = 1000$ finite volumes requires $2N - 2$ non-zero entries (1998), which is only 0.1% of the matrix! Since matrix calculations are computationally expensive, we would never carry out the matrix calculations as represented here. Instead, various vectorization approaches are implemented in all numerical analysis software packages. Typically, a mapping of volumes and coordinates are stored in a vector format that retains the relationship between volumes and the interpolation points (e.g., neighbor volumes). Then calculations per volume are carried out using this vectorized information. This can be easily implemented in any programming language, for example, with a for loop over each volume. However, note that as geometry of the domain of interest and the discretization scheme changes (e.g., non-square volumes!) more advanced vectorization techniques are required.

# Diffusion

Lets look at the diffusion equation.

$$\frac{\partial \phi}{\partial t} - D\nabla^2\phi = F_\phi$$

Previously, we analyzed the advection term and derived it for the finite volume method. Now, let's examine the diffusion equation, which is a second-order differential equation. For simplicity, assume that $F_\phi = 0$. We will follow a similar approach as before::

Integrate the diffusion equation over an arbitrary volume:

$$\int_V \frac{\partial \phi}{\partial t} dV = \int_V D\frac{\partial^2 \phi}{\partial x^2}\, dV$$

## Applying Gauss's theorem

Recognizing $\frac{\partial \phi}{\partial x}$ as the 1D divergence $\nabla \cdot u$ and applying Gauss's Theorem with $\mathbf{n}$ being the normal surface vector:

$$\int_V \frac{\partial \phi}{\partial t} dV = \int_S D\frac{\partial \phi}{\partial x} \cdot \mathbf{n}\, dS$$

We first focus on the diffusion term $\int_S D\frac{\partial \phi}{\partial x} \cdot \mathbf{n}\, dS$ :

$$\int_S D\frac{\partial \phi}{\partial x} \cdot \mathbf{n}\, dS = \int_{S_w} D\, \frac{\partial \phi_w}{\partial x} \cdot \mathbf{n}_w\, dS + \int_{S_e} D\, \frac{\partial \phi_e}{\partial x} \cdot \mathbf{n}_e\, dS$$

Here, the west and east faces are explicitly indicated with subscripts $w$ and $e$ in the normal vector $\mathbf{n}$ to show that it points outward from the volume. For example, the normal vector on the west surface is $-\hat{i}$ (pointing outwards), while on the east surface it is $\hat{i}$, leading to a negative sign for one integral and a positive sign for the other:

$$\int_S D\frac{\partial \phi}{\partial x} \cdot \mathbf{n}\, dS = -D\int_{S_w} \frac{\partial \phi_w}{\partial x}\, dS + D\int_{S_e} \frac{\partial \phi_e}{\partial x}\, dS$$

In Q1, we learned that we can approximate the first-order derivative forward and backward over the grid as::

$$\frac{\partial \phi_w}{\partial x} \approx \frac{\phi_i - \phi_{i-1}}{\Delta x}$$

$$\frac{\partial \phi_e}{\partial x} \approx \frac{\phi_{i+1} - \phi_i}{\Delta x}$$

Substituting these approximations, we get:

$$\int_S D\frac{\partial \phi}{\partial x} \cdot \mathbf{n}\, dS = -D\frac{\phi_i - \phi_{i-1}}{\Delta x}\Delta y\Delta z + D\frac{\phi_{i+1} - \phi_i}{\Delta x}\Delta y\Delta z$$

Substituting this expression back together with the first term, we get:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t}\Delta x\Delta y\Delta z = -D\frac{\phi_i^n - \phi_{i-1}^n}{\Delta x}\Delta y\Delta z + D\frac{\phi_{i+1}^n - \phi_i^n}{\Delta x}\Delta y\Delta z$$

This simplifies to:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t}\Delta x\Delta y\Delta z = D\frac{\phi_{i-1}^n - 2\phi_i^n + \phi_{i+1}^n}{\Delta x}\Delta y\Delta z$$

Finally , dividing by $\Delta x\Delta y\Delta z$:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = D\frac{\phi_{i-1}^n - 2\phi_i^n + \phi_{i+1}^n}{\Delta x^2}$$

# Stability

This equation can be rewritten to isolate the amplification factor:

$$\phi_i^{n+1} = (1 - 2 * \Delta t D/\Delta x^2)\phi_i^n + \Delta t D\frac{\phi_{i-1}^n + \phi_{i+1}^n}{\Delta x^2}$$

where the amplification factor is multiplying $\phi_i^n$ and should be lower than 1 to ensure stability:

$$-1 < 1 - 2D\frac{\Delta t}{\Delta x^2} < 1$$

thus

$$D\frac{\Delta t}{\Delta x^2} <= 1/2$$

> **Attribution**
>
> This chapter is written by Robert Lanzafame and Jaime Arriaga Garcia. `{ref}` Find out more here <finite_volume_method_credit>`.

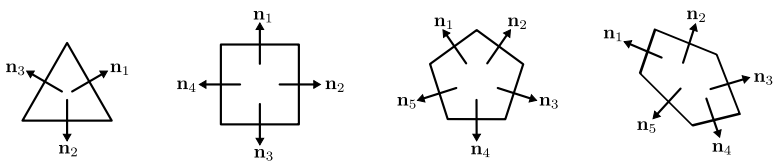
# Unstructured meshes

## Contents

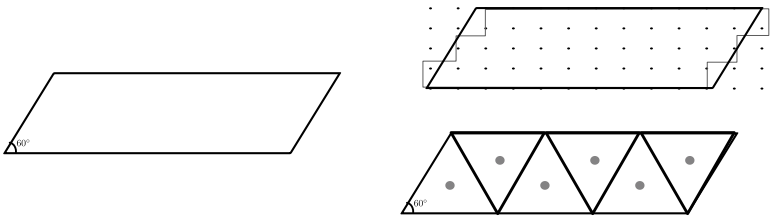

- Orthogonal and non-orthogonal meshes
- Special Treatment of Diffusion in non-orthogonal meshes
- Cross-Diffusion
- Conclusions of the implications of non-orthogonal meshes


So far we have used a Cartesian coordinate system to define the algebraic expressions. However, complex geometries cannot be represented accurately in such coordinate system. One advantage of the Finite Volume Method is its suitability for complex geometries, such as structured curvilinear meshes and specially for unstructured meshes. **It is important to mention that FDM can also be adapted to such meshes but it is more complicated to implement it.**

The essence of the Finite Volume Method is the conservation of a quantity by interpreting the influence of physical mechanisms as fluxes through the surfaces of a defined volume. That quantity is defined at the centroid and is representative of the entire volume. Rectangular shapes were used to illustrate a 1D convection problem but any shape can be used, “two” dimensions shapes (it is still a volume by assuming a uniform depth of the 2D shapes) like rectangles, triangles, pentagons, hexagons or non-regular shapes. It is evident that the number of surfaces with possible fluxes is different in every shape.

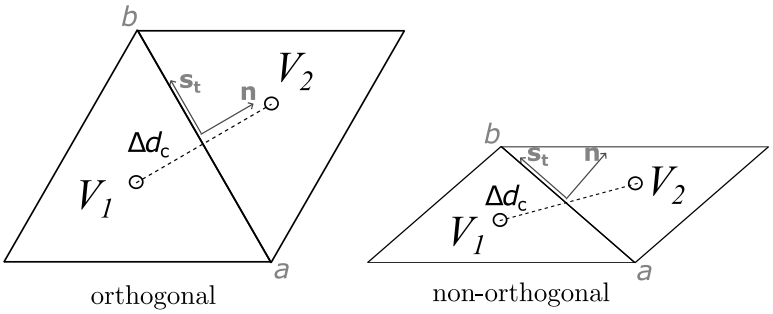



Consider a rectangular metal piece that has a 60 degree cut at both ends. How can you represent this problem using Finite Differences in a structured grid? For the FVM it is simple to just use triangle volumes.




# Orthogonal and non-orthogonal meshes

For a mesh composed only of equilateral triangles, just like for the metal with a cut of 60 degrees, the line connecting the centroids of two contiguous volumes aligns with the normal vector of the surfaces, this volumes are said to be orthogonal. However, if this is not the case, then the mesh is referred to as non-orthogonal. In practice, having a perfectly orthogonal mesh is too restrictive and a degree of non-orthogonality (non-uniform) is needed to represent complex geometries. A correction is needed!




# Special Treatment of Diffusion in non-orthogonal meshes

The flux due to diffusion in orthogonal volumes can be approximated adequately with a central difference scheme. However, the flux between non-orthogonal volumes (the line connecting the centroids and the normal to the surface are not parallel) needs a correction. Consider the formulation of the diffusion flux at the surface

$$\int_{S_i} \mathbf{n_i} \cdot (D\nabla\phi) dS = \mathbf{n_i} \cdot (D\nabla\phi)\Delta S_i$$

where the subindex $i$ refers to the surface number. The divergence operation projects the rate of change of $\phi$ in the normal direction of the surface $\mathbf{n_i}$:

$$\mathbf{n_i} \cdot (D\nabla\phi) = D\frac{\partial\phi}{\partial n}$$

The central differencing to define the flux at the surface using the values of $\phi$ at the centroids is:

$$\frac{\partial\phi}{\partial d_c} \approx \frac{\phi_{V2} - \phi_{V1}}{\Delta d_c}$$

This clearly shows that for orthogonal meshes the central differencing is adequate since $\mathbf{n} = \mathbf{d}_c$ therefore $\frac{\partial\phi}{\partial n} = \frac{\partial\phi}{\partial d_c}$. However, the more non-orthogonal the mesh is, the larger the deviation between $\mathbf{n}$ and $\mathbf{d}_c$. This is why cross-diffusion is required.

# Cross-Diffusion

To reveal the component of $\frac{\partial\phi}{\partial d_c}$ that contributes to $\frac{\partial\phi}{\partial n}$ we can rewrite $\nabla\phi$ in terms of the unit **normal** and **tangential** vectors of the surface:

$$\nabla\phi = \frac{\partial\phi}{\partial x}\mathbf{i} + \frac{\partial\phi}{\partial y}\mathbf{j} = \frac{\partial\phi}{\partial n}\mathbf{n} + \frac{\partial\phi}{\partial s}\mathbf{s_t}$$

$\mathbf{s}_t$ being the unit vector of the surface tangent. Then, $\nabla\phi$ can be projected in the direction of the line connecting the centroids:

$$\frac{\partial\phi}{\partial d_c} = \nabla\phi \cdot \mathbf{d}_c = \frac{\partial\phi}{\partial n}\mathbf{n} \cdot \mathbf{d}_c + \frac{\partial\phi}{\partial s}\mathbf{s_t} \cdot \mathbf{d}_c$$

which can be interpreted in geometric terms using the angle between $\mathbf{n}$ and $\mathbf{d}_c$:

$$\frac{\partial\phi}{\partial d_c} = \frac{\partial\phi}{\partial n}\cos\theta - \frac{\partial\phi}{\partial s}\sin\theta$$

from where it can be deduced

$$\frac{\partial\phi}{\partial n} = \frac{\partial\phi}{\partial d_c}\frac{1}{\cos\theta} - \frac{\partial\phi}{\partial s}\frac{\sin\theta}{\cos\theta}$$

The final expression is:

$$\int_{S_i} \mathbf{n_i} \cdot (D \nabla \phi) dS = (D \cdot \Delta S_i) \left( \frac{\partial \phi}{\partial d_c} \frac{1}{\cos \theta} - \frac{\partial \phi}{\partial s} \frac{\sin \theta}{\cos \theta} \right)$$

# Conclusions of the implications of non-orthogonal meshes

There is a dependence on $\theta$ which can be computed directly as it required the information of the known location of four points $1, 2, a, b$. The derivative $\frac{\partial \phi}{\partial d_c}$ is computed with the information of the location points $1, 2$ and the values $\phi_{V1}, \phi_{V2}$ and $\frac{\partial \phi}{\partial s}$ requires the location points $a, b$ and the values $\phi_a, \phi_b$. So we need to approximate the value of $\phi$ at the vertices! This last computation is specially costly and adds a potential error. Also, note that the intersection between the line connecting the centroids and the surface is not the midpoint of the surface, so the accuracy is reduced as consequence. These reasons highlight the importance of keeping the non-orthogonality of the mesh limited as much as possible.

> **Attribution**
>
> This chapter is written by Robert Lanzafame and Jaime Arriaga Garcia. Find out more here.

# Finite Element Method

> **Attribution**
>
> This chapter was written by Frans van der Meer. Find out more here.

In this chapter we will introduce all the steps that are usually followed to solve a PDE using the finite element method, from the definition of the PDE to the solution of a linear system of equations. To make the learning curve ease, we focus on the simplest PDE for finite elements: the Poisson equation, first in 1D and then in 2D. Note that the laplacian operator with second derivatives in space that appears in th Poisson equation is an ingredient in many different engineering problems, so this is not only a convenient place to start but also a fundamental one.

The general goal of the finite element method is to convert PDEs into a form which we, or rather a computer, can solve. The reason for doing this is because it allows a PDE with no analytical solution for the full domain, to be approximated by solving it on smaller 'elements'. This *discretization* into several elements enables us to solve the problems in irregular geometries, different materials, dynamic effects, jumps or complex changes in geometry, while the resulting formulation is well generalizable to nonlinear problems of different kinds.

We will first show the main steps of the derivation of the finite element formulation:

1. Starting point is a PDE with boundary conditions, the strong form of the problem
2. From the strong form, derive the weak form
3. Substitute an assumed discretization into the weak form to arrive at the discrete form

Then we will show a simple single-purpose implementation of the finite element method for the 1D Poisson equation

4. A finite element implementation translates the equations to code

In subsequent sections, key concepts of general finite element implementations are discussed:

5. The definition of shape functions which played a key role in the discretization step
6. Numerical integration for evaluating the integrals that are present in the discrete form

7. The derivation is repeated for the Poisson equation in 2D
8. The concept of isoparametric mapping for generalization to unstructured meshes

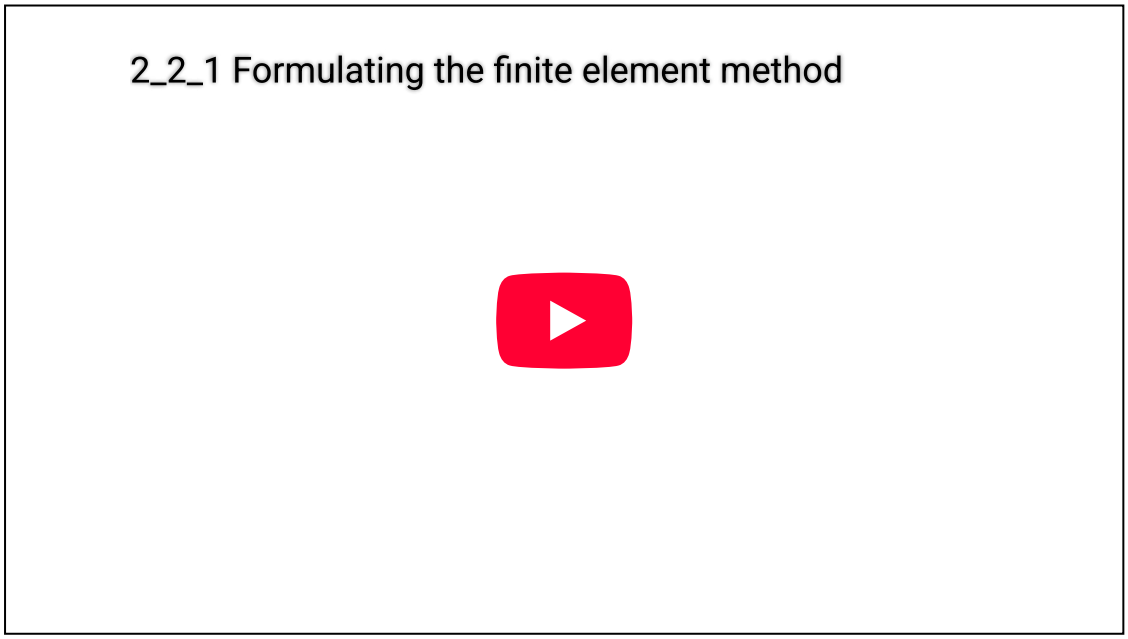
# Strong form of the 1D Poisson equation

## Contents


- 1D Poisson equation
- The strong form of a linear elastic rod


The *strong from* of the problem is a Partial Differential Equation (PDE), or a set of PDEs, describing the physics of the phenomena at hand, supplemented with appropriate boundary and initial conditions. Solving the strong form of the problem leads to the exact solution of the continuous system. However, for many engineering problems, it is not possible to obtain the exact solution, and numerical methods need to be employed to find an approximate solution. The finite element method is one approach to find approximate solutions that is applicable to a wide range of different problems. In this chapter we will focus on problems for which the strong form is the Poisson equation.

The story told below and up to the discrete form section is also presented in this video:



# 1D Poisson equation

At this point, let's take as an example the one dimensional Poisson equation to illustrate at first the so-called strong form of the problem:

$$-\frac{\partial}{\partial x}\left(\nu(x)\frac{\partial u(x)}{\partial x}\right) = f(x) \tag{2.1}$$

where $\nu(x)$ is a material property, $u(x)$ the solution and $f(x)$ a source, all of which can depend on the coordinate $x$. To simplify notation we will omit the $x$-dependence and write $\nu$, $u$ and $f$ in what follows.

Equation (2.1) can be used to express various physical phenomena such as steady-state heat conduction, elastic deformation of a rod or flow in a permeable media (such as Darcy's law).

Alternatively, the Poisson equation can be written as:

$$-\frac{\partial}{\partial x}\Lambda + f = 0 \tag{2.2}$$

where $\Lambda$ is a flux term, of which the interpretation depends on the problem at hand. In the case of heat conductivity, $\Lambda$ is the heat flux, while in Darcy's law, $\Lambda$ is the flow rate, and in extension of a rod, $\Lambda$ is the stress. The typical constitutive relationship between the flux $\Lambda$ and the unknown solution $u$ is given by:

$$\Lambda = -\nu\frac{\partial u}{\partial x} \tag{2.3}$$

In the case of heat conduction, for instance, $u$ in the above equation is the temperature. In the case of a structural problem, $u$ would be the displacement. For Darcy's law, $u$ is the hydraulic head.

Keep in mind that the constitutive relationship always depends on the problem being solved and $\nu$ can take different meanings. For the heat equation, $\nu$ is the thermal conductivity while for Darcy's law $\nu$ is the hydraulic conductivity and for extension of a rod $\nu$ is a measure for the stiffness.

# The strong form of a linear elastic rod

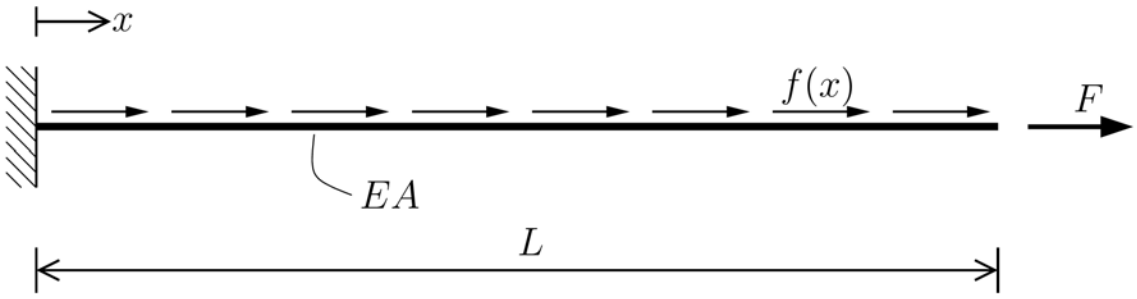



*Fig. 2.7* One dimensional rod with distributed load

An application of the Poisson equation is the extension of one-dimensional structural elements. Let us consider a horizontal steel rod with length $L$, stiffness $EA$ and distributed load $f(x)$. This equilibrium problem can be described with the following differential equation:

$$-\frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right) = f \tag{2.4}$$

where $u$ is the displacement of the rod as function of location $x$. For constant $EA$, this can be simplified to

$$-EA\frac{\partial^2 u}{\partial x^2} = f \tag{2.5}$$

We also consider as **boundary conditions** that the rod is fixed at the left end and loaded with a force $F$ at the right end:

$$\begin{aligned} u &= 0, \quad \text{at} \quad x = 0 \\ EA\frac{\partial u}{\partial x} &= F, \quad \text{at} \quad x = L \end{aligned} \tag{2.6}$$

As can be seen, the form of equation (2.4) is exactly the same as the 1D Poisson equation shown above. The only difference is in the physical interpretation of $u$ and $f$ and that instead of the constant $\nu$ we use the constant $EA$ (Young's modulus $E$ times cross section area $A$) to describe the stiffness. The exact same mathematical problem could be interpreted as a heat conduction problem.

**Quiz question**

The mathematical problem above can be interpreted as the heat equation. In that case, what is the physical meaning of the provided boundary conditions?

What is the meaning of  if the strong form equation is interpreted as the heat equation?

Turn

Card 1 of 2

**Attribution**

This chapter was written by Frans van der Meer. Find out more here.

# From strong to weak form

## Contents


- Derivation of the weak form
- Boundary conditions


In the finite element method, before the problem is discretized, the governing equation is rewritten in the so-called **weak form**. In some cases, the weak form can be given a physical interpretation. In solid mechanics, for instance, one can interpret the weak form as an expression for a minimization problem of an energy potential. The weak form is then also referred to as *variational form* in the sense that in the minimization problem a solution is found for which variations in the potential are equal to zero. Here, however, we follow a formal route that arrives at the weak form by recasting the strong form as an integral equation without physical considerations.

## Derivation of the weak form

In general, the weak form of the problem is derived by multiplying the strong form by a test function, integrating over the domain, and then applying integration by parts on the terms containing derivatives[1]. In this manner, the order of derivatives appearing in the equations is reduced and the resulting form is symmetric (in general), facilitating a more efficient numerical solution.

Let's put this in practice for the rod equation (2.4). The first step is pre-multiplication of left hand side and right hand side with a test function $w(x)$ and integrating over the domain. Assuming constant $EA$, we get:

$$-\int_0^L wEA\frac{\partial^2 u}{\partial x^2}\,dx = \int_0^L wf\,dx, \quad \forall \quad w \tag{2.7}$$

The $\forall w$ means that the equality has to hold for all possible test functions $w$. Note that a solution of the *strong form* will indeed satisfy this equation for every function $w$. Conversely, for any $u$ that does not satisfy the strong form anywhere in the domain, it is possible to find a function $w$ for which equation (2.7) does not hold.

Next, integration by parts is used to get rid of the second order derivative in $u$:

$$\int_0^L \frac{\partial w}{\partial x} EA \frac{\partial u}{\partial x}\, dx - \left[ wEA \frac{\partial u}{\partial x} \right]_0^L = \int_0^L wf\, dx, \quad \forall \quad w \tag{2.8}$$

Note that after integrating by parts, a boundary term appears.

**Integration by parts**

Recall the definition of integration by parts in 1D

$$\int_0^L \alpha\beta'\, dx = [\alpha\beta]_0^L - \int_0^L \alpha'\beta\, dx$$

Where the $'$ is used to indicate a derivative with respect to $x$. In our case, we have $\alpha = w$ and $\beta = \partial u/\partial x$ and every term is multiplied with a constant $EA$.

Note that this integration by parts definition is related to the product rule of differentiation, which says:

$$(\alpha\beta)' = \alpha'\beta + \alpha\beta'$$

# Boundary conditions

We typically distinguish between two[2] types of boundary conditions:

- **Dirichlet** (or *essential*) boundary conditions, where we enforce the value of the solution.
- **Neumann** (or *natural*) boundary conditions, where we enforce the flux (or force).

In the case introduced in the previous section (see Fig. 2.7), one boundary condition of each type is specified:

- Dirichlet: $u = 0$ at $x = 0$
- Neumann: $EA\frac{\partial u}{\partial x} = F$ at $x = L$

To enforce Dirichlet boundary conditions we strongly[3] prescribe the value of the solution and set the test function to zero at that boundary. This effectively means that the boundary term coming from the integration by parts evaluated at the Dirichlet boundary goes to zero.

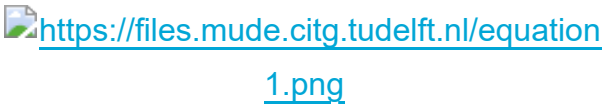
$$wEA\frac{\partial u}{\partial x}\bigg|_{x=0} = 0EA\frac{\partial u}{\partial x}\bigg|_{x=0} = 0 \tag{2.9}$$

To enforce the Neumann boundary condition we just need to replace the value of the prescribed flux (or force) in the boundary term appearing after the integration by parts. This type of boundary condition is often called *natural* as it naturally appears when deriving the weak form.

$$wEA\frac{\partial u}{\partial x}\bigg|_{x=L} = w(L)F. \tag{2.10}$$

**Strange formula?**

For some reason we don't fully understand the formula's above might show some strang colorred-statement. You can solve this by opening this page in a incognito window or by clearing your cache. The formula's should look like this:




$$wEA\frac{\partial u}{\partial x}\bigg|_{x=L} = w(L)F$$

Now we can replace equations (2.9) and (2.10) into the weak form (2.8), and send to the right-hand site the terms that do not depend on the unknown $u$. This leads us to the final weak form:

$$\int_0^L \frac{\partial w}{\partial x} EA\frac{\partial u}{\partial x}\, dx = \int_0^L wf\, dx + w(L)F, \quad \forall \quad w \tag{2.11}$$

**Quiz questions**

At what step in the derivation did the boundary term enter the formulation?

- Integration by parts
- Multiplication with test function
- Integration over the domain

**Exercise**

Show that the weak form can be derived to be exactly the same if $EA$ is not assumed to be constant (see (2.4) for the general strong form equation). If needed, look at the intermediate steps and solutions below:

1. What would be the equivalent of (2.7) for the case where $EA$ is not assumed to be constant?

**Solution**

We would have

$$-\int_0^L w\frac{\partial}{\partial x}\left(EA\frac{\partial u}{\partial x}\right)dx = \int_0^L wf\,dx, \quad \forall \quad w$$

2. Continuing with this integral expression what would be the result of applying integration by parts (i.e. what would be the equivalent of (2.8))?

**Solution**

We refer back to the formula of integration by parts

$$\int_0^L \alpha\beta'\,dx = [\alpha\beta]_0^L - \int_0^L \alpha'\beta\,dx$$

Now we have $\alpha = w$ and $\beta = EA\partial u/\partial x$, which gives exactly the same result as (2.8).

3. Check that the final weak form is indeed the same as (2.11).

**Solution**

The boundary condition $EA\frac{\partial u}{\partial x} = F$ at $x = L$ involves evaluation of both $EA$ and $\frac{\partial u}{\partial x}$ at $x = L$ exactly as is required in (2.8).

**Attribution**

This chapter was written by Frans van der Meer. Find out more here.

Note that this is not a general rule. There are cases of terms containing derivatives that might not be integrated by parts, typically non-symmetric terms.

We can also find Robin type of boundary conditions, which are a mix between Dirichlet and Neumann type, for example: $\alpha u + EA\frac{\partial u}{\partial x} = F$.

We can also enforce Dirichlet boundary conditions in a *weak* sense, by introducing additional terms on the weak form that penalize the difference between the solution and the prescribed value at the boundary. This will not be covered in this chapter.

# From weak to discrete form

## Contents


- Shape functions
- Discrete weak form


Finding the solution to the weak form (2.11) derived in the previous section is an infinite-dimensional problem, $u$ is unknown at every point $x$ in the domain and we also have an infinite number of possible weight functions $w$. In this section we will discretize the weak form such that it can be solved with a computer. With the aim to go from a continuous problem to a discrete problem, we approximate the solution by reducing the solution space to a finite number of unknowns and at the same time restricting the number of equation to a finite set.

## Shape functions

This is done by assuming a certain shape for $u(x)$ and restricting the set of functions $w(x)$ to all functions of that same shape.

Let's introduce a finite set of **shape functions** (or interpolation functions) $N_i(x)$ with $i \in [1, n]$, $n$ being the number of degrees of freedom (DOFs), and say that our approximate solution $u^h(x)$ is a linear combination of these shape functions:

$$u^h(x) = \sum_{i=1}^{n} N_i(x) u_i \tag{2.12}$$

where $u_i$ is the coefficient with which $N_i$ is multiplied. Note that $u_i$ is not a function of $x$, just a coefficient defined at each DOF.

In the Finite Element method, once we have the weak form at the continuous level, we define a *discretization* of the domain, that is a mesh with nodes and elements, see Fig. 2.8

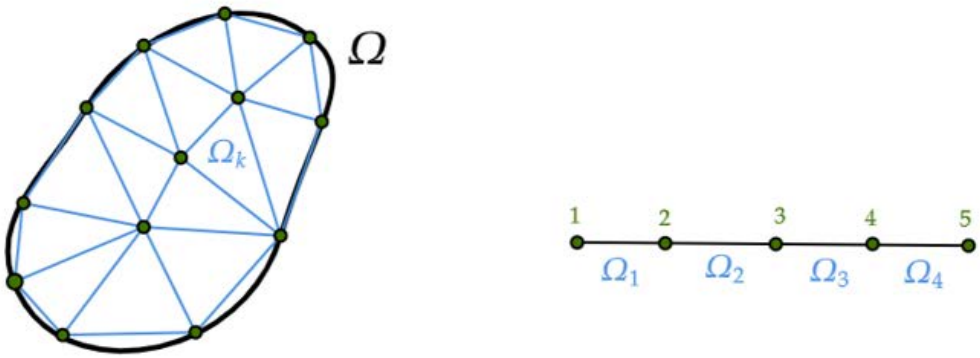



*Fig. 2.8* Triangulation of a 2D domain (left) and discretization of a 1D domain (right).

This mesh is used to define the shape functions with certain peculiarities:

- Every shape function $N_i$ belongs to a node with coordinates $x_i$.
- The shape function is equal to 1 at its own node and equal to zero at all other nodes.
- The shape function $N_i$ only takes values different from zero at the set of elements attached at to the $i$-th node, $S_i$.
- The shape function $N_i$ is defined by element-wise functions (typically polynomials)

$$N_i(x) = \begin{cases} p(x), & x \in \Omega_k \; \forall k \in S_i \\ 1, & x = x_i \\ 0, & \text{otherwise} \end{cases} \tag{2.13}$$

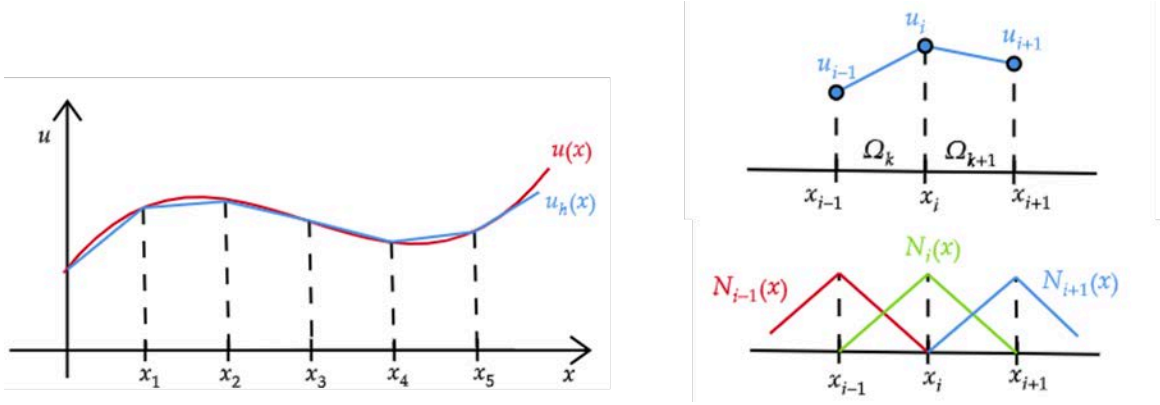


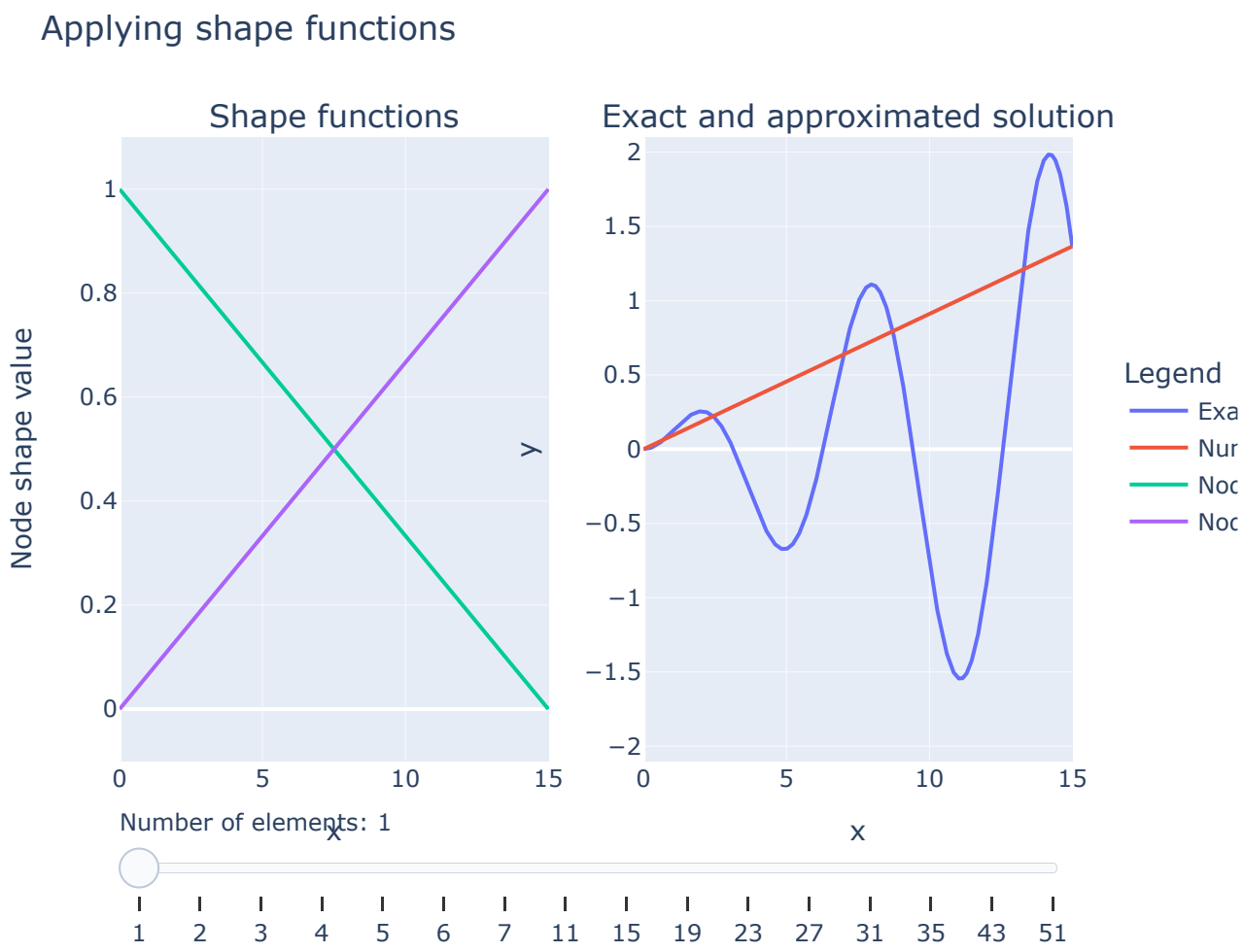

*Fig. 2.9* Interpolation of a function $u$ using piece-wise linear polynomials (left), approximated solution arround node $i$ (top-right) and shape functions for nodes $i-1$, $i$ and $i+1$ (bottom-right).

Note that the definition of the shape functions as in (2.13) has a useful property: the coefficent $u_i$ is equal to the solution at node $i$: $u^h(x_i) = u_i$.

By solving for $u^h$ instead of for $u$ our solution space is no longer infinitely dimensional, but has become $n$-dimensional. If we know the $u_i$ values for all $i$'s from 1 to $n$, using the

interpolation defined in (2.12), we know $u^h$ for every $x$ in our domain. We are making an approximation here, it may be impossible to reconstruct the true solution $u(x)$ exactly with the simple shape functions $N_i(x)$, but it can be shown that if we increase the number of nodes, the true $u(x)$ will be approximated with increased accuracy.

The following widget illustrates this concept.




**Note**

In the widget above, we are not solving anything yet. This is just an illustration of how increasing the resolution of the mesh allows for a better and better representation of the solution. For this plot, we are setting nodal values to the exact solution and using the shape functions to interpolate. In general, we will not know the exact solution and use the finite element method to find nodal values that give a good approximation of the PDE solution. The *good* nodal values that are not necessarily on top of the exact solution, but the idea that increasing the number of nodes leads to a better and better approximation does still hold.

# Discrete weak form

In the Finite Element method, we substitute the discretized solution $u^h$ into the weak form equation, as derived for our 1D-rod in equation (2.11). We are solving for $n$ values $u_j$. For this, we need to construct $n$ equations. Be reminded that with the "$\forall w$", we had an infinite number of equations. It turns out that it is useful to use the same shape functions $N_i(x)$ to pick $n$ relevant equations from this infinite set. In more technical terms, we select the same vectorial space for both the approximated solution and the test functions[1]. We replace the $w(x)$ with its discretized counterpart $w^h(x)$:

$$w^h(x) = \sum_{i=1}^{n} N_i(x) w_i$$

In alternative notation, we can collect all shape functions in a row vector or $1 \times n$ matrix:

$$\mathbf{N} = [N_1, N_2, \dots, N_n]$$

which gives

$$u^h = \mathbf{N}\mathbf{u} \quad \text{and} \quad w^h = \mathbf{N}\mathbf{w} \tag{2.14}$$

Similarly, we introduce $\mathbf{B}$ as a matrix containing the derivatives of the shape functions

$$\mathbf{B} = \left[ \frac{\partial N_1}{\partial x}, \frac{\partial N_2}{\partial x}, \dots, \frac{\partial N_n}{\partial x} \right]$$

which gives

$$\frac{\partial u^h}{\partial x} = \mathbf{B}\mathbf{u} \quad \text{and} \quad \frac{\partial w^h}{\partial x} = \mathbf{B}\mathbf{w} \tag{2.15}$$

Substitution of equations (2.14) and (2.15) into equation (2.11) gives

$$\int_0^L \mathbf{B}\mathbf{w} E A \mathbf{B}\mathbf{u} \, dx = \int_0^L \mathbf{N}\mathbf{w} f \, dx + \mathbf{N}(L)\mathbf{w} F$$

Note that $\mathbf{u}$ and $\mathbf{w}$ are not a function of $x$ and can therefore be taken out of the integrals

$$\mathbf{w}^T \int_0^L \mathbf{B}^T E A \mathbf{B}\, dx \mathbf{u} = \mathbf{w}^T \int_0^L \mathbf{N}^T f\, dx + \mathbf{w}^T \mathbf{N}^{\mathbf{T}}(L) F \tag{2.16}$$

where we have made use of the the equalities $\mathbf{B}\mathbf{w} = \mathbf{w}^T \mathbf{B}^T$ and $\mathbf{N}\mathbf{w} = \mathbf{w}^T \mathbf{N}^T$. Finally, we can divide by $\mathbf{w}^T$ and obtain an expression of the form

$$\mathbf{K}\mathbf{u} = \mathbf{f}$$

with

$$\mathbf{K} = \int_0^L \mathbf{B}^T E A \mathbf{B}\, dx$$

and

$$\mathbf{f} = \int_0^L \mathbf{N}^T f\, dx + \mathbf{N}^T(L) F$$

**Elimination of $\mathbf{w}^T$**

Note that the "division by $\mathbf{w}^T$" that was performed above is not really a valid mathematical operation. What we are actually doing in that step is described more rigorously in this box. Equation (2.16) must hold for all $\mathbf{w}$. If we collect all terms in equation (2.16) to the left hand side, it can be written in the form

$$\mathbf{w}^T \mathbf{v} = 0$$

You may recognize this as an inner product between two vectors. If the inner product $\mathbf{w}^T \mathbf{v}$ must be equal to 0 for all $\mathbf{w}$, it follows that

$$\mathbf{v} = \mathbf{0}$$

In making this step we go from a scalar equation to a vector equation. Specifically, we have

$$\mathbf{v} = \mathbf{K}\mathbf{u} - \mathbf{f} = \mathbf{0}$$

Now we have $n$ equations that are linear in the $n$ degrees of freedom in the vector $\mathbf{u}$.

**Exercise**

### Fill the blanks

For a 1D problem with 10 nodes, the size of $\mathbf{B}$ is _____ rows by _____ columns

Check

**Quiz question**

Where does the term $\mathbf{B}^T$ in the $\mathbf{K}$ matrix come from?

- $\mathbf{B}^T$ comes from the discretization of the test function
- $\mathbf{B}^T$ comes from the discretization of the solution

**Attribution**

This chapter was written by Frans van der Meer. Find out more here.

Selecting the same space for both solution and test functions is known as the *Bubnov-Galerkin* method. In some cases is useful to select different spaces for solution and trial functions, leading to the so called *Petrov-Galerkin* methods, which will not be covered in this chapter.

# Finite element implementation

## Contents


- Elemental matrix and vector
- Global matrix and vector
- Boundary conditions
- Additional case with non-uniform stiffness


In equation (2.16) we still had a global integral and a vector including all (derivatives of) shape functions. However, we can make use of the fact that every individual shape function (and hence its derivatives) is equal to zero for the largest part of the domain, only different from zero at the set of elements attach to the corresponding node. Moreover, we can always split the integral in a sum of integrals over subdomains. Then, since the piece-wise polynomials are defined on each element $\Omega_k$, it is useful to select these subdomains as the **elements** of the mesh, see Fig. 2.8.

## Elemental matrix and vector

To begin, we redefine the global integral over the global domain $\Omega$ as a sum of integrals over the domain of all individual elements $\Omega_k$.

$$\int_{\Omega} \mathbf{B}^T E A \mathbf{B} d\Omega = \sum_{k=1}^{n_{\text{elements}}} \int_{\Omega_k} \mathbf{B}^T E A \mathbf{B} d\Omega$$

We will call the matrix contribution from an individual element, the *element matrix*. To evaluate the element matrix, we need to consider only those shape functions that are active in the element.

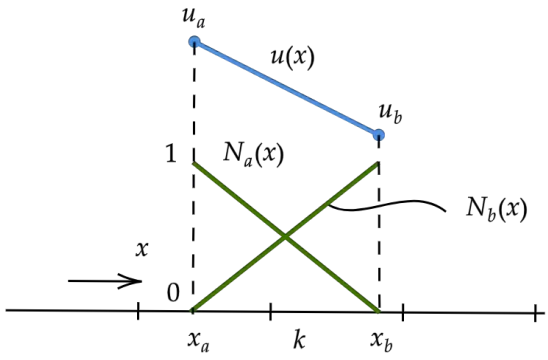



*Fig. 2.10* Linear two-node element: element $k$ defined on the domain from $x_a$ to $x_b$

Using piece-wise linear shape functions, we have that for a given element $k$, the shape functions associated to nodes $a$ and $b$ are:

$$N_a(x) = \frac{x_b - x}{x_b - x_a} = \frac{x_b - x}{\Delta x}, \quad N_b(x) = \frac{x - x_a}{x_b - x_a} = \frac{x - x_a}{\Delta x},$$

where $x_a$ and $x_b$ are the $x$-coordinates of the two nodes of the element.

This elemental shape function matrix $\mathbf{N} = [N_a \ N_b]$ is defined below as a python function. It is convenient to work with local coordinates, which can be generalized to the concept of iso-parametric mapping. If you are curious how this is generally done in finite elements, there is additional material on numerical integration and iso-parametric mapping in the Videos chapter. Here, we work with local coordinates $x_{\text{local}} = x - x_a$:

```
# Returns the evaluated N matrix
# - The local coordinate of evaluation "x_local"
# - The element size "dx"
def evaluate_N(x_local, dx):
    return np.array([[1-x_local/dx, x_local/dx]])
```

We can also define the shape functions derivative, which for piece-wise linear functions is:

$$\frac{\partial N_a}{\partial x}(x) = \frac{-1}{\Delta x}, \quad \frac{\partial N_b}{\partial x}(x) = \frac{1}{\Delta x}.$$

If we introduce the $\mathbf{B}$ matrix (size $1 \times 2$) to collect the derivatives of the shape functions in the element, we can write the element matrix as:

$$\mathbf{K}_e = \int_{x_a}^{x_b} \mathbf{B}^T E A \mathbf{B} \, dx.$$

For the 1D element with linear shape functions, $\mathbf{B}$ is defined as:

$$\mathbf{B} = \begin{bmatrix} \frac{-1}{\Delta x} & \frac{1}{\Delta x} \end{bmatrix}$$

with $\Delta x = x_b - x_a$. Note that for the linear shape functions, $\mathbf{B}$ is constant over the element. In general, that is not always the case.

```
# Returns the evaluated B vector
# - The local coordinate of evaluation "x_local"
# - The element size "dx"
def evaluate_B(x_local, dx):
    return np.array([[-1/dx, 1/dx]])
```

Assuming a constant $EA$, evaluating the integrand over the element domain results in:

$$\mathbf{K}_e = \frac{EA}{\Delta x} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{2.17}$$

In practice, finite element implementations make use of numerical integration to evaluate integrals over the element:

$$\int_{\Omega_k} \phi(x) dx \approx \sum_{i=1}^{n_{ip}} w_i \phi(x_i).$$

For this specific case numerical integration is not necessary because we integrate a constant. However, we follow the standard finite element workflow here. The equation above requires definition of an integration scheme (i.e. a set of $n_{ip}$ coordinates $x_i$ and integration weights $w_i$).

```

```

```
# Returning the matrix of a single element
# - The stiffness of the rod "EA"
# - The length of an element "dx"
def get_element_matrix(EA, dx):

    # Defining integration locations and weights
    integration_locations = [(dx - dx/(3**0.5))/2, (dx + dx/(3**0.5))/2]
    integration_weights = [dx/2, dx/2]
    n_ip = len(integration_weights)

    # Setting up the local element matrix
    n_node = 2
    K_loc = np.zeros((n_node,n_node))

    # Evaluation of the matrix in a loop over integration points
    for x_i, w_i in zip(integration_locations, integration_weights):
        B = evaluate_B(x_i, dx)
        K_loc += EA*np.dot(np.transpose(B), B)*w_i

    return K_loc
```

Similarly, the integral in the right hand side vector needs to be evaluated. For a constant distributed load $f(x) = q$, we get:

$$\mathbf{f}_e = \int_{x_a}^{x_b} \mathbf{N}^T f \, dx \approx \sum_{i=1}^{n_{ip}} w_i \mathbf{N}^T(x_i) q$$

```
# Returning the force vector of a single element (distrubted load only)
# - The magnitude of constant distributed load "q"
# - The length of the element "dx"
def get_element_force(q, dx):
    # Creating a matrix with the required size
    n_node = 2
    N = np.zeros((1,n_node))    # Defining in 2 dimensions is nessecary for transpose

    # Defining integration locations and weights
    integration_locations = [(dx - dx/(3**0.5))/2, (dx + dx/(3**0.5))/2]
    integration_weights = [dx/2, dx/2]

    # Setting up the local element force vector
    f_loc = np.zeros((n_node,1))

    for x_i, w_i in zip(integration_locations, integration_weights):
        N = evaluate_N(x_i,dx)
        f_loc += np.transpose(N)*q*w_i

    return f_loc
```

# Global matrix and vector

Once each elemental matrix is defined, we need to assemble the elemental contributions to the corresponding entries of the global matrix.

To illustrate this step, we consider a discretization of the rod with three elements of equal length. We give each element and node a number, there are 4 nodes, numbered $n_1$ to $n_4$ and 3 elements numbered $e_1$ to $e_3$, see Fig. 2.11.

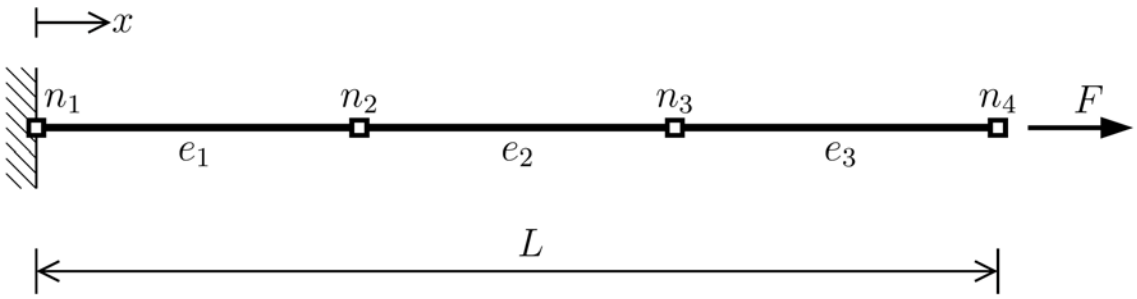



*Fig. 2.11* One dimensional rod discretized into 3 elements

By a simple addition of the influences of each element we can find the total $\mathbf{K}$ matrix:

$$\mathbf{K} = \frac{EA}{\Delta x}\left(\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}\right) \tag{2.18}$$

After assembly of all the local matrices, we have the global $\mathbf{K}$ matrix:

$$\mathbf{K} = \frac{EA}{\Delta x}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{2.19}$$

In practice it is very inefficient to compute the element matrices at the same size as the global matrices. You can see in the operation above quite some zeros are added every time an element matrix is added. It is better to evaluate smaller element matrices (as in equation (2.17)) that only contain the nonzero values and then put in the right position in an assembly operation. Some bookkeeping is needed to know which nodes (or degrees of freedom) belong to which elements. For our 1D mesh, this is very simple, as reflected by the function below.

```
def get_nodes_for_element(ie):
    return np.array([ie,ie+1])
```

For putting the element matrix in the right position, we make use of the `numpy.ix_` function. A naive implementation of the assembly operation would be to do:

```
    elnodes = get_nodes_for_element(i)
    K_local = get_element_matrix(EA, dx)
    for k in range(2):
        for l in range(2):
            K_global[elnodes[k], elnodes[l]] += K_local[k, l]
```

However, this double loop can be vectorized with the `numpy` function `ix_`, where the function `ix_` is used to access the relevant submatrix from the global matrix:

```
    elnodes = get_nodes_for_element(i)
    K_global[np.ix_(elnodes, elnodes)] += get_element_matrix(EA, dx)
```

```
# - The length of the rod "rod_length"
# - The number of elements "n_el"
# - The stiffness of the rod "EA"
def assemble_global_K(rod_length, n_el, EA):
    n_DOF = n_el+1
    dx = rod_length/n_el
    K_global = np.zeros((n_DOF, n_DOF))

    for i in range(n_el):
        elnodes = get_nodes_for_element(i)
        K_global[np.ix_(elnodes,elnodes)] += get_element_matrix(EA, dx)

    return K_global
```

A similar function is defined for assembling the right hand side vector.

```
# Returns the global f vector with continuous forces on a rod
# - The length of the rod "rod_length"
# - The number of elements "n_el"
# - The force on the rod "EA"
def assemble_global_f(rod_length, n_el, q):
    n_DOF = n_el+1
    dx = rod_length/n_el
    f_global = np.zeros((n_DOF,1))

    for i in range(n_el):
        elnodes = get_nodes_for_element(i)
        f_global[elnodes] += get_element_force(q, dx)

    return np.squeeze(f_global)
```

# Boundary conditions

We have now derived the relation between $\mathbf{u}$ and $\mathbf{f}$ through a matrix $\mathbf{K}$. However, when we try to solve the equation $\mathbf{Ku} = \mathbf{f}$, we will run into a problem. On the left hand side one of the values for $u$ is already known, the displacement of the left node ($n_1$) will be zero. We will keep the discussion a bit more general here and use the symbol $\bar{u}$ for the prescribed values at $x = 0$. The corresponding force $f_1$ is as of yet unknown. Since there is unknowns on either side of the equation we cannot solve it by inverting $\mathbf{K}$ and bringing it to the other side.

$$\frac{EA}{\Delta x}\begin{bmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} \bar{u} \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ q\Delta x \\ q\Delta x \\ \frac{1}{2}q\Delta x + F \end{bmatrix} \tag{2.20}$$

A way to overcome this problem is to eliminate equations belonging to nodes for which the solution is known. As long as the prescribed degrees of freedom have a value of 0 (i.e. homogeneous Dirichlet conditions), we can do this by simply removing the corresponding columns and rows out of our system. In general, however, when $\bar{u} \neq 0$ we also need to modify the right hand side so as to not change the other equations. Doing that for node 1 in our example, we get:

$$\frac{EA}{\Delta x}\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}\begin{bmatrix} u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} q\Delta x + \frac{EA}{\Delta x}\bar{u} \\ q\Delta x \\ \frac{1}{2}q\Delta x + F \end{bmatrix} \tag{2.21}$$

**Elimination of Dirchlet boundary conditions**

In a general scenario where multiple nodal values are prescribed. A subdivision of the matrix in free and prescribed dofs is performed. Using subscript $f$ for free dofs and $p$ for prescribed dofs, we can write the partition the complete system of equations as:

$$\begin{bmatrix} \mathbf{K}_{pp} & \mathbf{K}_{pf} \\ \mathbf{K}_{fp} & \mathbf{K}_{ff} \end{bmatrix} \begin{bmatrix} \mathbf{u}_p \\ \mathbf{u}_f \end{bmatrix} = \begin{bmatrix} \mathbf{f}_p \\ \mathbf{f}_f \end{bmatrix}$$

Excluding the equations associated with $\mathbf{f}_p$ and moving the known values for $\mathbf{u}_p$ in the remaining equations to the right hand side, the reduced system of equations becomes:

$$\mathbf{K}_{ff}\mathbf{u}_f = \mathbf{f}_f - \mathbf{K}_{fp}\mathbf{u}_p$$

***Exercise***: check that the modification to the right hand side of equation (2.21) is equivalent to the general formulation given here

This system can solved by inverting the matrix on the left hand side or, more efficiently, by using an appropriate linear solver. Solving the system of equations will finally result in a vector with values for the nodal displacements $\mathbf{u}$. Note that with the shape functions, we also have an interpolation between the nodes, so a continuous approximate solution $u^h(x)$ can be obtained with:

$$u^h(x) = \mathbf{N}\mathbf{u} \tag{2.22}$$

Moreover, the reaction force $f_1$ can be computed by multiplying first row from the complete $4 \times 4$-matrix with the nodal displacement vector.

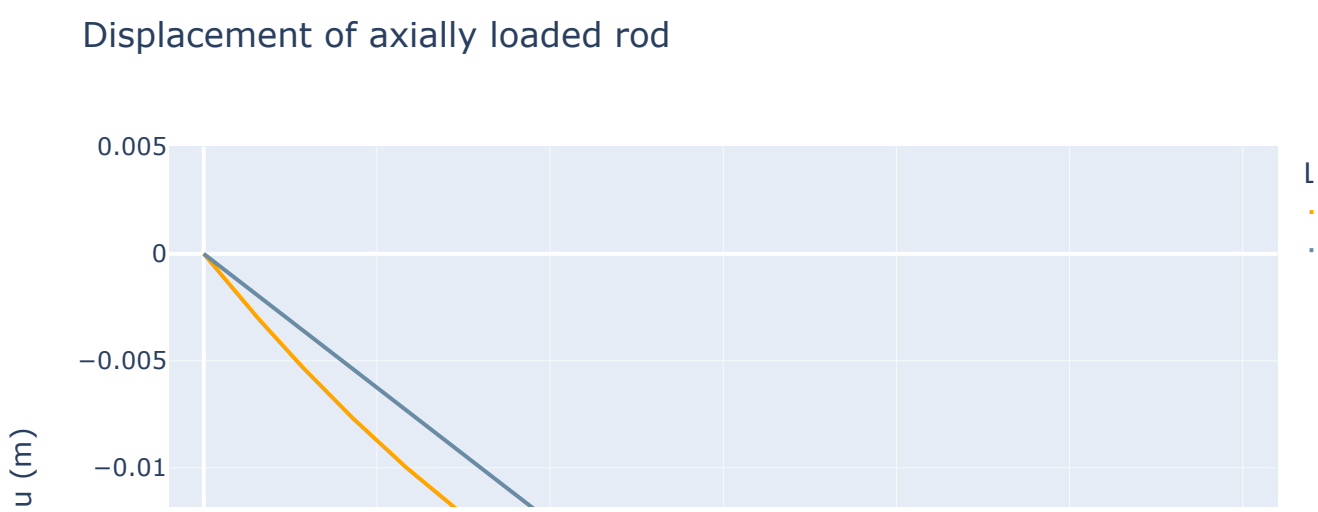
The function `simulate` returns the solution of the problem given the number of discrete elements. At this point we also need to give numbers to the parameters of the problem. We will use $L = 3$ m, $EA = 10^3$ N, $F = 10$ N and $q = -10$ N/m.

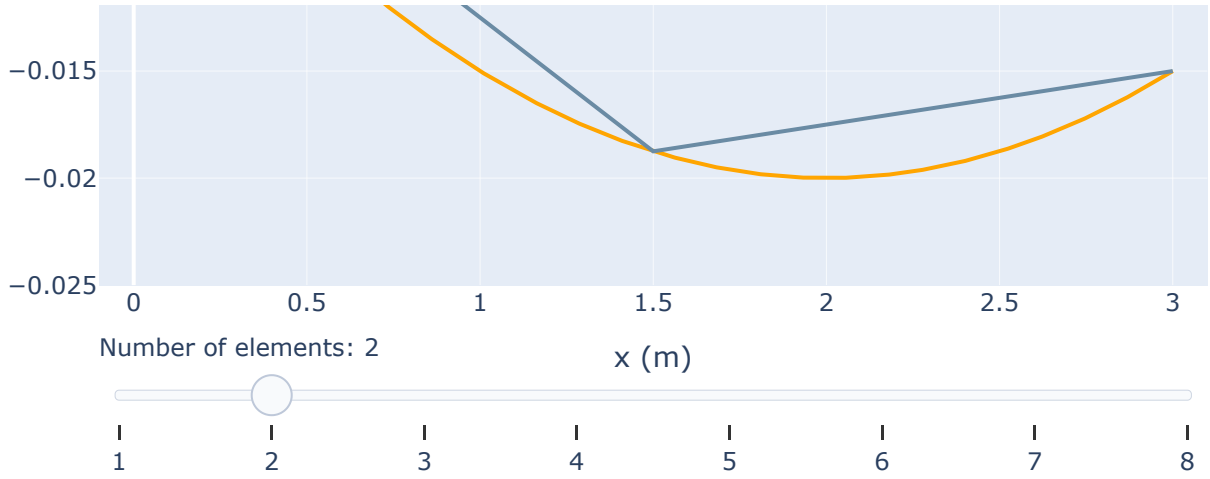
```
def simulate(n_element):
    length = 3              # Length
    EA = 1e3                # Stiffness EA
    n_node = n_element + 1  # Number of nodes
    F_right = 10            # Load applied at the right boundary
    u_left = 0              # Displacement at the left boundary
    q_load = -10            # Distibuted load

    dx = length/n_element
    x = np.linspace(0,length,n_node)

    # Assmemble K for the rod problem
    K = assemble_global_K(length, n_element, EA)

    # Assemble f first with distributed load only
    f = assemble_global_f(length, n_element, q_load)

    # Add boundary conditions to f
    f[n_node-1] += F_right

    # Initialize vector u with displacements
    u = np.zeros(n_node)

    # Solve the system for unknown displacements
    K_inv = np.linalg.inv(K[1:n_node, 1:n_node])
    u[1:n_node] = np.dot(K_inv, f[1:n_node])

    return x, u
```

Finally, we can run the simulation and inspect the results:

```
import numpy as np
import matplotlib.pyplot as plt

x, u = simulate(8)
plt.figure()
plt.plot(x,u)
```

In the following widget you can see how the solution converges to the analytical solution when we increase the number of elements.






**Quiz question**

So far the nodes are uniformly spaced. The finite element method can easily deal with unstructured meshes. The 1D equivalent of that is a mesh with uneven spacing. Suppose we change the position of the nodes (e.g. instead of $x = [0, 1, 2, 3]$ we use $x = [0, 0.5, 1.5, 3]$), what will happen to the system of equations $\mathbf{Ku} = \mathbf{f}$?

- [ ] The values in $\mathbf{K}$ change
- [ ] The values in $\mathbf{f}$ change
- [ ] The size of $\mathbf{K}$ and $\mathbf{f}$ changes

Check

**Exercise**

In which functions is the numerical integration procedure implemented? (Hint: there are two)

**Solution**

In the functions `get_element_force` and `get_element_matrix`

In these functions, which variables contain information on the integration scheme?

**Solution**

The variables `integration_locations` and `integration_weights`

How many integration points are employed in these functions?

**Solution**

Two integration points (per element)

For going to an integration scheme with a single integration point per element, how would you define these two variables?

**Solution**

For 1 point Gauss integration, the integration point needs to be at the center of the element and the integration weight needs to equate the length of the element:

```
integration_locations = [ dx/2 ]
```

```
integration_weights = [ dx ]
```

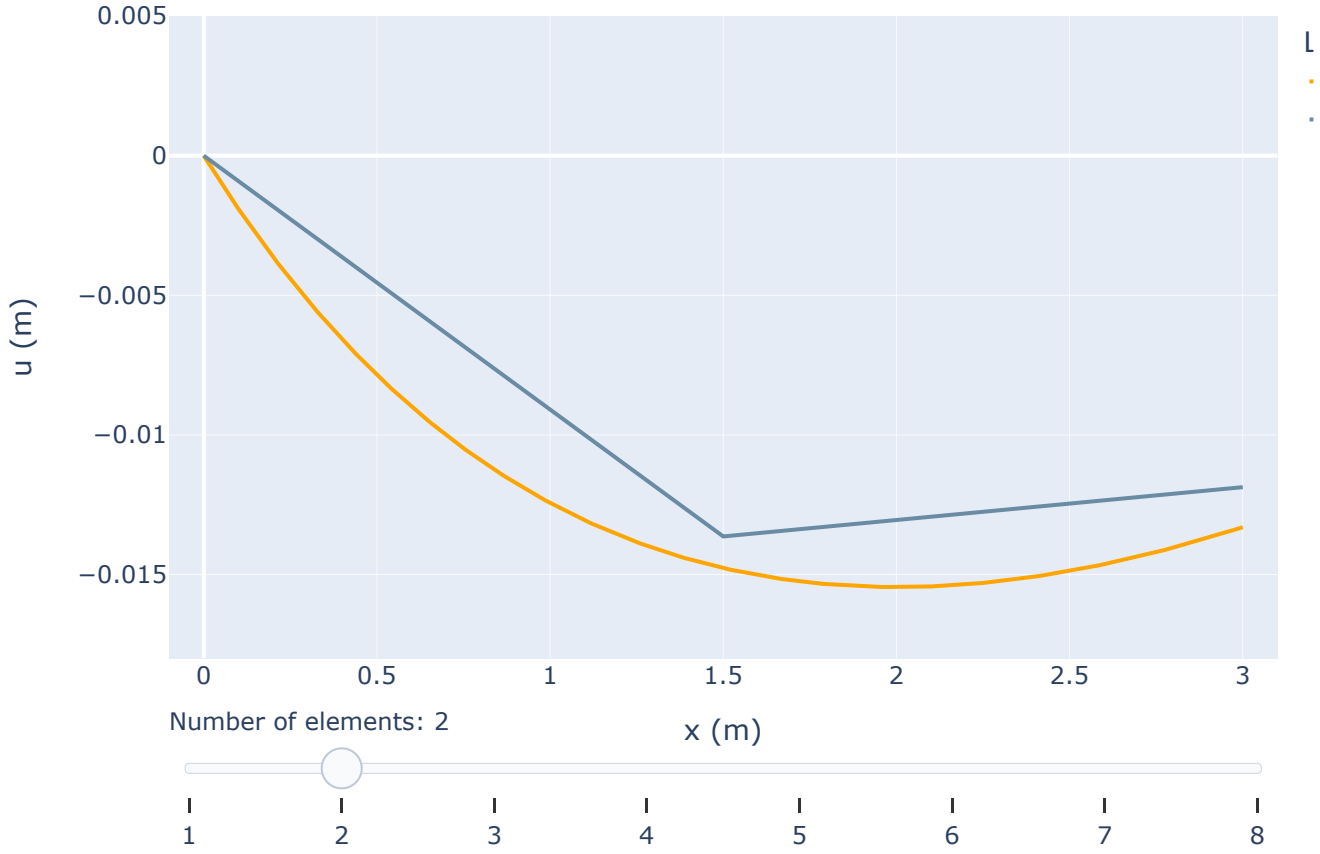
# Additional case with non-uniform stiffness

The problem above has one peculiar feature, which is that the nodal values always coincide with the exact solution. This is not a property of the finite element method. For completeness, we show the convergence behavior for a slightly more complicated problem, namely one where $EA$ is a linear function of $x$:

$$EA = 1 \cdot 10^3 (1 + \alpha x)$$

with $\alpha = 0.5\ \mathrm{m}^{-1}$. With the same boundary conditions, we get a convergence behavior is more typical for what one can expect from finite element solutions. The finite element model still converges to the exact result, but the approximate nature of the solution is now not only visible for the field inside the elements but also for the nodal values themselves.

Displacement of axially loaded rod with non-uniform EA




Number of elements: 2

1 2 3 4 5 6 7 8

# Elements and shape functions

## Contents


- Recap of linear shape function
- Shape function properties
- Higher order shape functions
- Finite elements shape functions in two dimensions
- Shape functions on a triangular element
- Shape functions on a quadrilateral element
- Shape functions in 3D


So far you have seen simple linear shape functions and elements that are defined over a 1D subdomain with two nodes. Most applications of the finite element method are not in 1D but in 2D or 3D. In higher dimensional space, the same key concepts can be used. The central idea is that the solution space is reduced with the introduction of shape functions. This can also be done for 2D or 3D fields.

Another important feature of the finite element method is that it allows for higher order approximations. In finite difference and finite volume schemes it can be shown that their spatial discretization is second order accurate, i.e. there is an error in the solution that is of the order ($\Delta x^2$). With the concept of shape functions it is possible to improve the order of accuracy of the spatial discretization.

In summary, there are two directions in which we can generalize the finite element method by choosing different shape functions

- Increasing the dimensionality
- Increasing the interpolation order

In this section, we will start with the second of these.

2_2_2 The finite element method: shape functions and ele...

# Recap of linear shape function

The linear shape functions are visualized once more in Fig. 2.12. They can be interpreted as global functions (left image), but are typically evaluated and implemented element-by-element (right image). In the figure, there are five nodes and four elements. The two shape functions in every element are each equal to 1 at one of the nodes and equal to 0 at the others.

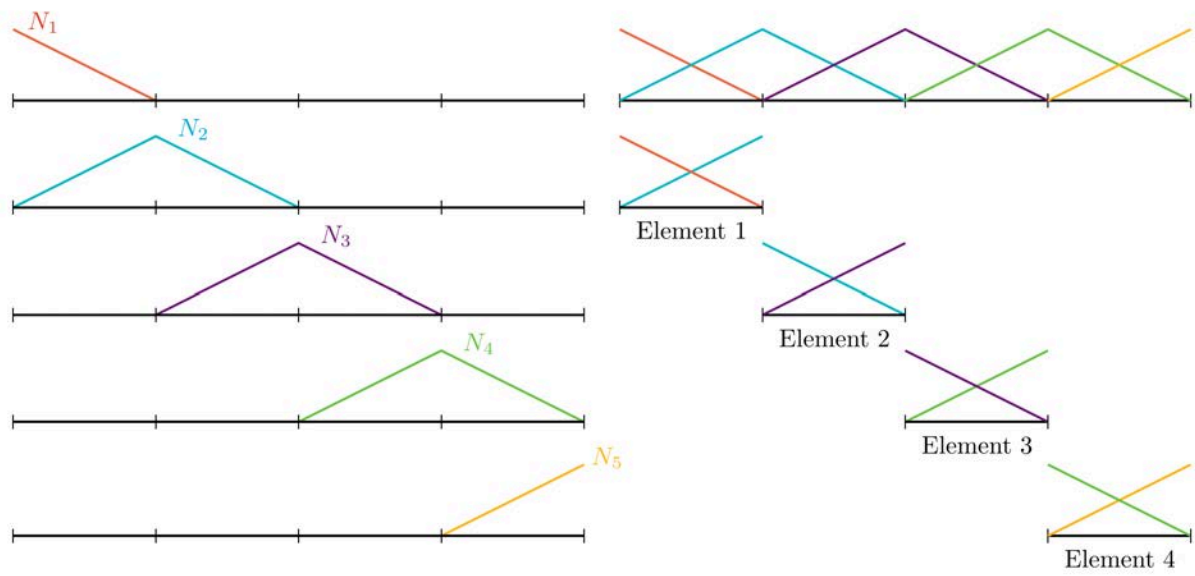



*Fig. 2.12* Linear shape functions as global functions (left) and element-by-element (right)

Inside the element, each of the two shape functions takes the form

$$N_i = a_i + b_i x$$

With the coordinates of the nodes it is possible to find the coefficients $a_i$ and $b_i$ for each shape function that makes it go from 0 to 1 over the element domain. However, as you have seen in the implementation, they are often evaluated in a local coordinate frame.

# Shape function properties

The linear shape functions we have discussed so far have two properties that hold for most finite element shape functions:

**Partition of unity**

$$\sum N_i(x) = 1$$

The sum of the shape functions is equal to one at every position inside the element. This is essential for ability to describe trivial constant solutions exactly.

**Kronecker delta property**

$$N_i(x_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Every shape function is equal to one at one of the nodes and equal to zero at all other nodes. This is helpful for the interpretation of the degrees of freedom of the solution vector as nodal values.

# Higher order shape functions

To improve the quality of the approximation, it is possible to introduce more nodes. The solution will converge to the exact solution. However, it is possible to make it converge faster and that is by increasing the polynomial order of the shape functions. Similar to the linear shape functions, quadratic shape functions can be introduced, see Fig. 2.13. The approximate solution is then not piecewise linear as we have seen so far, but piecewise quadratic.

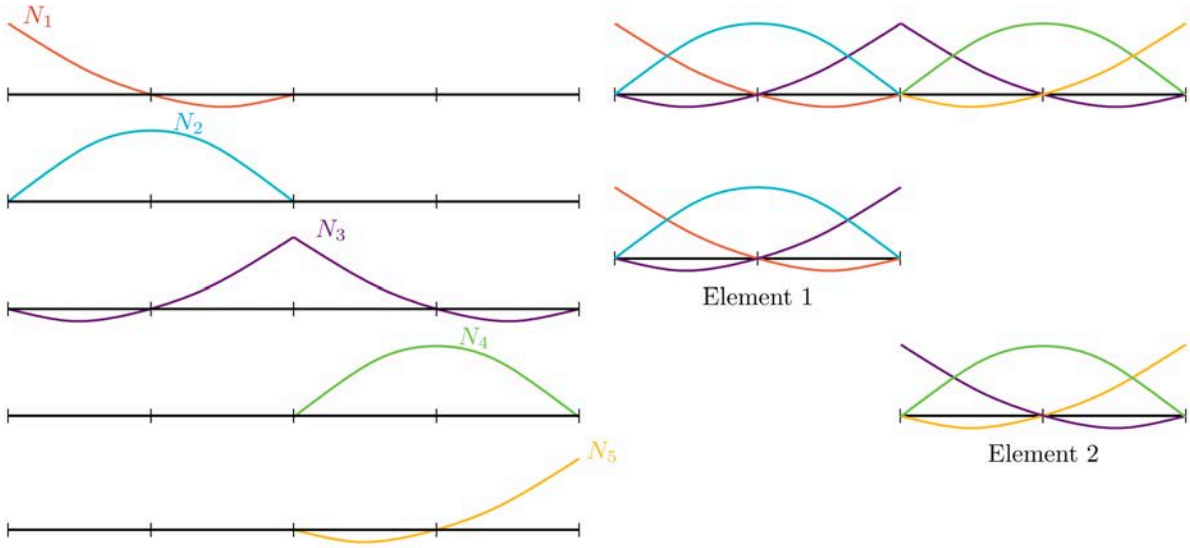



*Fig. 2.13* Quadratic shape functions as global functions (left) and element-by-element (right)

In the 1D discretization of 5 nodes, now 2 elements can be identified with 3 nodes each. Inside the elements three parabolic functions are defined that still have the property that they each are equal to 1 at one of the nodes and equal to 0 at the other nodes. This can be written compactly as

The general form of the quadratic shape functions is

$$N_i = a_i + b_i x + c_i x^2$$

You may notice that the Kronecker delta property gives three equations for each shape function (e.g. for $N_1$ we have $N_1(x_1) = 1$, $N_1(x_2) = 0$ and $N_1(x_3) = 0$). For given coordinates $x_1, x_2, x_3$, the coefficients $a_i, b_i, c_i$ are uniquely determined for each of the three shape functions.

# Finite elements shape functions in two dimensions

For 2D problems we seek the solution of a PDE defined over a two-dimensional domain $u(x, y)$. Shape functions still play a central role: the solution space is restricted by stating that it must be a linear combination of a set of shape functions. Now, the shape functions are functions defined over the two-dimensional domain. The approximate solution $u^h(x, y)$ is

constructed by interpolating the nodal values $u_i$ with their corresponding shape function $N_i(x, y)$:

$$u^h(x, y) = \sum_{i=1}^{n_{\text{nodes}}} N_i(x, y) u_i$$

Again, the nodal values $u_i$ are sought that provides a solution $u^h$ that best satisfy the governing equation by substituting the discretized solution into the weak form equation.

For two-dimensional analysis, finite element meshes of usually consist in quadrilaterals and/or triangles. Triangular elements are superior for constructing unstructured meshes on complex geometries, whereas quadrilaterals are useful for structured meshes. In both cases, linear and quadratic versions exist.

# Shape functions on a triangular element

The simplest 2D element is a 3-node triangle (see Fig. 2.14). The shape function of the 3-node triangle is defined as:

$$N_i = a_i + b_i x + c_i y$$

where the coefficients $a_i$, $b_i$ and $c_i$ depend on the coordinates of the element. Again, for given coordinates, the Kronecker delta properties defines what the values of the coefficients should be for the shape functions of an element.

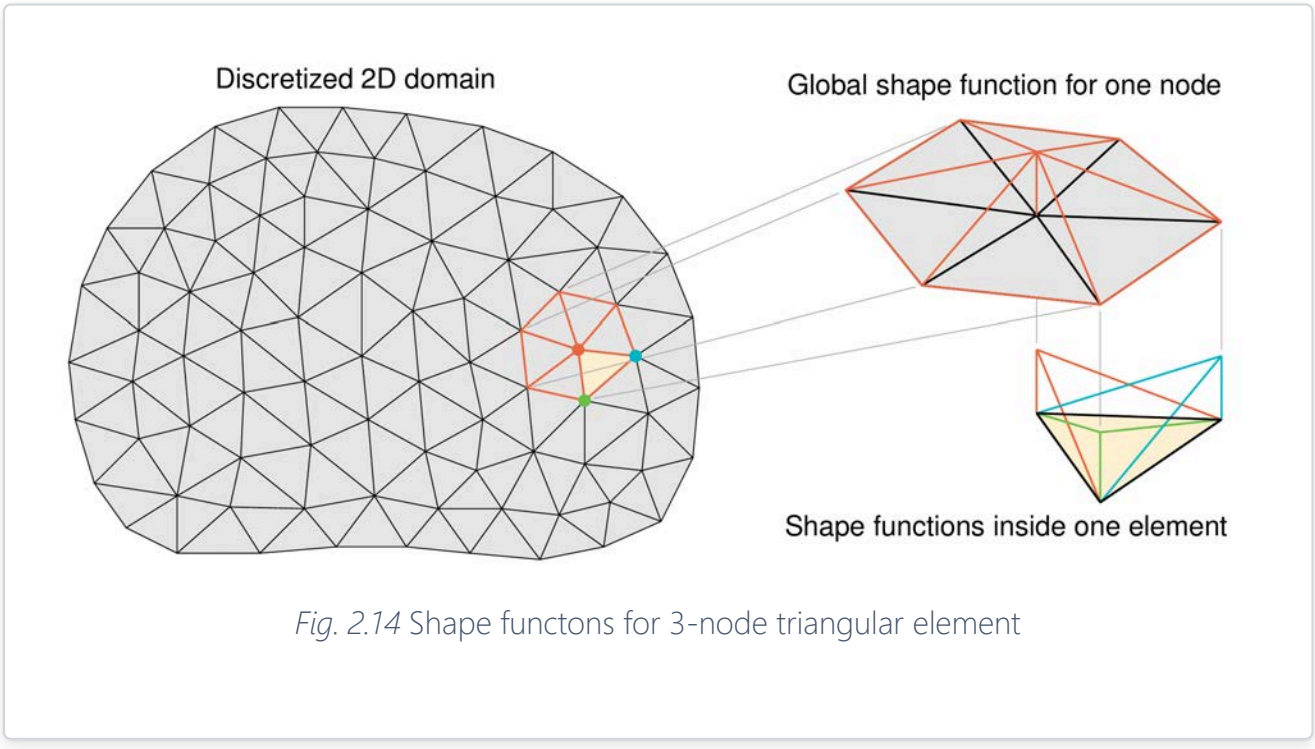



*Fig. 2.14* Shape functons for 3-node triangular element

Triangles with quadratic shape functions have 6 nodes, 3 at each corner and 3 at the middle of each edge. The shape functions are written as:

$$N_i = a_i + b_i x + c_i y + d_i x^2 + e_i xy + f_i y^2$$

# Shape functions on a quadrilateral element

Shape functions on quadrilateral elements can be understood as the product of 1D shape functions in the two principal directions of the element. The lowest order quadrilateral element has 4 nodes, one at each corner of the element. The shape function are obtained by multiplying linear shape functions in $x$ direction with linear shape functions in $y$ direction. The resulting shape functions are *bilinear*, the are linear in the coordinates except for a cross term $xy$. In general the shape functions on the 4-node element take the shape:

$$N_i = a_i + b_i x + c_i y + d_i xy$$

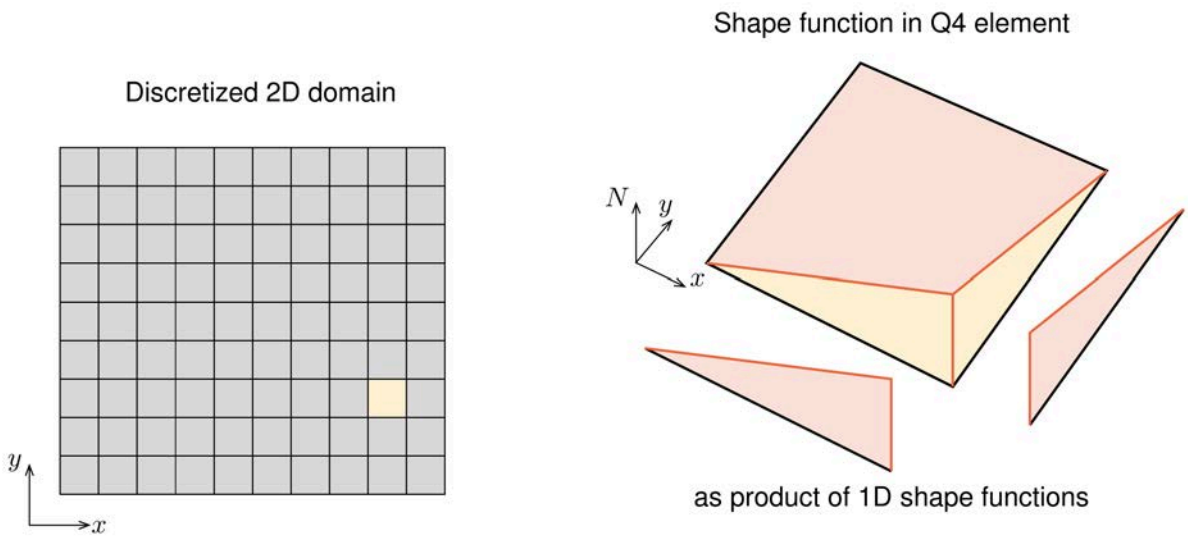



*Fig. 2.15* The 4-node quadrilateral element

Quadratic shape functions for a 9-node quadrilateral element can be constructed by multiplying the shape functions of the 3-node line element in two different directions. The shape function is expressed as:

$$N_i = a_i + b_i x + c_i y + d_i x^2 + e_i xy + f_i y^2 + g_i x^2 y + h_i x y^2 + j_i x^2 y^2$$

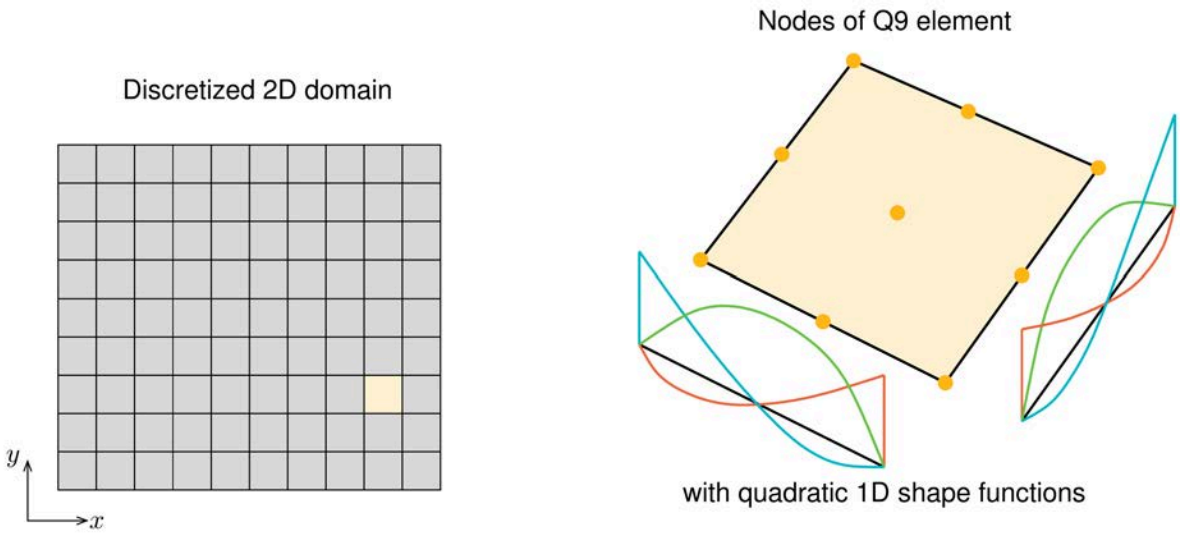



*Fig. 2.16* The 9-node quadrilateral element

A popular alternative is one where the center node is removed, resulting in an 8-node quadrilateral element with the following shape function:

$$N_i = a_i + b_i x + c_i y + d_i x^2 + e_i xy + f_i y^2 + g_i x^2 y + h_i x y^2$$

# Shape functions in 3D

For 3D problems, the solution is defined over a three-dimensional domain. The shape functions are as well. The mesh is then constructed with tetrahedral or hexehedral elements. The 4-node tetrahedral element has shape functions that are a generalization of the triangle with only linear terms:

$$N_i = a_i + b_i x + c_i y + d_i z$$

The hexahedral element is like a 3D version of the quadrilateral element. The 8-node hexahedral element has mixed terms and is thus called *trilinear*:

$$N_i = a_i + b_i x + c_i y + d_i z + e_i xy + f_i xz + g_i yz + h_i xyz$$

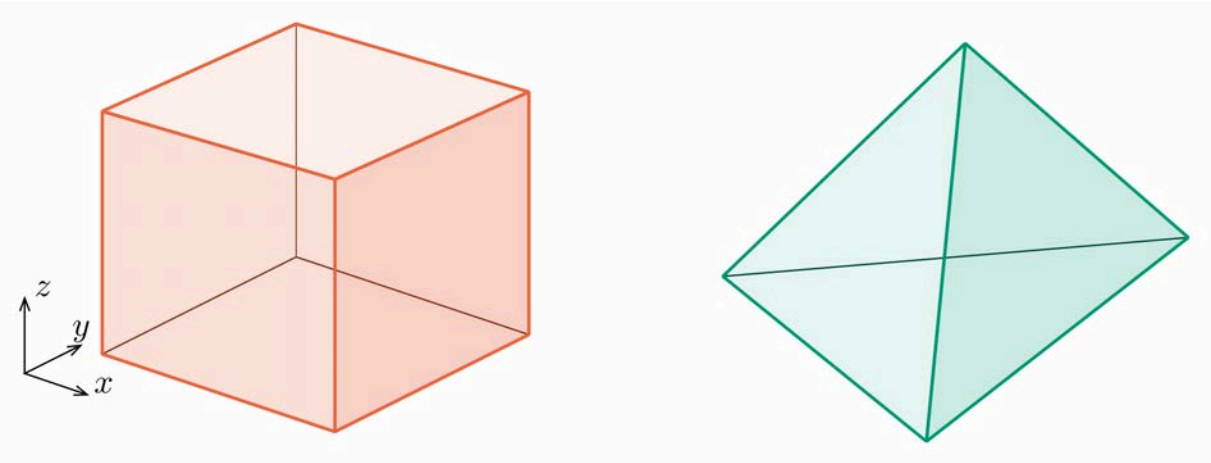
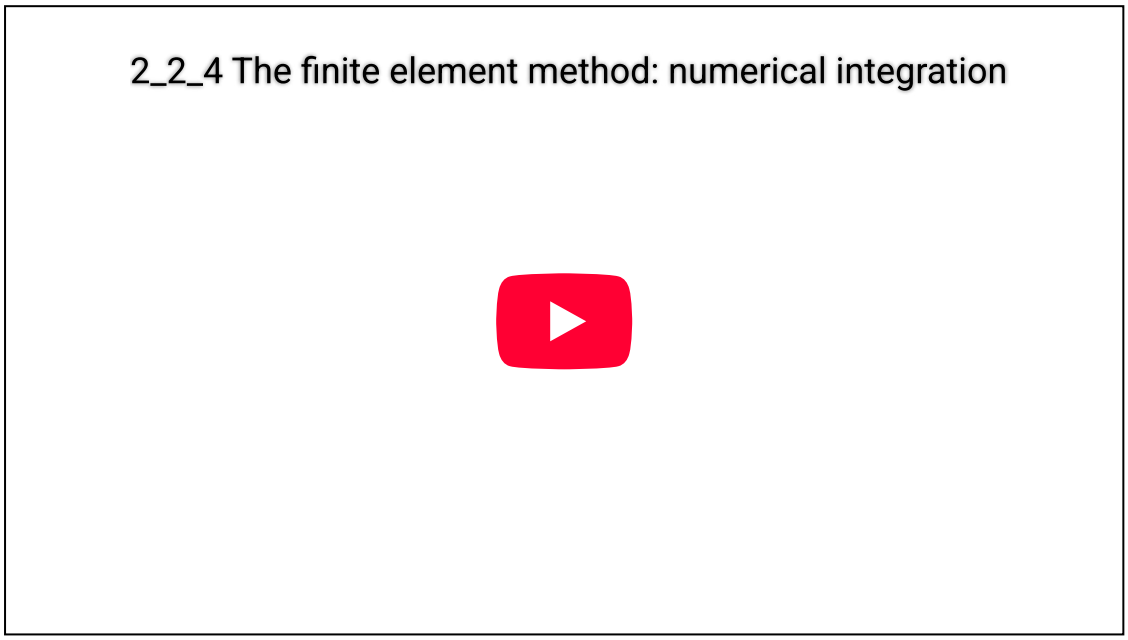



*Fig. 2.17* 3D elements: hexahedral (left) and tetrahedral (right)

> **Attribution**
>
> This chapter was written by Frans van der Meer. Find out more here.

# Numerical integration

In the discrete equations of the finite element method, integrals appear. These need to be evualated




For example in the 1D Poisson equation it is possible to perform this analytically, as follows.

$$\mathbf{K}_e = \int_{x_1}^{x_2} \mathbf{B}^T E A \mathbf{B}\, dx$$

$$\mathbf{B} = \begin{bmatrix} B1 & B2 \end{bmatrix}$$

$$B1 = \frac{1}{x_2 - x_1}, B2 = \frac{-1}{x_2 - x_1}$$

$$\mathbf{K}_e = \frac{v}{x_2 - x_1} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

In practice, however, numerical integration is performed. The idea behind numerical integration is that an integral can be replaced by a weighted sum, as follows:

$$\mathbf{K}_e = \int_{\Omega^\varepsilon} f(x,y) d\Omega \approx \sum_{i=1}^{n_{ip}} w_i f(x_i, y_i)$$

The function $f$ can be evaluated at a selected number of points, with coordinates $(x_i, y_i)$ and multiplied by weights $w_i$. The next step involves considering how to perform efficient and accurate numerical integration. One first requirement that arises is that the sum of all weights within an element should be equal to the integration domain, in this case the length of the element. Secondly, the number and location of the integration points must be specified, as well as the individual weights.

The following two integration schemes are relevant for Finite Element Analysis:

- Newton-Cotes
- Gauss Integration

A Newton-Cotes scheme uses equally spaced integration points. In this scheme, with the appropriate set of weights, $n + 1$ integration points are needed to integrate a $n$-th order polynomial exactly. Gauss integration (or *Gauss quadrature*) is more common in finite element analysis. This defines integration locations and weights such that they are optimal in the sense that polynomial functions can be integrated exactly with a minumum number of integration points.

Let's consider a reference element defined from -1 to +1 in a local $\xi$-coordinate. In the Gauss integration scheme the position and weights are optimised for exactly integrating polynomials to as high order as possible.Let's now look at an example.

- For a O-th order polynomial ($f = c$) the position of the integration point is not important, as long as the weight is equal to the length of the domain, which in this case is 2.
- For a 1-st order polynomial ($f = b\xi + c$), still we can be exact with one integration point if and only is the integration point is positioned at the centre of ξ-axis.
- For a 2-nd order polynomial ($f = a\xi^2 + b\xi + c$), exact integration is possible with two integration points, located at $\xi = \pm 1/\sqrt{3}$ and weight 1. In fact, this integration scheme is also exact for 3-rd order polynomials.

The following rule applies, regarding the order of the polynomial $p$ that can be integrated exactly for a given number of integration points $n_{ip}$.

$$p = 2n_{ip} - 1$$

This information can be summarised in the following manner.

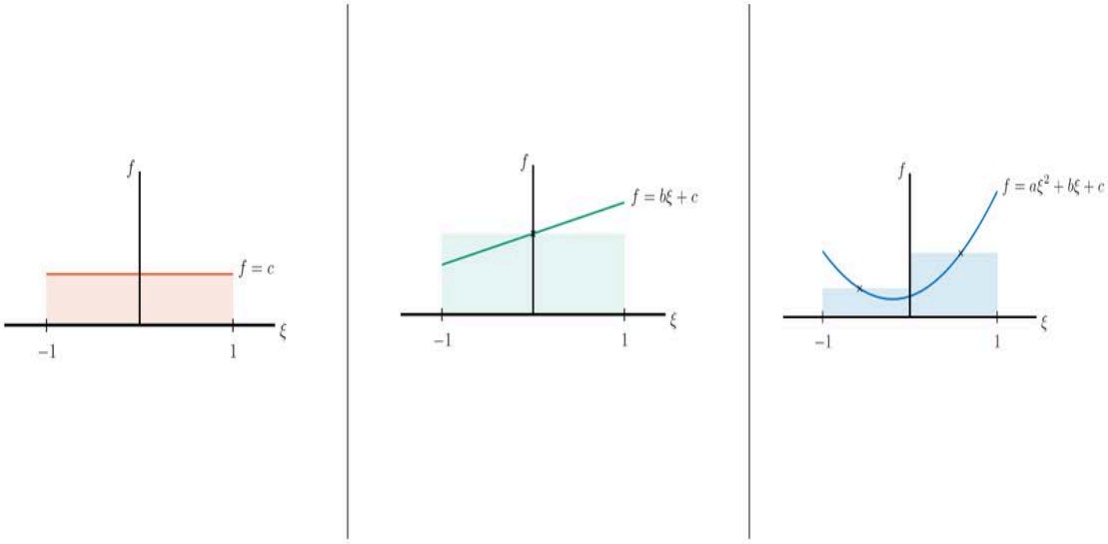



*Fig. 2.18* Gauss Integration points

| Number of points $n_{ip}$ | Position $\xi_i$ | weight | Polynomial order $p$ |
|---|---|---|---|
| $1$ | $0$ | $2$ | $0$ or $1$ |
| $2$ | $-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$ | $1, 1$ | $2$ or $3$ |
| $3$ | $-\frac{3}{\sqrt{5}}, 0, \frac{3}{\sqrt{5}}$ | $\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$ | $4$ or $5$ |

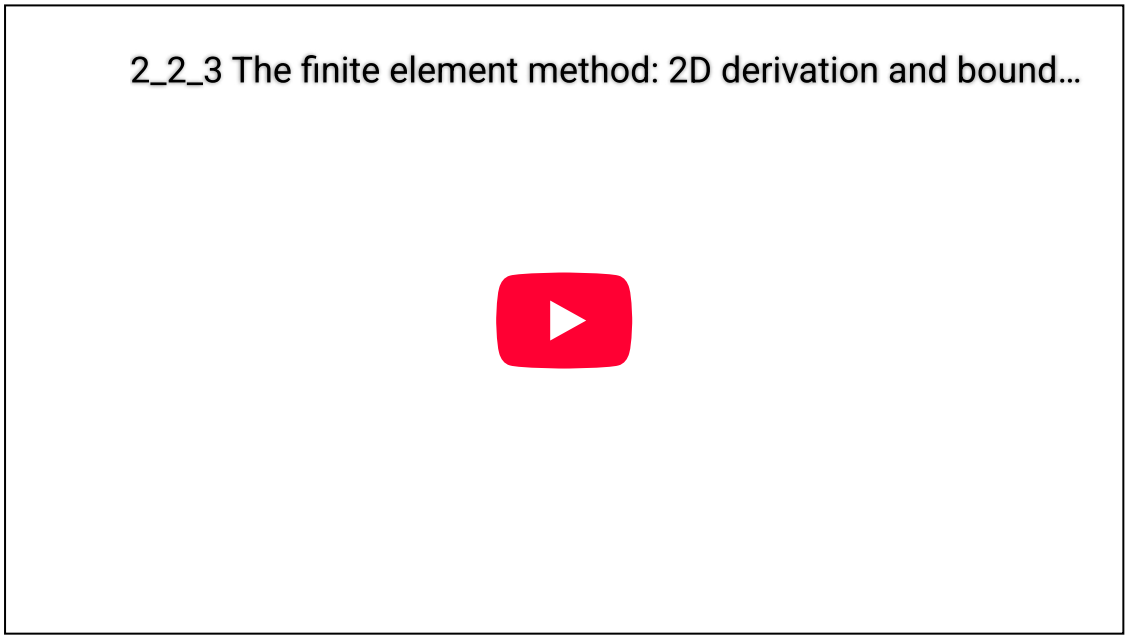
> **Attribution**
>
> This chapter was written by Frans van der Meer. Find out more here.

# Poisson equation in 2D

## Contents


- Strong form equation
- Weak form
- Discrete form


In this section, we will introduce finite element formulations for problems in higher dimensions. Again, we will focus on the Poisson equation, but not in 2D. In this problem, the solution is a field in 2D space. This implies the elements and shape functions need to be different (e.g. triangular, quadrilateral, tetrahedral).




## Strong form equation

The 2D Poisson equation describes the steady state (time-independent solution) of a diffusion problem.

$$-\nu\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)=q \tag{2.23}$$

Where $u$ is the unknown field, $\nu$ is the diffusivity parameter and $q$ is a source term. At least $u$ and possibly also $\nu$ and $q$ are a function of $x$ and $y$.

There are several applications where this PDE can be used to solve problems in sciences and engineering, such as:

- potential flow
- pressure in saturated soil
- temperature distribution in solids

Equation (2.23) can be written with the Laplacian operator $\Delta = \nabla \cdot \nabla$ as

$$-\nu\Delta u = q$$

To formulate a problem the strong form governing equation is combined with boundary conditions

$$\begin{aligned} u &= \bar{u}, \quad \text{on} \quad \Gamma_D \\ \nu\nabla u \cdot \mathbf{n} &= h \quad \text{on} \quad \Gamma_N \end{aligned}$$

where $\bar{u}$ stands for prescribed values for the unknown field on the part of the boundary where Dirichlet boundary conditions are applied ($\Gamma_D$) and $h$ for a flux on the part of the boundary where Neumann boundary conditions are applied ($\Gamma_N$) and $\mathbf{n}$ is the normal to that boundary.

# Weak form

As usual, we start with multiplying the strong form equation with a (for now arbirtary) test function and integrating over the domain

$$-\int_\Omega w\nu\Delta u \, d\Omega = \int_\Omega wq \, d\Omega \tag{2.24}$$

Integration by parts in combination with Divergence theorem gives:

$$\int_\Omega \nabla w \cdot (\nu \nabla u)\, d\Omega - \int_\Gamma w \nu \nabla u \cdot \mathbf{n}\, d\Gamma = \int_\Omega wq\, d\Omega$$

**Integration by parts / Divergence theorem**

For the step above, we make use of two fundamental relations. Firstly the product rule of differentation for divergence (with scalar $a$ and vector $\mathbf{b}$) which reads:

$$\nabla \cdot (a\mathbf{b}) = a\nabla \cdot \mathbf{b} + \nabla a \cdot \mathbf{b} \tag{2.25}$$

And secondly, divergence theorom, which reads (with vector $\mathbf{v}$)

$$\int_\Omega \nabla \cdot \mathbf{v}\, d\Omega = \int_\Gamma \mathbf{v} \cdot \mathbf{n}\, d\Gamma \tag{2.26}$$

With (2.25), we rewrite the left hand side of Equation (2.24) (with $w$ for $a$ and $\nabla u$ for $\mathbf{v}$). Then with (2.26) (with $w\nabla u$ for $\mathbf{v}$), we rewrite the second integral into a surface integral

Substitution of the Neumann boundary in combination with the condition that $w = 0$ on $\Gamma_D$ eliminates the unknown $u$ from the boundary term, after which it is moved to the right hand side to arrive at the weak form equation:

$$\int_\Omega \nabla w \cdot (\nu \nabla u)\, d\Omega = \int_\Omega wq\, d\Omega + \int_{\Gamma_N} wh\, d\Gamma \tag{2.27}$$

# Discrete form

Discretization of the solution is done with 2D shape functions

$$u \approx u^h = \sum_j N_j(x, y) u_j = \mathbf{Nu}$$

where $\mathbf{u}$ contains the degrees of freedom of the discretized field and $\mathbf{N}$ is a row vector with all shape functions:

$$\mathbf{N} = \begin{bmatrix} N_1 & N_2 & \dots & N_n \end{bmatrix}$$

in which $n$ is the number of degrees of freedom, which is for conventional shape functions equal to the number of nodes in the mesh.

Following the Bubnov-Galerkin method, the same discretization is introduced for $w$:

$$w \approx w^h = \mathbf{N}\mathbf{w}$$

The gradients of $u$ and $w$ are defined with the $\mathbf{B}$-matrix as:

$$\nabla u = \mathbf{B}\mathbf{u} \qquad \text{and} \qquad \nabla w = \mathbf{B}\mathbf{w}$$

with

$$\mathbf{B} = \begin{bmatrix} \frac{\partial N_1}{\partial x}, \frac{\partial N_2}{\partial x}, \ldots, \frac{\partial N_n}{\partial x} \\ \frac{\partial N_1}{\partial y}, \frac{\partial N_2}{\partial y}, \ldots, \frac{\partial N_n}{\partial y} \end{bmatrix}$$

Substitution into Equation (2.27) gives

$$\int_\Omega \mathbf{B}\mathbf{w}\nu\mathbf{B}\mathbf{u}\, d\Omega = \int_\Omega \mathbf{N}\mathbf{w}q\, d\Omega + \int_{\Gamma_N} \mathbf{N}\mathbf{w}h\, d\Gamma$$

As for the 1D problem, eliminating $\mathbf{w}$ involves transposing $\mathbf{N}$ and $\mathbf{B}$ matrices where they are multiplied with $\mathbf{w}$. Finally, we arrive at a linear system of equations written as:

$$\mathbf{K}\mathbf{u} = \mathbf{f}$$

with

$$\mathbf{K} = \int_\Omega \mathbf{B}^T\nu\mathbf{B}\, d\Omega \qquad \text{and} \qquad \mathbf{f} = \int_\Omega \mathbf{N}^T q\, d\Omega + \int_{\Gamma_N} \mathbf{N}^T h\, d\Gamma$$

### Poisson equation in 1D vs 2D vs 3D

The finite element derivation for the Poisson equation in 2D follows the same lines as the in 1D. We need Divergence theorem now to arrive at the term where Neumann boundary conditions are applied. This is now an integral over the boundary instead of a term evaluated at the ends of the rod.

The derivation above also holds for 3D. Expressions and steps are exactly the same, although the interpretation of some of the symbols changes: in 3D, $\Omega$ is a volume, $\Gamma$ is a surface and the $\mathbf{B}$ matrix has three rows.

### Attribution

This chapter was written by Frans van der Meer. Find out more here.

# Isoparametric mapping

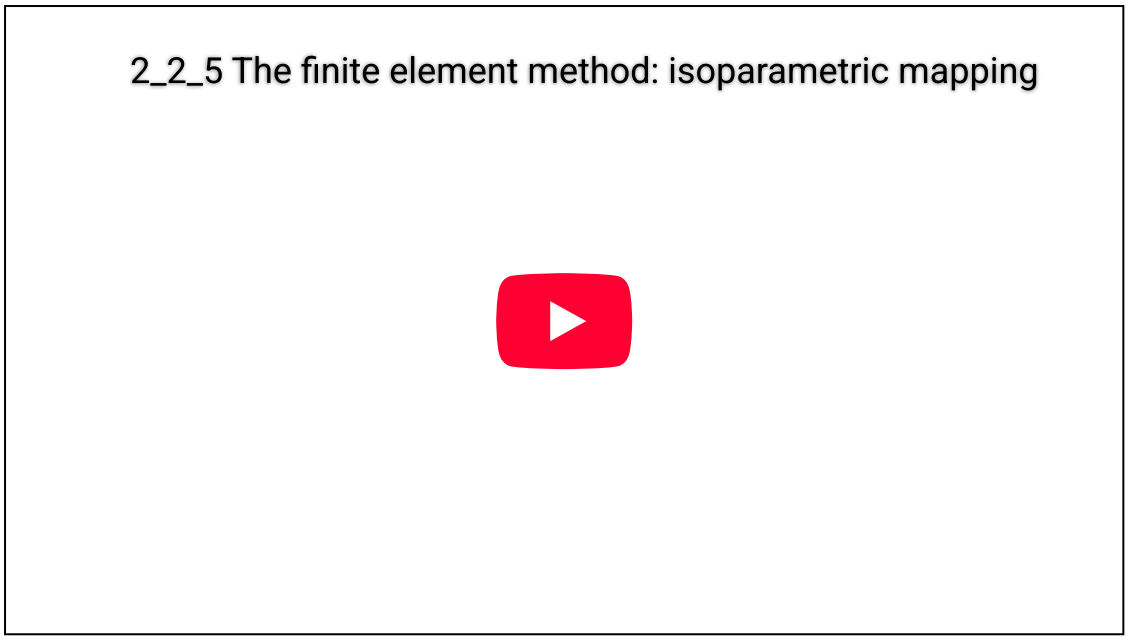
## Contents


- Isoparametric mapping on a 3-node line element
- Isoparametric mapping in higher dimensions


> **MUDE exam information**
>
> The contents of this page are *not* part of the MUDE exam. It is included in the textbook to provide additional information about the finite element method.

After introducing the first 2D finite element formulations, it is necessary to give some more attention to how finite element matrices are in practice evaluated. We have not given much attention to the formulation of shape functions and numerical integration. In Chapter 1 you have seen a simple 1D implementation with linear elements, where definition of shape functions and evaluation of element integrals is relatively straightforward. In general, for 2D or 3D elements, particularly when unstructured meshes need to be supported, this becomes more complicated. Most popular finite element codes make use of the same technique that relates to the definition of integration schemes and the definition of the shape functions, this technique is called *isoparametric mapping*.

The basic motivation is that we want to define shape functions and the integration scheme on a so-called *reference element* and that we want to use this same definition of shape functions and integration scheme for any element in a mesh, irrespective of its position and orientation.




# Isoparametric mapping on a 3-node line element

We will start illustrating the principle of isoparametric mapping with a 3-node line element. In principle, shape functions can be defined globally. For instance for the quadratic line element in 1D, we can state that all shape functions should have the form of a quadratic polynomial

$$N_i(x) = a_i x^2 + b_i x + c_i$$

with coefficients $a_i$, $b_i$ and $c_i$, where $i$ refers to the node number the shape function is associated with. We know $N_i$ should be equal to 1 at node $i$ and equal to 0 at the other two nodes. Then, if we have an element with three nodes with certain coordinates, that gives three equations for every shape function, which can be solved for the three coefficients. However, in practice $N_i(x)$ is not explicitly defined in implementations of the finite element method. The shape functions are defined instead on a reference element, which is the same for all elements in the mesh.

# The reference element

This reference element is defined in an auxiliary coordinate system, the *natural coordinates* of the element. In 1D, we use a $\xi$ as symbol for the natural coordinate. The reference element is defined on the domain $\xi \in [-1, 1]$. We can define the shape functions here, with

nodes at $\xi = -1$, $\xi = 0$ and $\xi = 1$, the quadratic shape functions that satisfy the Kronecker delta property are given as:

$$N_1(\xi) = \frac{1}{2}\xi^2 - \frac{1}{2}\xi$$
$$N_2(\xi) = 1 - \xi^2$$
$$N_3(\xi) = \frac{1}{2}\xi^2 + \frac{1}{2}\xi$$

**Exercise**

Show that the shape functions given above satisfy the partition of unity property and the Kronecker delta property (see chapter 1)

Note that the weights given for numerical integration in chapter 1, for instance with two point Gauss integration at $\xi = -1/\sqrt{3}$ and $\xi = 1/\sqrt{3}$ with $w_1 = w_2 = 1$ is directly suitable for this reference element. However, in a finite element model, where we have multiple elements, we need to perform integration over the actual domain of the elements. Moreover, in most models we will need the derivatives of the shape functions, and these are derivatives with respect to global coordinates.

# Requirements for the mapping

In order to be able to use the reference element for evaluation of these shape function derivatives and for performing numerical integration, we need a relationship between the actual coordinates $x$ of any physical element and the natural coordinates $\xi$ of the reference element.

In general, if we have an element defined over $x \in [a, b]$, we need to evaluate perform numerical integration from $x = a$ to $x = b$. It is possible to rewrite an integral of any function $f(x)$ over the element as an integral over the reference element as:

$$\int_a^b f(x)\, dx = \int_{-1}^{-1} f(x(\xi)) \left| \frac{\partial x}{\partial \xi} \right| d\xi$$

Addtionally, for evaluating the derivatives of shape functions, if the shape functions are defined in terms of $\xi$ but we need derivatives with respect to $x$, the operation we need is:

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial x}$$

In both of these operations, a relation between $x$ and $\xi$ is needed. A relationship between two coordinate systems is called a *mapping*.

# How the mapping is constructed

For the 1D element, it is quite simple to construct a linear mapping between $x$ and $\xi$, but with isoparametric mapping we will take an approach that can generalize well to higher dimensionality. In isoparametric mapping, we define $x(\xi)$ as the sum over all nodes of $N_i$ times the nodal coordinate $x_i$:

$$x(\xi) = \sum_i N_i(\xi) x_i \tag{2.28}$$

**Exercise**

What mapping do you obtain with the reference shape functions given above for an element with nodal coordinates $x_1 = 2$, $x_2 = 4$ and $x_3 = 6$?

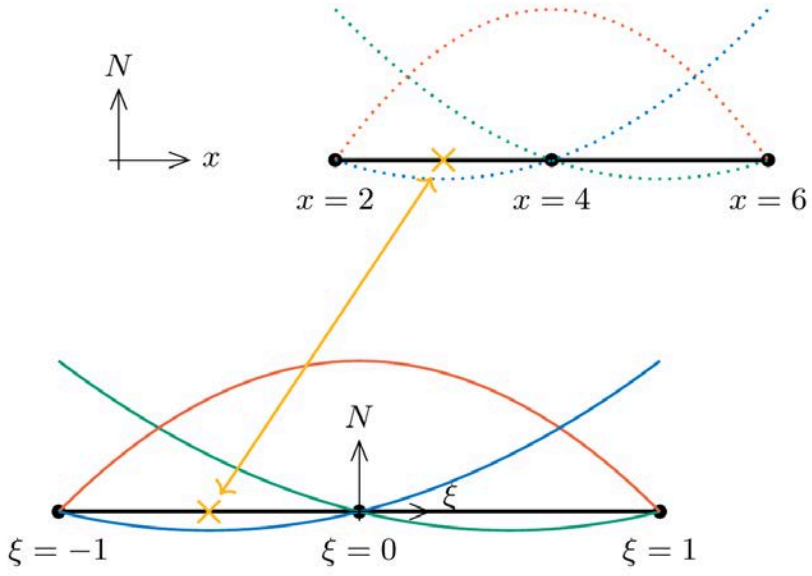



*Fig. 2.19* Example of isoparametric mapping for a 3-node line element

**Solution**

$x = 4 + 2\xi$

Can you think of any sanity checks whether this results makes sense?

**Solution**

Substitution of nodal coordinates in the reference element gives the physical coordinates of the nodes

$$\begin{aligned} \xi = -1 \quad &\rightarrow \quad x = 2 \\ \xi = 0 \quad &\rightarrow \quad x = 4 \\ \xi = 1 \quad &\rightarrow \quad x = 6 \end{aligned}$$

Also, the length of the element in physical domain is 4, while in the reference domain it has length 2. The difference is a factor 2. The scale factor of the length appears in the integral over the reference domain as the factor $\left|\frac{\partial x}{\partial \xi}\right|$, which indeed evaluates to 2.

The term *isoparametric* refers to the fact that the mapping of coordinates is **parametrized** in the **same** way as we have parametrized the unknown field, namely with shape functions and nodal values.

# The Jacobian

The term $\frac{\partial x}{\partial \xi}$ is referred to as the Jacobian of the mapping. It follows from the definition in Equation~(2.28) that this derivative can be evaluated for an element with given nodal coordinates $x_i$ as:

$$\frac{\partial x}{\partial \xi} = \sum_i \frac{\partial N_i}{\partial \xi} x_i$$

**Exercise**

For the example from the previous exercise, compute the Jacobian $\frac{\partial x}{\partial \xi}$ following the definition above.

**Solution**

Of course it is possible to take a direct derivative from the mapping found in the previous exercise, but here we use the formal approach

$$\frac{\partial x}{\partial \xi} = \frac{\partial N_1}{\partial \xi} x_1 + \frac{\partial N_2}{\partial \xi} x_2 + \frac{\partial N_3}{\partial \xi} x_3 = \left(\xi - \frac{1}{2}\right) \cdot 2 + (-2\xi) \cdot 4 + \left(\xi +$$

The result here is independent of $\xi$. Upon inspection of the performed operation it can be observed that this is the case if and only if $x_2 = \frac{1}{2}(x_1 + x_3)$, i.e. if the mid-node of the physical element is positioned exactly at the middle of the element domain.

The Jacobian itself is immediately what is needed to perform the numerical integration over the reference domain. For evaluation of shape function derivatives, we need $\frac{\partial \xi}{\partial x}$, which is the inverse of the Jacobian. Introducing the symbol $J$ for the Jacobian, we can write:

$$\int_a^b f(x)\, dx = \int_{-1}^{1} f(x(\xi))|J|\, d\xi$$

and

$$\frac{\partial N_i}{\partial x} = \frac{1}{J} \frac{\partial N_i}{\partial \xi}$$

# Isoparametric mapping in higher dimensions

Isoparametric has been illustrated above for the case of 1D. It deserves to be said, however, that if there were only finite elements in 1D, the concept of isoparametric mapping would likely not be widespread. The approach really shines in higher dimensions, particularly on unstructured meshes. As an example, take the arbitrarily oriented quadrilateral element shown below.

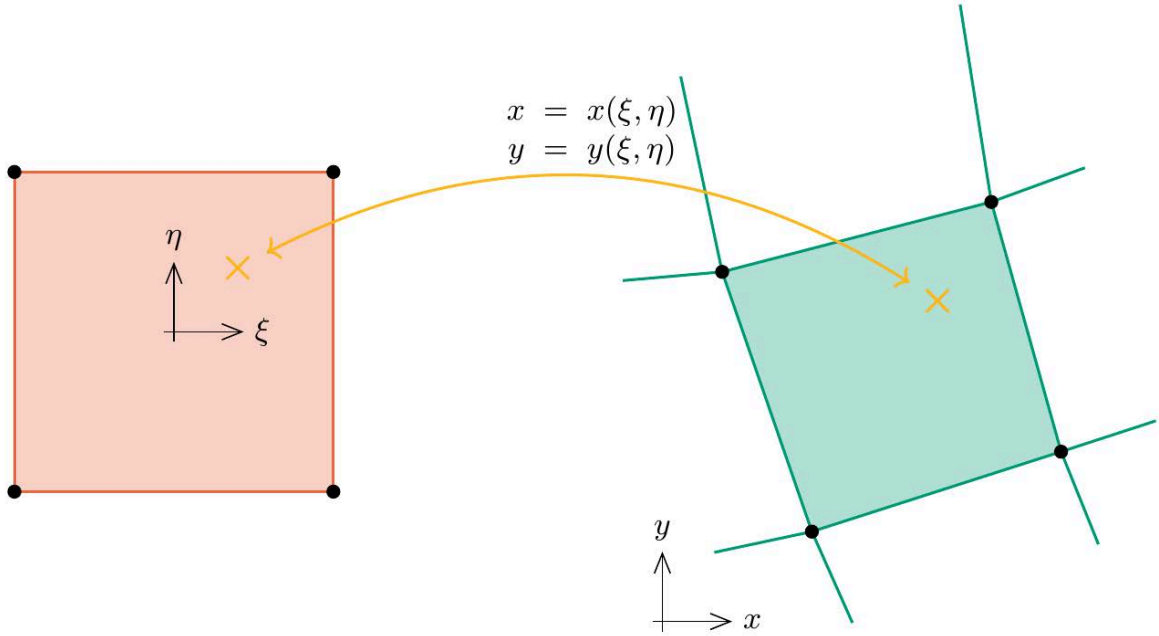



*Fig. 2.20* Example of isoparametric mapping for a 4-node quadrilateral element: reference quad (left) and physical counterpart (right)

There are now two natural coordinates $\xi$ and $\eta$. The reference quad is defined over the domain $(\xi \in [-1, 1], \eta \in [-1, 1])$ and its physical counterpart is arbitrarily positioned in 2D space with nodal coordinates $(x_i, y_i),\ i = 1 \dots 4$. Shape functions are defined once and for all for the reference element as:

$$
\begin{aligned}
N_1 &= \frac{1}{4}(1-\xi)(1-\eta) \\
N_2 &= \frac{1}{4}(1+\xi)(1-\eta) \\
N_3 &= \frac{1}{4}(1+\xi)(1+\eta) \\
N_4 &= \frac{1}{4}(1-\xi)(1+\eta)
\end{aligned}
$$

The $(\xi, \eta)$-coordinates of any point inside the reference element are mapped to $(x, y)$-coordinates in physical space through a mapping:

$$
\begin{aligned}
x(\xi, \eta) &= \sum_i N_i(\xi, \eta) x_i \\
y(\xi, \eta) &= \sum_i N_i(\xi, \eta) y_i
\end{aligned}
$$

The Jacobian is now a matrix, there are $2 \times 2$ derivatives of physical coordinates with respect to natural coordinates:

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \quad \text{with} \quad \frac{\partial x}{\partial \xi} = \sum_i \frac{\partial N_i}{\partial \xi} x_i \quad \text{etc.}$$

The inverse of the Jacobian matrix takes the place of $\frac{1}{J}$ in the calculation of shape function derivatives with respect to $x$ and $y$:

$$\begin{bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{bmatrix} = \mathbf{J}^{-1} \begin{bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \end{bmatrix}$$

Numerical integration of a quantity $f$ over the element domain $\Omega^e$ is performed over the reference element, now the correction involves the determinant of the Jacobian matrix

$$\iint_{\Omega^e} f \, dx \, dy = \int_{-1}^{1} \int_{-1}^{1} f \, |\mathbf{J}| \, d\xi \, d\eta$$

with $|\mathbf{J}| = \det \mathbf{J}$. Now the integral on the right can be evaluated with numerical integration. This means that position of the integration points for numerical integration can be defined once and for all in the reference element, irrespective of the size and orientation of the element in physical space. The factor $|\mathbf{J}|$, which does depend on the coordinates of the nodes corrects for the size of the actual element. Similarly, shape functions never need to be defined as function of $x$ and $y$. Only when we need their derivatives with respect to $x$ and $y$ is a transformation needed, which the $\mathbf{J}^{-1}$ takes care of.

**Exercise**

Consider a distorted quadrilateral element with nodal coordinates

$$\mathbf{x}_1 = (0,0), \quad \mathbf{x}_2 = (4,0), \quad \mathbf{x}_3 = (6,2), \quad \mathbf{x}_4 = (0,5)$$

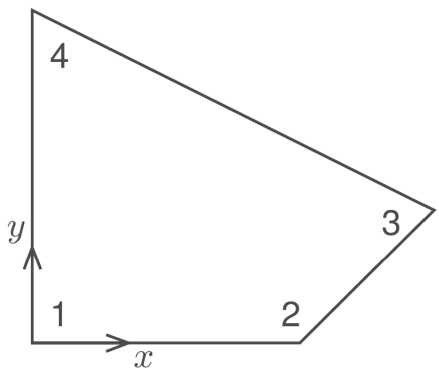


*Fig. 2.21* 4-node quadrilateral element in physical space

- Express $x$ and $y$ in terms of $\xi$ and $\eta$
- Evaluate the jacobian matrix $\mathbf{J}$

**Solution**

Coordinates are found with $x = \sum_i N_i x_i$ and $y = \sum_i N_i y_i$:

$$x = \frac{1}{2}(1+\xi)(5+\eta)$$
$$y = \frac{1}{4}(7-3\xi)(1+\eta)$$

The Jacobian matrix is found by taking derivatives of the expressions for $x$ and $y$ that were just obtained, or by multiplying the coordinates with the shape function derivatives. For this exercise, the first of these two approaches is most convenient, but the second one is used in finite element implementations as shape function derivatives can be pre-evaluated and hard-coded. Both result in the same Jacobian matrix:

$$\mathbf{J} = \begin{bmatrix} \frac{1}{2}(5+\eta) & -\frac{3}{4}(1+\eta) \\ \frac{1}{2}(1+\xi) & \frac{1}{4}(7-3\xi) \end{bmatrix}$$

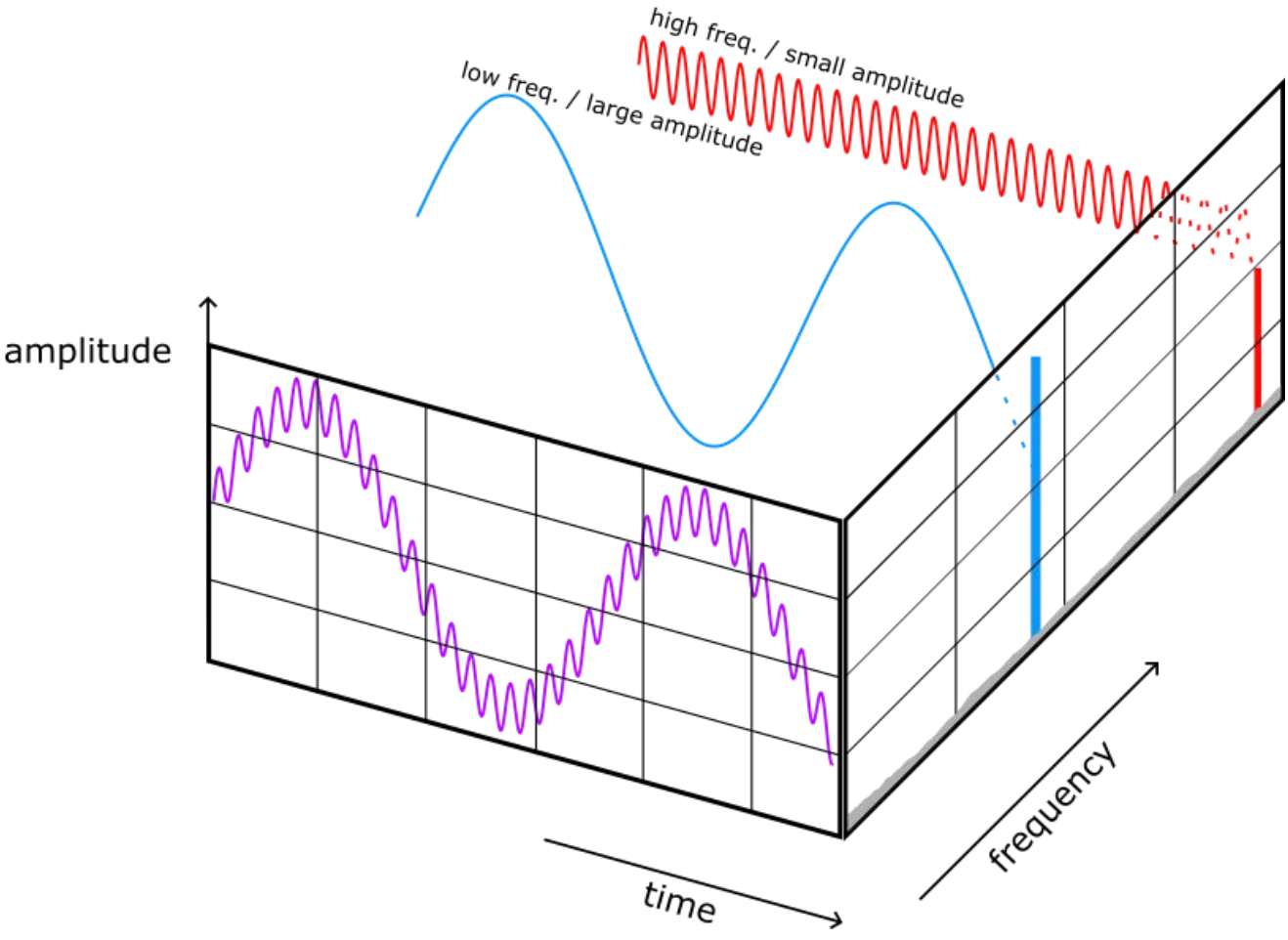
**Attribution**

This chapter was written by Frans van der Meer. Find out more here.

# Signal Processing

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

The goal of this week is to be able to identify and analyze frequency components in a signal; this is referred to as *spectral analysis.* The following chapters are told in a story-like sequence that will guide you through this process naturally.




We generally work with recordings of a signal in the *time domain.* Since it can be difficult to interpret such a signal, we will explore techniques for converting and evaluating this record in the *frequency domain.* The story first begins with a theoretical perspective (e.g., working with continuous-time functions), after which we consider the discrete-time case. The latter is what we use in practice. Our objective is to understand and work with signals that have been measured and recorded, i.e., understand and work with a discrete-time data set.

**MUDE Exam Information**

Proofs and derivations are provided in this textbook, but will not be asked in exam; there is no need to memorize equations, which will be provided at the exam, as needed. Instead, you are expected to apply the theory to actual problems (problem solving), and interpret the results (as obtained with a Python Notebook). Although all of the chapters are needed to understand the complete Signal Processing story, for the exam it is recommended to focus on two chapters in particular: Sampling and Discrete Fourier Transform. In-class activities (not in this book) are also relevant; for example, the Jupyter notebook assignments of the workshop and the project.

# Fourier Series

## Contents


- Periodic functions (a recap)
- Fourier Series
- Summary
- Exercise

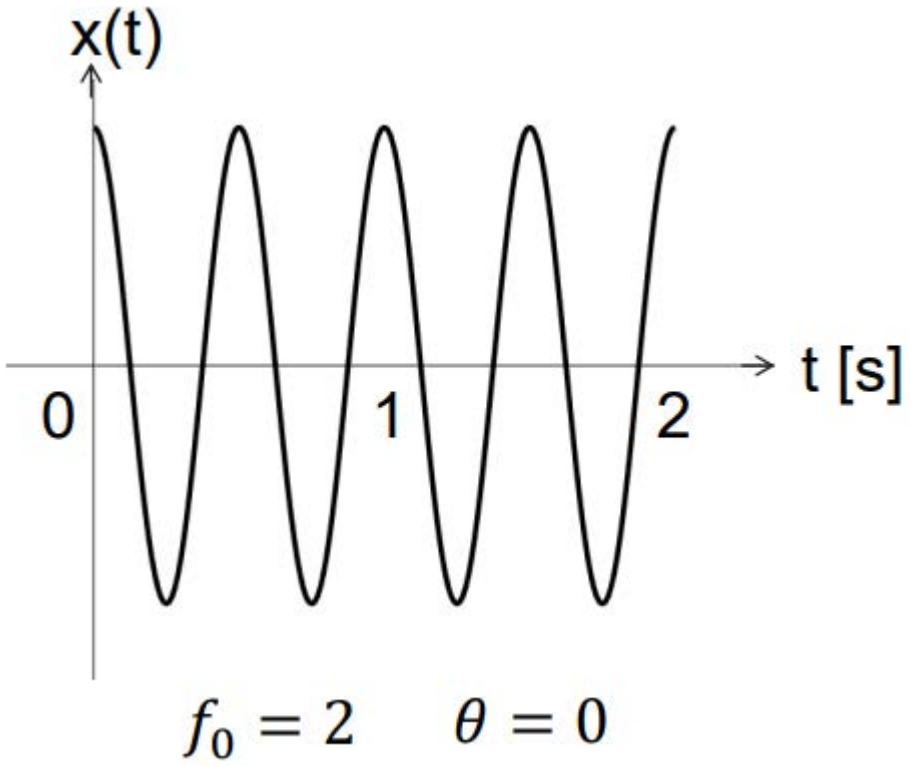

Our goal in this chapter is to be able to express a periodic signal $x(t)$ as a sum of harmonically related cosines and sines

$$x(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega_0 t) + \sum_{k=1}^{\infty} b_k \sin(k\omega_0 t)$$

Here we will formulate the series using real-valued functions (known as the *Real Fourier Series* or simply *Fourier Series*), and the next chapter will use complex numbers (the *Complex Fourier Series*).

## Periodic functions (a recap)

*Example:* $x(t) = A\cos(2\pi f_0 t + \theta)$



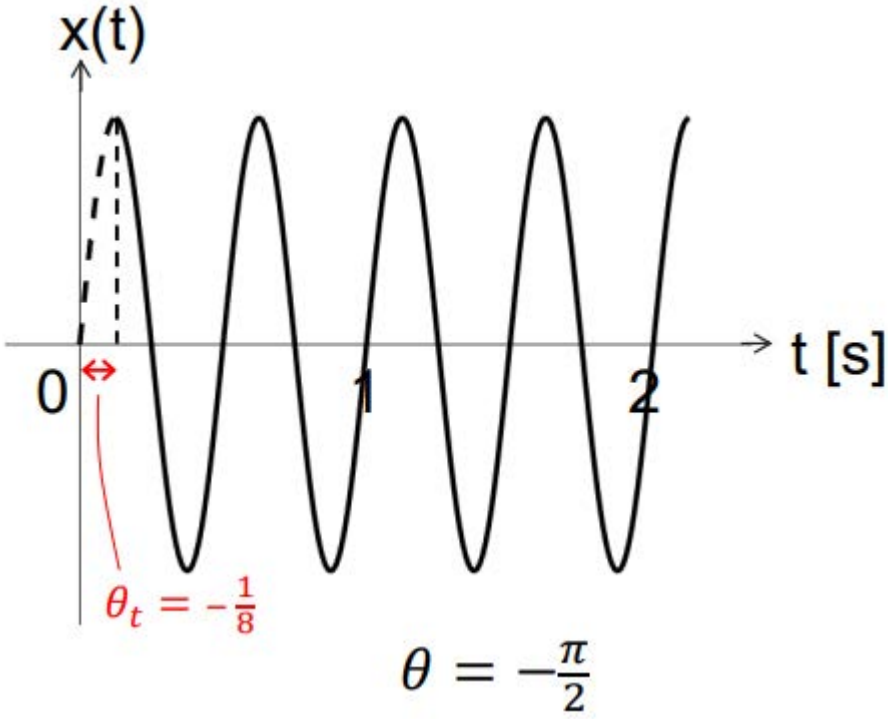

It is easy to see that, in this case we will have:

- Amplitude $A$ (e.g. V, m, m s$^{-2}$)
- Frequency $f_0$ [Hz]
  - Angular frequency $\omega_0 = 2\pi f_0$ [rad/s]
  - Period $T_0 = \frac{1}{f_0} = \frac{2\pi}{\omega_0}$ [s]
- Phase $\theta$ [rad]

With this, the **time delay**, $\theta_t$ [s], caused by the phase can be computed as

$$\theta_t = \frac{\theta}{2\pi f_0}$$

as $x(t) = A\cos(2\pi f_0(t + \theta_t))$




Cosine and sine can be thought of as originating from an object travelling over a circle; the object is *rotating* at angular frequency $\omega_0$. The unit circle is interpreted as being in the complex plane, with Euler's formula:

$$\tilde{x}(t) = e^{j\omega_0 t} = \cos(\omega_0 t) + j\sin(\omega_0 t)$$

The position of the object is described by the complex number $\tilde{x}(t)$, which is illustrated by the h vector in the figure below. We can also easily state that:

- $\text{Re}(\tilde{x}(t)) = \cos(\omega_0 t)$
- $\text{Im}(\tilde{x}(t)) = \sin(\omega_0 t)$

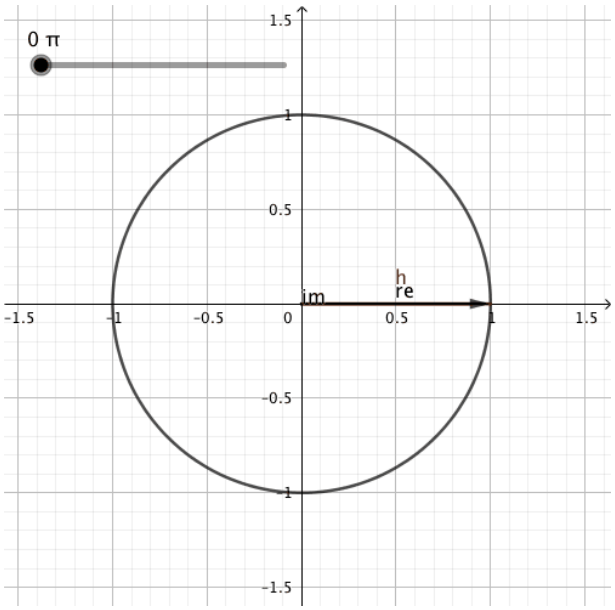


*Fig. 3.18* CC-BY-SA-4.0 licensed by BFG (2020)

A deterministic function/signal is **periodic** if

$$x(t+T_0) = x(t), \quad \text{for } -\infty < t < \infty$$

where $T_0$ is the **period** of the signal. The smallest value of $T_0$ that satisfies the above equation is the so-called **fundamental period**. And, finally, if the equation is not satisfied for any value of $T_0$ then the signal is **aperiodic**.

We consider signals in single dimension, typically time $t$ (independent variable), hence $x(t)$. Instead we can have $x(r)$, with $r$ (1D) position coordinate, or even multi-variate position vector **r** (the Earth's surface or the sea surface with waves can be considered as 2D examples).

# Fourier Series

**Motivation:** Can we express any periodic function/signal, e.g. square wave, as a sum of just cosines and sines with different frequencies?

Apparently, there exists the possibility of building up arbitrary periodic signal from **sums of harmonically related sinusoidal terms**. Therefore:

> Given a specific periodic signal, how do we find its trigonometric series representation?

It will be shown that the resulting series, **unique** for each periodic signal, is called the **trigonometric Fourier series** of that signal, sometimes also real Fourier series, or trigonometric polynomial, i.e. sum of cosines and sines, all with zero phase.

**A bit of history - Fourier biography**

Jean Baptiste Joseph Fourier (1768-1830) was a French mathematician and physicist.

He went with Napoleon to Egypt, then returned to France to become préfect of Isère, and there, in his spare time, he worked on the mathematical descriptions of heat transfer and vibrations.

In 1822 he presented his main work on heat flow, in which he claims that any function of a variable, whether continuous or discontinuous, can be expanded in a series of sines of multiples of the variable.

Though this result is not fully correct, Fourier's observation that some discontinuous functions are the sum of infinite series was a breakthrough. The Fourier series and Fourier transform have been named in his honour.

The general form of the real trigonometric Fourier Series is

$$x(t) = a_0 + a_1\cos(\omega_0 t) + a_2\cos(2\omega_0 t) + \cdots + b_1\sin(\omega_0 t) + b_2\sin(2\omega_0 t)\ldots$$

or, in a more compact form:

$$x(t) = a_0 + \sum_{k=1}^{\infty} a_k\cos(k\omega_0 t) + \sum_{k=1}^{\infty} b_k\sin(k\omega_0 t), \quad \omega_0 = \frac{2\pi}{T_0}$$

with **integer** values for $k$. Note that as the right-hand side is a sum of *harmonically-related* sinusoids, so must be the left-hand side (i.e. it must be periodic). Now our goal is to find the coefficients $a_0$, $a_k$ and $b_k$ such that $x(t)$ is represented (or approximated) best.

**Derivation**

**MUDE Exam Information**

This derivation is provided for additional insight and will not be part of the exam.

We start with $a_0$. Integrate all terms over one period:

$$\int_{T_0} x(t)dt = \int_{T_0} a_0 dt + \int_{T_0} a_1 \cos(\omega_0 t)dt + \ldots + \int_{T_0} b_1 \sin(\omega_0 t)dt + \ldots$$

When integrating sine or cosine over one period, the same area appears above and below the $t$ axis. The same holds for any $\cos(k\omega_0 t)$ or $\sin(k\omega_0 t)$. So, all terms on right-hand side, except for the first one, are zero. This yields:

$$\int_{T_0} x(t)dt = a_0 T_0$$

So:

$$a_0 = \frac{1}{T_0} \int_{T_0} x(t)dt$$

Hence, $a_0$ is the **average value of the signal**. It can also be interpreted as cosine with zero frequency.

Next, for the $a_k$ coefficients, these can be found by multiplying both sides of the equation with $\cos(m\omega_0 t)$ and integrating over $T_0$:

$$\int_{T_0} x(t)\cos(m\omega_0 t)dt = a_0 \int_{T_0} \cos(m\omega_0 t)dt + \int_{T_0} \left(\sum_{k=1}^{\infty} a_k \cos(k\omega_0 t)\right) \cos(m\omega_0$$

First term is zero, as we are again integrating cosine over multiple of periods. Next, we move $\cos(m\omega_0 t)$ inside summation and integrate term by term.

$$\int_{T_0} x(t)\cos(m\omega_0 t)dt = \sum_{k=1}^{\infty} a_k \int_{T_0} \cos(k\omega_0 t)\cos(m\omega_0 t)dt + \sum_{k=1}^{\infty} b_k \int_{T_0} \sin(k\omega_0 t)$$

Essentially, we switch order of summation and integration. Now, we use **orthogonality** of integrals involving products of sines and cosines:

$$\int_{T_0} \cos(k\omega_0 t)\cos(m\omega_0 t)dt = \begin{cases} 0 : k \neq m \\ T_0/2 : k = m \neq 0 \end{cases}$$
$$\int_{T_0} \sin(k\omega_0 t)\sin(m\omega_0 t)dt = \begin{cases} 0 : k \neq m \\ T_0/2 : k = m \neq 0 \end{cases}$$
$$\int_{T_0} \sin(k\omega_0 t)\cos(m\omega_0 t)dt = 0, \quad \text{for all } k, m$$

These hold for integer values of $k$ and $m$, and $T_0$ corresponding to $\omega_0$

**Note**

The above integrals are quite easily proven by using trigonometric identities, such as

$$\sin u \sin v = \frac{1}{2}\cos(u - v) - \frac{1}{2}\cos(u + v)$$

Applying these properties, we see that all terms in second series are equal to 0 for all $k$. In the first series, only the term for which $k = m$ is non-zero. So:

$$\int_{T_0} x(t)\cos(m\omega_0 t)dt = \int_{T_0} x(t)\cos(k\omega_0 t)dt =$$
$$\sum_{k=1}^{\infty} a_k \int_{T_0} \cos(k\omega_0 t)\cos(m\omega_0 t)dt + \sum_{k=1}^{\infty} b_k \int_{T_0} \sin(k\omega_0 t)\cos(m\omega_0 t)dt = \frac{a_k T_0}{2}$$

So, finally:

$$a_k = \frac{2}{T_0}\int_{T_0} x(t)\cos(k\omega_0 t)dt, \quad \text{with } k \in \mathbb{N}^+$$

In a similar manner, we can find the coefficients $b_k$ by multiplying series with $\sin(m\omega_0 t)$, integrating over $T_0$ and using orthogonality properties. We therefore obtain:

$$b_k = \frac{2}{T_0}\int_{T_0} x(t)\sin(k\omega_0 t)dt, \quad \text{with } k \in \mathbb{N}^+$$

# Summary

Any periodic signal can be written into series of harmonically related sinusoids:

$$x(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega_0 t) + \sum_{k=1}^{\infty} b_k \sin(k\omega_0 t), \quad \text{with } \omega_0 = \frac{2\pi}{T_0}$$

where the coefficients can be found as:

$$a_0 = \frac{1}{T_0} \int_{T_0} x(t) dt$$
$$a_k = \frac{2}{T_0} \int_{T_0} x(t) \cos(k\omega_0 t) dt, \quad \text{with } k \in \mathbb{N}^+$$
$$b_k = \frac{2}{T_0} \int_{T_0} x(t) \sin(k\omega_0 t) dt, \quad \text{with } k \in \mathbb{N}^+$$

# Exercise

**Fourier Series Exercise**

The following periodic signal with period $T_0$ signal is a so-called “square wave:”

$$x(t) = \begin{cases} +1, & \text{for } 0 \leq t < \frac{T_0}{2}. \\ -1, & \text{for } \frac{T_0}{2} \leq t < T_0. \end{cases}$$

Derive expressions for the coefficients $a_0$, $a_k$ and $b_k$, with $k \in \mathbb{N}^+$, of the real, trigonometric Fourier series of $x(t)$. Include also the full expression of the Fourier series of signal $x(t)$.

*Hint: it is always a good idea to first make a sketch of signal $x(t)$. In addition, at some point you may want to use the relation $\omega_0 = 2\pi/T_0$.*

**Solution**

The video below illustrates how to arrive at the solutions, which are presented here:

$$a_0 = 0 \quad \text{and} \quad a_k = 0$$

$$b_k = \begin{cases} \frac{4}{k\pi}, & \text{for } k.\text{odd} \\ 0, & \text{for } k.\text{even} \end{cases}$$

**Attribution**

This chapter is written by Christiaan Tiberius. Find out more here.

# Square wave example

## Contents


- Using SymPy!
- Conclusion


Can we express basically any periodic function/signal as a sum of just cosines and sines with different frequencies?
Yes! This notebook will demonstrate that a square wave function of 1 Hz can be constructed as a sum of sine functions in the form below:

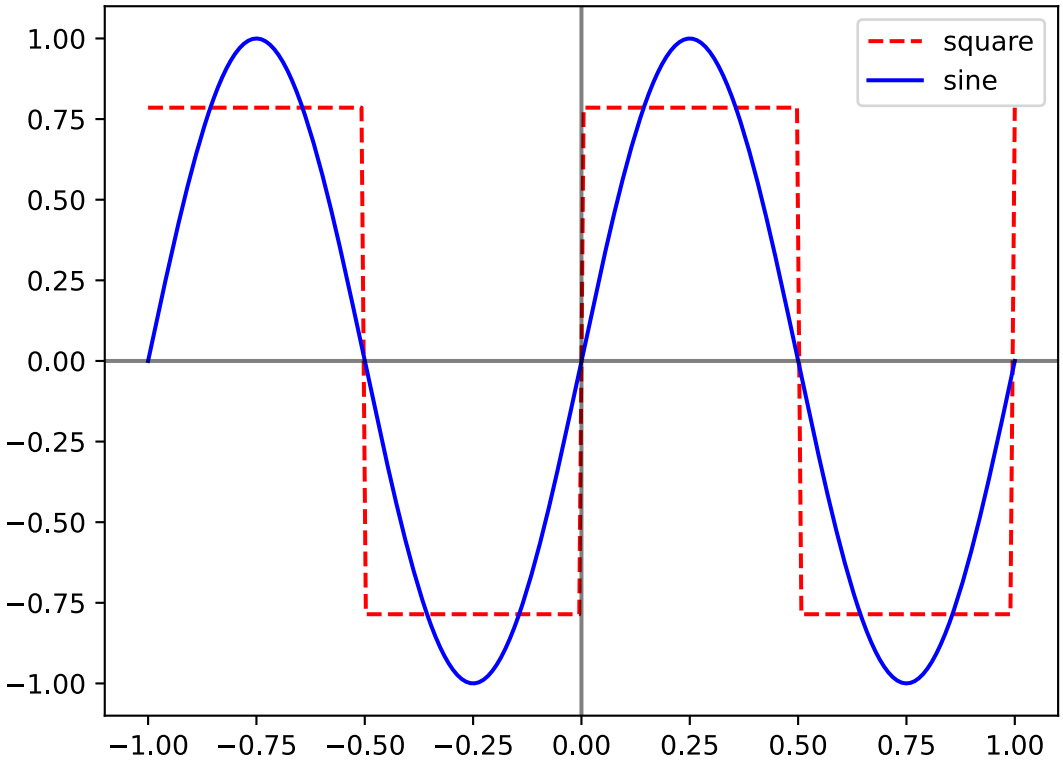
$$x(t) = \sin(2\pi t) + \frac{\sin(3 \cdot 2\pi t)}{3} + \frac{\sin(5 \cdot 2\pi t)}{5} + \ldots \frac{\sin(N \cdot 2\pi t)}{N}$$

$x(t)$ as a function of time t , and with odd number N.

```
import numpy as np
from matplotlib import pyplot as plt
```

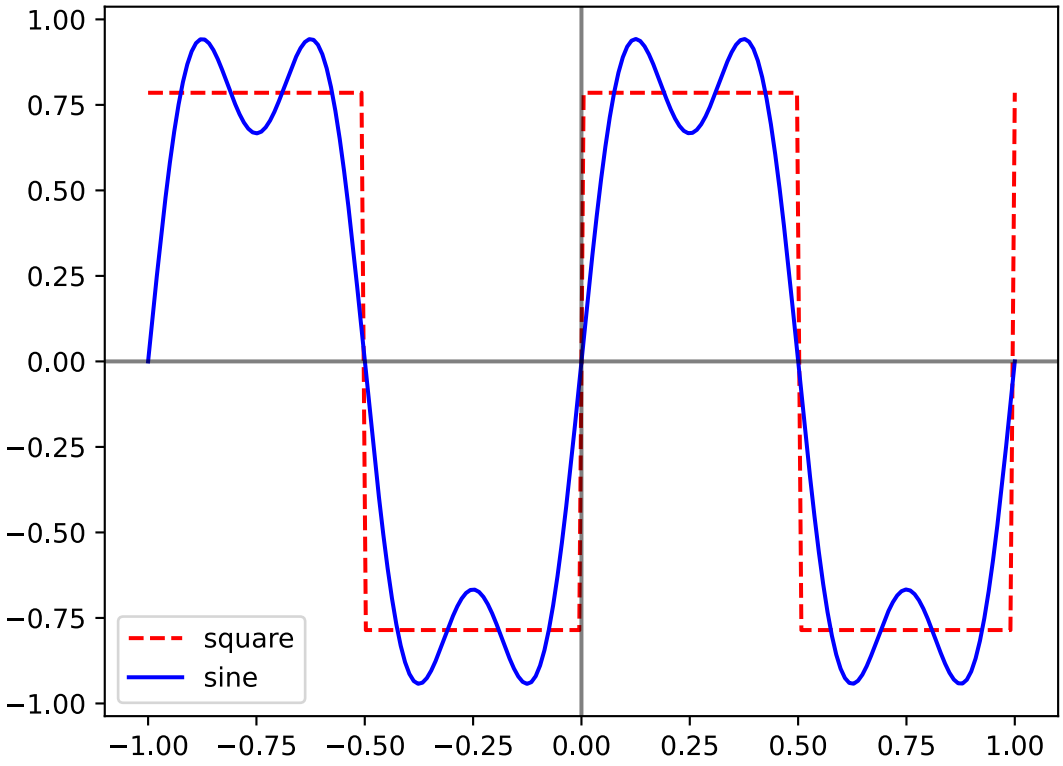
First, create the square wave function and the first sine term in the above equation

```
from scipy.signal import square
L = 1
t = np.linspace(- L, L, 200) # 200 time steps from -1 to 1 seconds
ysquare = square(2 * np.pi * t) * np.pi / 4
x1 = np.sin(2 * np.pi* t)
plt.plot(t, ysquare, 'r--', label='square')
plt.plot(t, x1, 'b-', label='sine')
plt.axhline(0, color='k', alpha=0.5)
plt.axvline(0, color='k', alpha=0.5)
plt.legend();
```



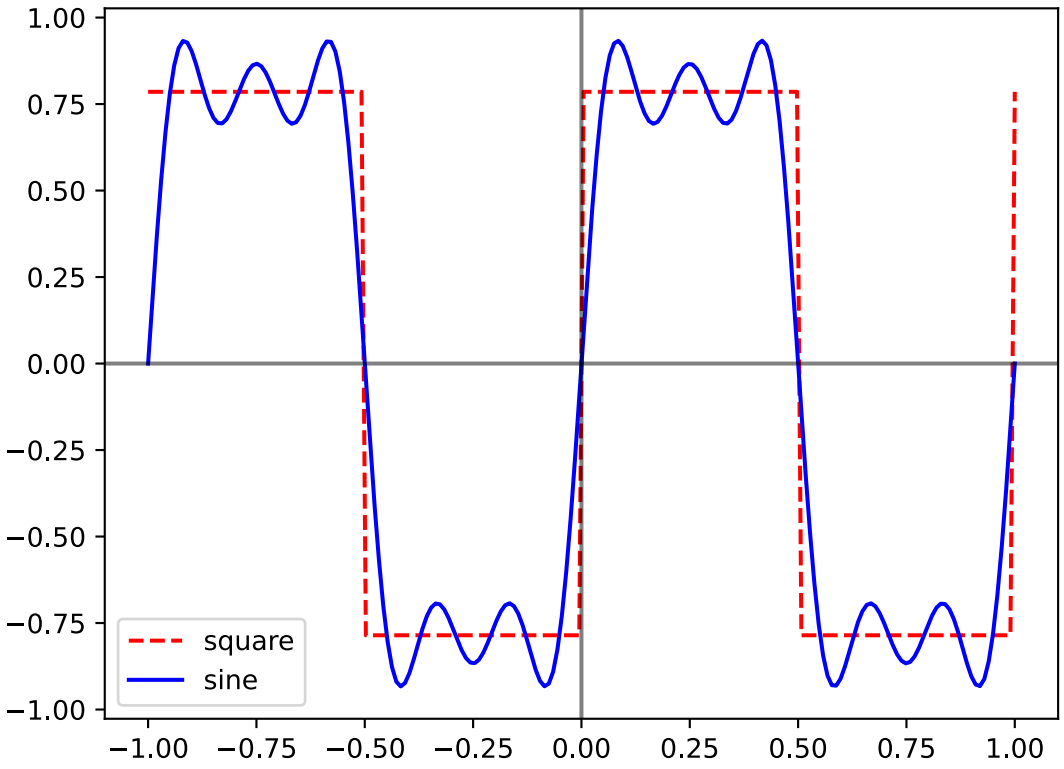

Now, we add the second sine term x2

```
x2 = 1 / 3 * np.sin(2 * np.pi * 3 * t)
plt.plot(t, ysquare, 'r--', label='square')
plt.plot(t, x1 + x2, 'b-', label='sine')
plt.axhline(0, color='k', alpha=0.5)
plt.axvline(0, color='k', alpha=0.5)
plt.legend();
```



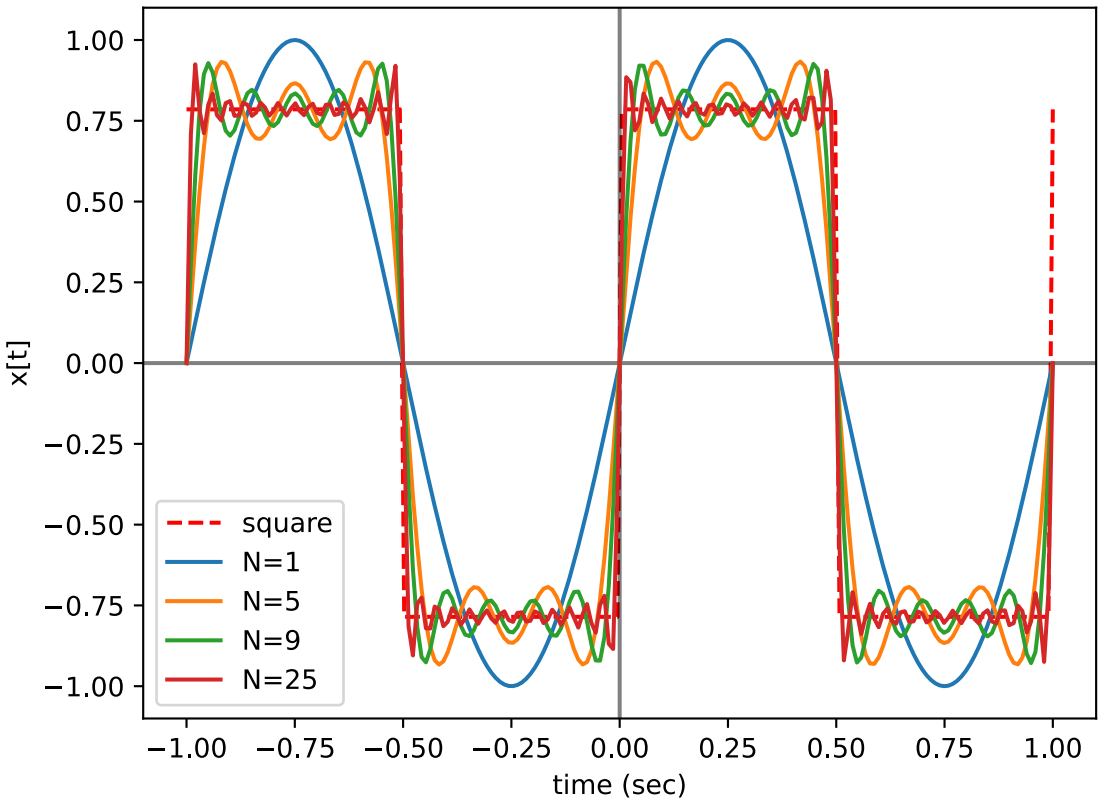
Then, the third sine term x3. We see that a square wave starts to take shape as more terms are included.

```
x3 = 1 / 5 * np.sin(2 * np.pi * 5 * t)
plt.plot(t, ysquare, 'r--', label='square')
plt.plot(t, x1 + x2 + x3, 'b-', label='sine')
plt.axhline(0, color='k', alpha=0.5)
plt.axvline(0, color='k', alpha=0.5)
plt.legend();
```




To observe the effects of adding more sine terms, we define a function to do so.

```
# Define variables
x = np.zeros(len(t)) # create x array that is the same length as t
odds = np.arange(1,1000,2) # array consists of odd numbers from 1,3,5, to 999

# Define a function that takes certain length of odds array as inputs to construct the x
def transform(number, plot=False, print_value=False):
    """
    number: the length of the odds array that is used for constructing signal x(t) in arr
    """
    for i in range(len(t)):
        term = []
        for j in range(len(odds[0:number])):
            term.append(1 / odds[j] * np.sin(2 * np.pi * odds[j] *t[i]))
        x[i] = np.sum(term)
    if plot==True:
        plt.plot(t,x, label=f'N={odds[number-1]}')
        plt.xlabel('time (sec)')
        plt.ylabel('x[t]')
        plt.legend()
    if print_value==True:
        return x

# Take a look at the x series given by different number of sine terms
# Test number of terms N = 1,9,19,49

plt.figure()
plt.plot(t, ysquare,'r--', label='square')
plt.axhline(0, color='k', alpha=0.5)
plt.axvline(0, color='k', alpha=0.5)
for number in (np.array([1, 5, 9, 25]) + 1) / 2:
    transform(int(number), True, False)
# You can replace any odd number with np.array([1, 5, 9, 25]) to see the effect of using
```



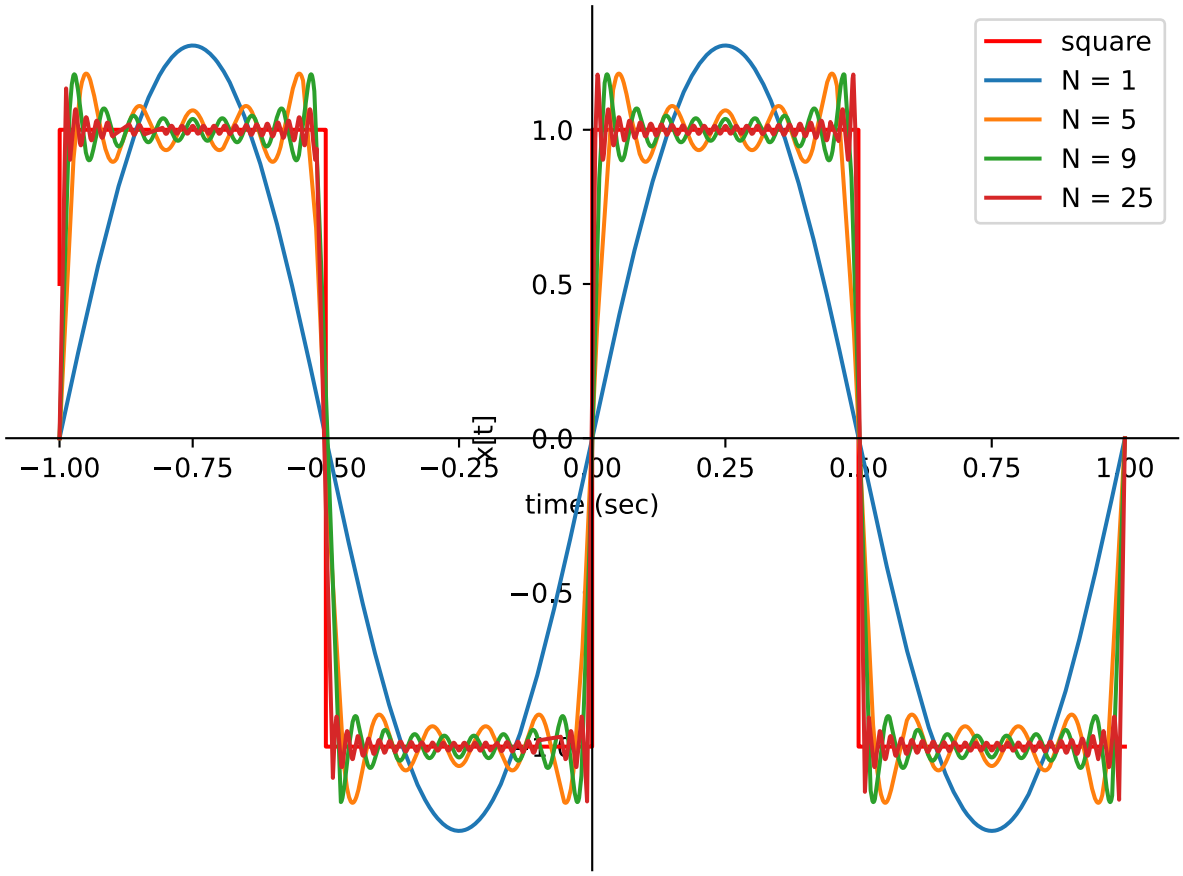
# Using SymPy!

The calculation of Fourier series is (among others) implemented in SymPy, see documentation. The calculations above are repeated using sympy:

```
import sympy as sym

x = sym.symbols('x')
N = sym.symbols('N')
f = sym.Heaviside(x+1) - sym.Heaviside(x+1/2)*2+sym.Heaviside(x)*2 - sym.Heaviside(x-1/2)
s = sym.fourier_series(f, (x, 0, 1))
s1  = s.truncate(1)
s5  = s.truncate(5)
s9  = s.truncate(9)
s25 = s.truncate(25)
display(s5)
```

$$\frac{4\sin(2\pi x)}{\pi} + \frac{4\sin(6\pi x)}{3\pi} + \frac{4\sin(10\pi x)}{5\pi} + \frac{4\sin(14\pi x)}{7\pi} + \frac{4\sin(18\pi x)}{9\pi}$$

```
p = sym.plot(f, s1, s5, s9, s25, (x, -1, 1), show=False, legend=True,ylabel='x[t]',xlabel
p[0].line_color = 'red'
p[0].label = 'square'
p[1].label = 'N = 1'
p[2].label = 'N = 5'
p[3].label = 'N = 9'
p[4].label = 'N = 25'
p.show()
```




# Conclusion

We can observe that the created function becomes more like a square wave function as the number of terms (N) increases. So, a square wave function can be simulated by a sum of harmonically related sine functions.

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

# Complex Fourier Series

## Contents


- A quick recap
- Complex Fourier Series
- Line Spectra
- Summary


## A quick recap

From the previous chapter, we have found that any periodic signal can be written into series of harmonically related sinusoids:

$$x(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(k\omega_0 t) + \sum_{k=1}^{\infty} b_k \sin(k\omega_0 t), \quad \text{with } \omega_0 = \frac{2\pi}{T_0}$$

where the coefficients can be found as:

$$a_0 = \frac{1}{T_0} \int_{T_0} x(t) dt$$

$$a_k = \frac{2}{T_0} \int_{T_0} x(t) \cos(k\omega_0 t) dt, \quad \text{with } k \in \mathbb{N}^+$$

$$b_k = \frac{2}{T_0} \int_{T_0} x(t) \sin(k\omega_0 t) dt, \quad \text{with } k \in \mathbb{N}^+$$

Now the *same* signal decomposition, but *in a different form* using complex algebra as commonly done in spectral analysis of signals, will be used.

## Complex Algebra

$j$: imaginary unit (or number) such that $j^2 = -1$

$e$: Euler's number (base of the natural logarithm), $e \approx 2.71$

**Euler's formula:**

$$e^{j\theta} = \cos\theta + j\sin\theta$$
$$e^{-j\theta} = \cos\theta - j\sin\theta$$

and, therewith:

$$e^{j\theta} + e^{-j\theta} = 2\cos\theta \implies \cos\theta = \frac{1}{2}(e^{j\theta} + e^{-j\theta})$$
$$e^{j\theta} - e^{-j\theta} = 2j\sin\theta \implies \sin\theta = \frac{1}{2j}(e^{j\theta} - e^{-j\theta})$$

# Complex Fourier Series

$$x(t) = \sum_{k=-\infty}^{\infty} X_k e^{jk\omega_0 t}$$

First equation is known as **synthesis** equation of Fourier Series, as in contructs the signal using complex exponential basis functions

$$X_k = \frac{1}{T_0}\int_{T_0} x(t)e^{-jk\omega_0 t}dt, \quad k \in \mathbb{Z}$$

Second equation is known as **analysis** equation of Fourier Series, as it allows us to analyze how signal can be represented by complex exponential basis functions (where index $k$ refers to frequency $k\omega_0 = k2\pi f_0$).

**Derivation**

**MUDE Exam Information**

This derivation is provided for additional insight and will not be part of the exam.

Start by substituting the complex exponential forms of both $\cos(k\omega_0 t)$ and $\sin(k\omega_0 t)$ into trigonometric series:

$$\begin{aligned} x(t) &= a_0 + \sum_{k=1}^{\infty} a_k \frac{1}{2}(e^{jk\omega_0 t} + e^{-jk\omega_0 t}) + \sum_{k=1}^{\infty} b_k \frac{1}{2j}(e^{jk\omega_0 t} - e^{-jk\omega_0 t}) = \\ &= a_0 + \sum_{k=1}^{\infty} \underbrace{\frac{1}{2}(a_k - jb_k)}_{X_k} e^{jk\omega_0 t} + \sum_{k=1}^{\infty} \underbrace{\frac{1}{2}(a_k + jb_k)}_{X_{-k}} e^{-jk\omega_0 t} = \\ &= a_0 + \sum_{k=1}^{\infty} X_k e^{jk\omega_0 t} + \sum_{k=1}^{\infty} X_{-k} e^{-jk\omega_0 t} \end{aligned}$$

If we replace the last term index $-k$ for $k$ then we get:

$$x(t) = a_0 + \sum_{k=1}^{\infty} X_k e^{jk\omega_0 t} + \sum_{k=-1}^{-\infty} X_k e^{jk\omega_0 t}$$

As when $k = 0$ we have $X_k = X_0 = a_0$ and $e^{jk\omega_0 t} = e^{j0\omega_0 t} = 1$, then we can compact everything and write

$$x(t) = \sum_{k=-\infty}^{\infty} X_k e^{jk\omega_0 t}$$

where

$$X_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-jk\omega_0 t} dt, \quad k \in \mathbb{Z}$$

# Coefficients

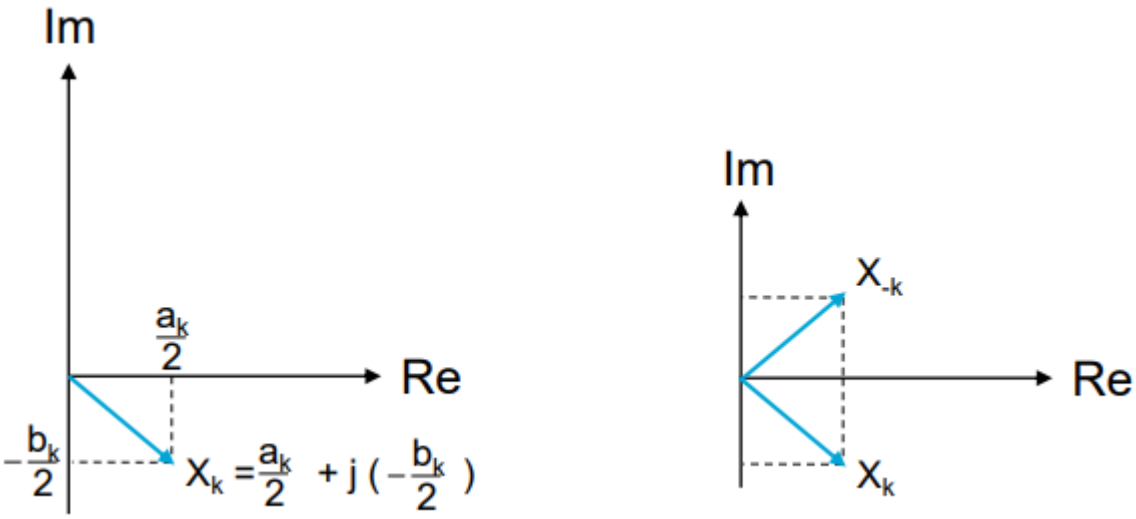
Real and complex Fourier series coefficients are related by:

$$X_k = \begin{cases} \frac{1}{2}(a_k - jb_k), & k > 0 \\ \frac{1}{2}(a_{-k} + jb_{-k}), & k < 0 \end{cases}$$

so

$$X_1 = \frac{1}{2}(a_1 - jb_1), X_2 = \frac{1}{2}(a_2 - jb_2)$$
$$X_{-1} = \frac{1}{2}(a_1 + jb_1), X_{-2} = \frac{1}{2}(a_2 + jb_2)$$

And we clearly see that $X_k = X^*_{-k}$, with $*$ denoting the complex conjugate. The visual representation of $X_k$ and $X_{-k}$ will be, therefore:




We have $a_k = 2\text{Re}(X_k)$ and $b_k = -2\text{Im}(X_k)$ for $k > 0$

Visualising complex coefficients $X_k$ as vectors in **complex plane**: *magnitude*, or modulus (amplitude), and *argument*, or phase. We can write the coefficients in **polar** (or phasor) form:

$$X_k = |X_k|e^{j\theta_k}, \quad \text{with } \theta_k = \angle X_k = \arg(X_k) \quad \text{for } -\infty < k < \infty (k \in \mathbb{Z})$$

and we find:

- **Magnitude** $|X_k| = \frac{1}{2}\sqrt{a_k^2 + b_k^2}$, k>0
- **Argument** $\theta_k = \arctan\left(-\frac{b_k}{a_k}\right)$, k>0

For real signals $x(t)$, as we use in this practical course, we have $|X_k| = |X_{-k}|$ and $\theta_k = -\theta_{-k}$

# Line Spectra

The complex exponential Fourier series coefficients

$$X_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-jk\omega_0 t} dt, \quad k \in \mathbb{Z}$$

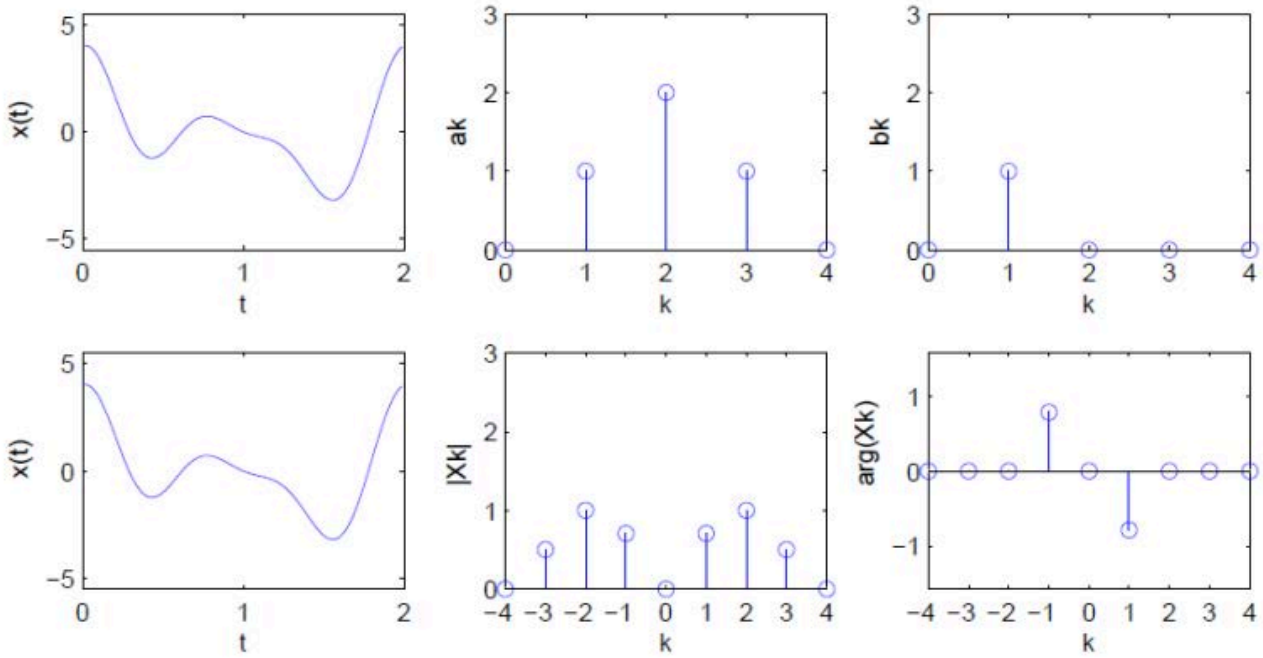
- shows **amplitudes** of all phasors involved, through modulus of complex exponential coefficients $|X_k|$ versus frequency $kf_0$, yields **amplitude spectrum** (actually magnitude spectrum)
- shows **phases** of all phasors, through argument of the complex exponential coefficients $\theta_k$ versus frequency $kf_0$, yields **phase spectrum**

# Example

Consider the following signal, with period $T_0 = 2s$, so $f_0 = 0.5$ Hz, composed of three cosines and one sine:

$$x(t) = \cos(2\pi f_0 t) + 2\cos(2\pi 2 f_0 t) + \cos(2\pi 3 f_0 t) + \sin(2\pi f_0 t)$$

Real and complex Fourier coefficients are shown below




# Summary

The complex exponential Fourier series is given by

$$x(t) = \sum_{k=-\infty}^{\infty} X_k e^{jk\omega_0 t}$$

with coefficients

$$X_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-jk\omega_0 t} dt, \quad k \in \mathbb{Z}$$

The complex coefficients can be written in **polar form**: $X_k = |X_k| e^{j\theta_k}$

$$\text{with } \theta_k = \angle X_k = \arg(X_k) \text{ for } -\infty < k < \infty (k \in \mathbb{Z})$$

then, for real signals:

- magnitude: $|X_k| = |X_{-k}|$, **even** function of $k$ (and hence frequency $kf_0$) forming **amplitude spectrum**
- argument (or phase): $\theta_k = -\theta_{-k}$, **odd** function of $k$, forming **phase spectrum**

$\Longrightarrow$ periodic signal written in terms of sum of cosines and sines, together (pairwise) represented in complex exponentials

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

# Fourier Transform

## Contents


- Before starting…
- Fourier Transform
- Summary
- Exercises


## Before starting…

Let us start by taking a look into the results found in the previous chapter. We have defined the complex exponential Fourier series for a periodic signal $x(t)$ as:

$$x(t) = \sum_{k=-\infty}^{\infty} X_k e^{jk\omega_0 t}$$

with $\omega_0 = 2\pi f_0$ and $k \in \mathbb{Z}$ and complex coefficients found as

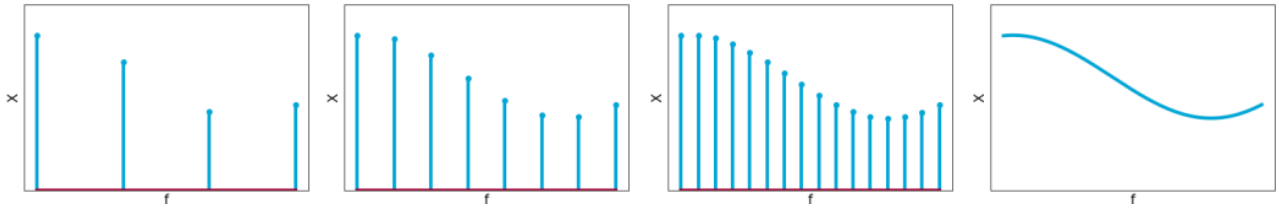
$$X_k = \frac{1}{T_0} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jk\omega_0 t} dt, \quad k \in \mathbb{Z}$$

where the integration period $T_0$ is taken symmetrically about zero.

## Fourier Transform

When the period $T_0$ increases, the frequencies belonging to the Fourier series coefficients ( $kf_0$) lie closer and closer together as $f_0 = 1/T_0$ decreases. Now, what if $T_0$ approaches infinity? That is, what would happen if we have an *a-periodic* signal?

Fourier coefficients will lie infinitesimally close to each other, so they define **continuous function of frequency**, $f \approx kf_0$



**Derivation**

> **MUDE Exam Information**
>
> This derivation is provided for additional insight and will not be part of the exam.

Substituting the definition of the series coefficients into $x(t)$, we find

$$x(t) = \sum_{k=-\infty}^{\infty} \left( f_0 \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} x(t) e^{-jk2\pi f_0 t} dt \right) e^{jk2\pi f_0 t}$$

Now, when $T_0 \to \infty$, $f_0$ becomes infinitesimally small, $f_0 \to df$. Multiplying $f_0$ with $k$ then becomes a continuous function, $kf_0 \to f$. The summation from $k = -\infty$ to $k = \infty$ then becomes an integration in $f$ from $f = -\infty$ to $f = \infty$

$$x(t) = \int_{-\infty}^{\infty} \left( df \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt \right) e^{j2\pi ft}$$

Re-arranging the terms:

$$x(t) = \int_{-\infty}^{\infty} \underbrace{\left( \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft}\, dt \right)}_{X(f)} e^{j2\pi ft}\, df$$

The integral between parentheses is defined as **Fourier integral** or (continuous-time) **Fourier transform**, $\mathcal{F}()$:

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$

The complex number $X_k$ of the Fourier series is now a complex **function** of the frequency, $f$: $X(f)$.

The **inverse Fourier transform**, $\mathcal{F}^{-1}()$ can also be derived

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df$$

> **Note**
>
> To obtain $x(t)$ from $X(f)$, we integrate over frequency, $f$, and the result is a function of time, $t$

Hence, Fourier transform and its inverse can be used to transform time signal to the frequency domain, and the other way around. This yields a **Fourier transform pair**:

$$x(t) \overset{\mathcal{F}}{\longrightarrow} X(f) \overset{\mathcal{F}^{-1}}{\longrightarrow} x(t)$$

# Conditions

**Sufficient** conditions for convergence of Fourier transform integrals are:

- $x(t)$ absolutely integrable, i.e. $\int_{-\infty}^{\infty} |x(t)| dt < \infty$
- discontinuities, if any, should be finite

# Amplitude and phase

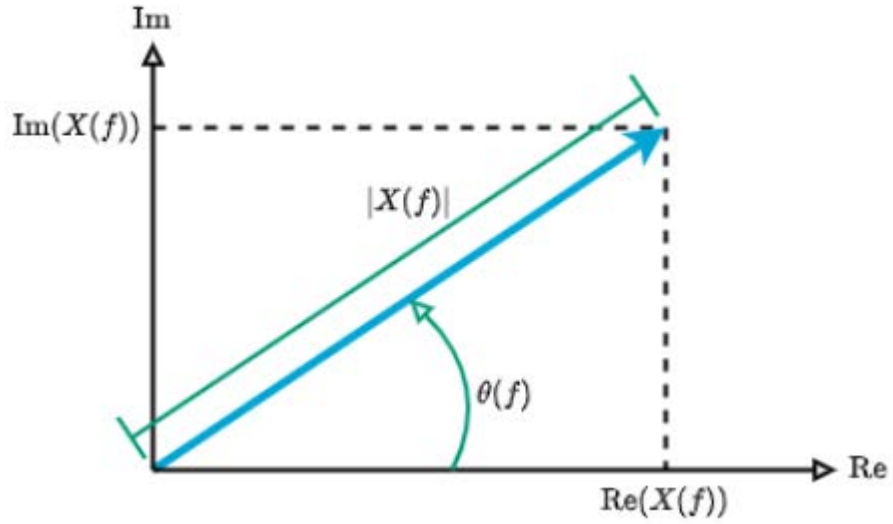
Like the coefficients of complex Fourier series, a Fourier transform can also be written in terms of **magnitude** and **phase**:

$$X(f) = |X(f)| e^{j\theta(f)}$$

with

$$|X(f)| = \sqrt{(\text{Re}(X(f)))^2 + (\text{Im}(X(f)))^2}$$

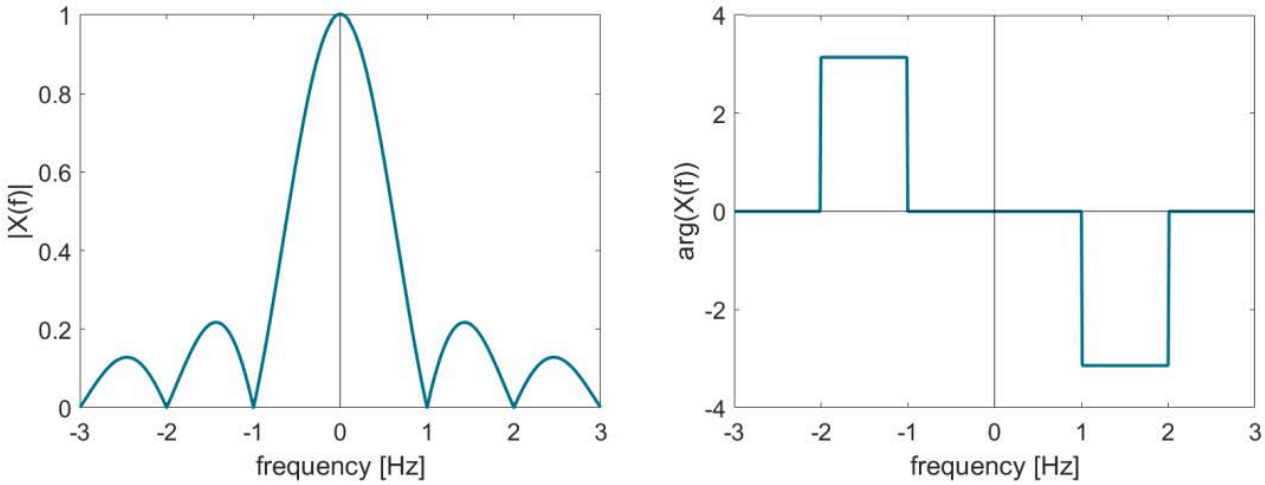
$$\theta(f) = \arctan\left(\frac{\text{Im}(X(f))}{\text{Re}(X(f))}\right)$$




When $x(t)$ is real, we have $X(f) = X^*(-f)$, and then

$$|X(f)| = |X(-f)| \text{ and } \theta(f) = -\theta(-f)$$

- Magnitude $|X(f)|$ is an **even** function of $f \rightarrow$ **amplitude spectrum**
- Phase $\theta(f)$ is an **odd** function of $f \rightarrow$ **phase spectrum**




The figure above illustrates an example of $|X(f)|$ and $\theta(f)$ both as a function of frequency.

# Summary

A-periodic functions can be written as integrals, **continuous** over **frequency**. This yields the so-called **Fourier tranform** (and its inverse):

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt$$
$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df$$

Amplitude and phase spectrum are given by:

$$|X(f)| = \sqrt{(\mathrm{Re}(X(f)))^2 + (\mathrm{Im}(X(f)))^2}$$
$$\theta(f) = \arctan\left(\frac{\mathrm{Im}(X(f))}{\mathrm{Re}(X(f))}\right)$$

# Exercises

**Fourier Transform Exercise (1 of 2)**

The signal $x(t) = \Pi(\frac{t}{\tau})$ is defined by a pulse function with duration $\tau$ [s]:

$$\Pi\left(\frac{t}{\tau}\right) = \begin{cases} 1, & \text{for } -\frac{\tau}{2} \leq t \leq \frac{\tau}{2}. \\ 0, & \text{otherwise.} \end{cases}$$

Derive an expression for the Fourier transform $X(f)$ of signal $x(t)$.

*Hint: you may want to use Euler's formula to turn the difference of two complex exponentials into a sine function. In addition, to turn your answer into an even more compact form, you may want to use the cardinal sine, which is defined as* $\mathrm{sinc}(z) = \frac{\sin(\pi z)}{\pi z}$.

## Solution

The video below illustrates how to arrive at the final solution, which is presented here:

$$X(f) = \tau\,\text{sinc}(\tau f)$$

The solution can also be found using SymPy:

```
import sympy as sym
```

```
t = sym.symbols('t')
tau, f = sym.symbols('tau, f', positive=True)
x = sym.Piecewise((0, t < -tau/2), (1, t < tau/2), (0, True))
display(x)
```

$$\begin{cases} 0 & \text{for} < -\frac{\tau}{2} \\ 1 & \text{for} < \frac{\tau}{2} \\ 0 & \text{otherwise} \end{cases}$$

```
solution = sym.integrate(x*sym.exp(-sym.I*2*sym.pi*f*t), (t,-sym.oo,sym.oo))
simplified_solution = sym.simplify(solution)
display(solution)
display(simplified_solution)
```

$$-\frac{ie^{i\pi f\tau}}{2\pi f} + \frac{ie^{-i\pi f\tau}}{2\pi f}$$

$$\frac{\sin(\pi f\tau)}{\pi f}$$

**Fourier Transform in the Limit Exercise (2 of 2)**

Given the Fourier transform $X(f) = \delta(f)$, find the expression for signal $x(t)$. Then, verify the following two points:

1. When $x(t) = e^{j2\pi f_0 t}$, then $X(f) = \delta(f - f_0)$, and
2. When $x(t) = \cos(2\pi f_0 t)$, then $X(f) = \frac{1}{2}(\delta(f - f_0) + \delta(f + f_0))$.

*Hint: for the first part (finding $x(t)$), use the inverse Fourier transform, and use the sifting property of the Dirac delta function (in this case in the frequency domain) when solving the integral.*

**Solution**

The video below illustrates how to arrive at the final solution, which is presented here:

$$x(t) = \cos(2\pi f_0 t)$$

**Attribution**

This chapter is written by Christiaan Tiberius. Find out more here.

# Sampling

## Contents

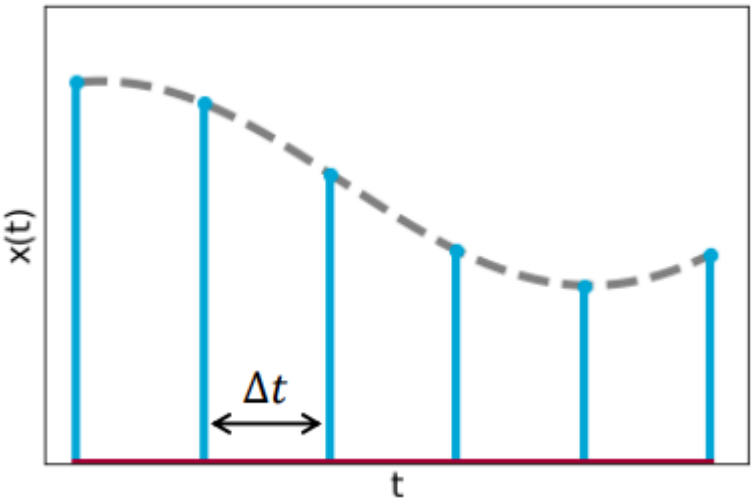

- Discrete and Quantized signals
- Singularity function
- Analog-to-digital conversion
- Fourier transform of sampled signal
- Sampling Theory
- Aliasing
- Summary
- Exercises


## Discrete and Quantized signals

**Discrete signals**

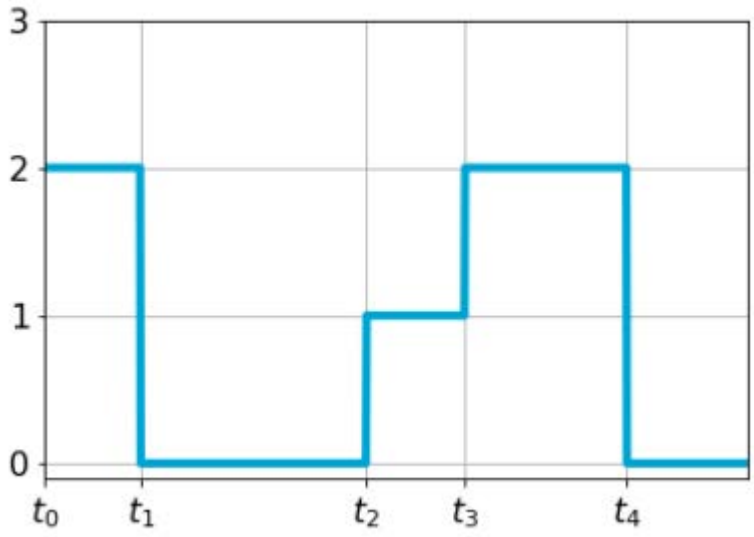
A **discrete** signal has a value only at discrete values of the running variable (usually time). Formally the signal is then to be referred to as *discrete-time* signal. The interval between these discrete values of running variable is often uniform, e.g. $\Delta t$. In between these values, the signal may be zero, undefined, or of no interest!

> **Note**
>
> **Continuous-time** signal is written as $x(t)$. A **discrete-time** signal is usually written as $x[n]$, or $x_n$ (sequence $x_0$, $x_1$, $x_2$...)




**Continuous signals**

A **quantized** signal may assume only countable number of values. Yet, changes from value to value may occur at any time. Think of describing an analog signal with 2 bits! The description of that signal can only have 4 possible values: `00`, `01`, `10` and `11`, which represent the values 0, 1, 2 and 3 in the figure below.




When we have more bits available, like 8, the signal description may have $2^8 = 256$ possible values. Clearly, the number of available bits defines the **resolution** of the quantization.

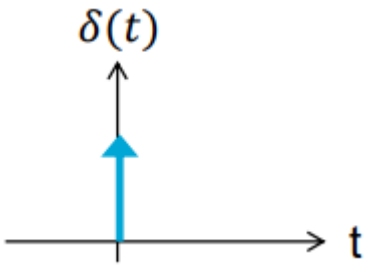
# Singularity function

## Unit impulse

A very important function for sampling is the **unit-impulse** or **Dirac-delta** function:

$$\delta(t) = 0, \quad t \neq 0 \text{ and } \int_{-\infty}^{\infty} \delta(t)dt = 1$$

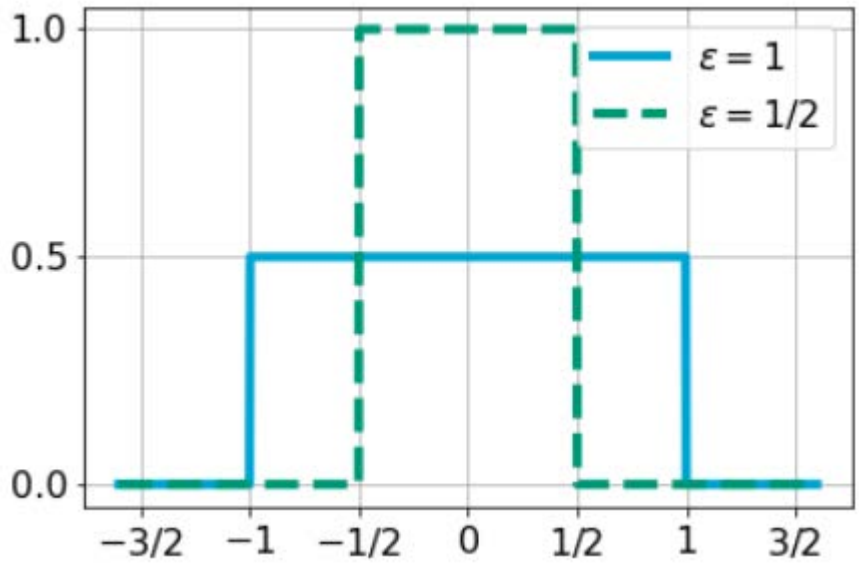
and its graph looks like:




The **area** of the function equals 1, and this area is obtained in **infinitesimal** interval of time. We use this function to represent phenomena that occur in **very short time intervals** compared to resolution capability of measuring device, but which produce almost instantaneous change in measured quantity.

However, no conventional function exists with properties of the Dirac delta pulse. Let us define:

$$\delta_\epsilon = \frac{1}{2\epsilon}\Pi\left(\frac{t}{2\epsilon}\right) = \begin{cases} \frac{1}{2\epsilon} : |t| \leq \epsilon \\ 0 : |t| > \epsilon \end{cases}$$

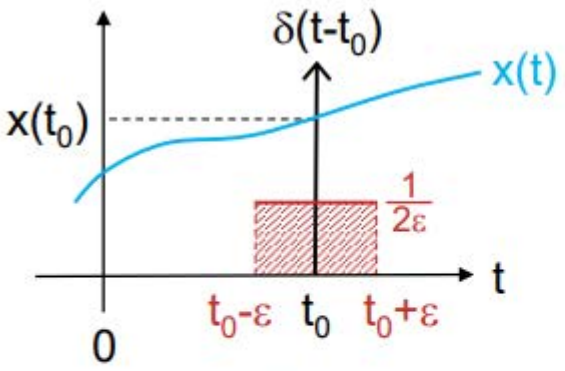


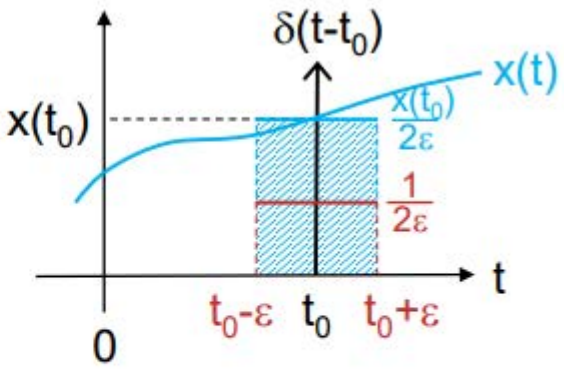

where $\epsilon \to 0$ yields a **unit impulse** function.

# Sifting property

A very useful property of the Dirac delta function is the **sifting property**:

$$\int_{-\infty}^{\infty} x(t)\delta(t-t_0)dt = x(t_0)$$



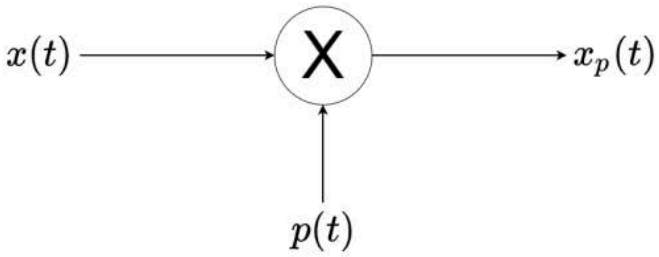



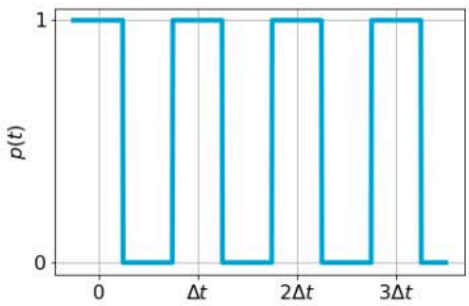

$$\delta_\epsilon(t-t_0) = \begin{cases} \frac{1}{2\varepsilon} & : \ |t-t_0| \le \varepsilon \\ 0 & : \ |t-t_0| > \varepsilon \end{cases}$$

The conclusion is that this blue area equals $x(t_0)$ (in the above diagram $\epsilon = 1$ was taken).

# Analog-to-digital conversion

Sampling can be modelled by multiplication of continuous-time signal $x(t)$ with sampling function $p(t)$.



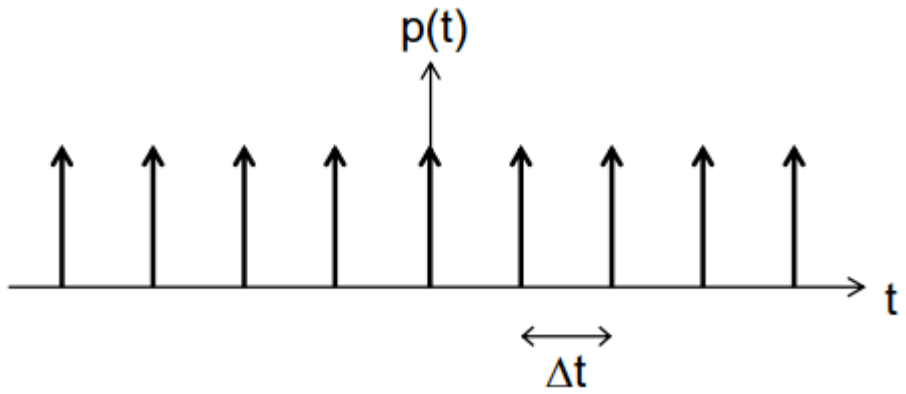

The sampled signal then becomes $x_p(t) = x(t)p(t)$.

The sampling function, $p(t)$, is assumed to be periodic pulse train with **sampling period** $\Delta t$ (note that earlier, $T$ or $T_0$ denoted the period of a periodic signal).

In practice, time during which $p(t)$ is non-zero (pulse width) is very small relative to period $\Delta t$.

In fact, in digital systems, where sample is the number corresponding to the value of the signal $x(t)$ at the sampling instant, $t$, the **pulse width** of the sampling function is **infinitelly**

**small**. Hence, we can model the sampling funtion as an **impulse train**




# Fourier transform of sampled signal

The impulse train $p(t) = \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t)$ is periodic ($n \in \mathbb{Z}$), with fundamental period $\Delta t$, and can be written as Fourier series:

$$p(t) = \sum_{k=-\infty}^{\infty} P_k e^{j2\pi k f_0 t}, \text{ with } k \in \mathbb{Z}$$

With fundamental frequency $f_0 = \frac{1}{\Delta t} = f_s$ (the sampling frequency), and Fourier coefficients $P_k$ as:

$$P_k = \frac{1}{\Delta t} \int_{-\frac{\Delta t}{2}}^{\frac{\Delta t}{2}} p(t) e^{-j2\pi k f_0 t} dt = \frac{1}{\Delta t} \int_{-\frac{\Delta t}{2}}^{\frac{\Delta t}{2}} \delta(t) e^{-j2\pi k f_0 t} dt = \frac{1}{\Delta t} = f_s$$

using the sifting property, and considering only one impulse occuring in the integration interval (at $t = 0$). The Fourier series of $p(t)$ then becomes:

$$p(t) = f_s \sum_{k=-\infty}^{\infty} e^{j2\pi k f_s t}$$

In the time domain, sampling is modelled as:

$$x_s(t) = \Delta t x(t) p(t)$$

multiplied with $\Delta t = \frac{1}{f_s}$ as a *correction factor* (to restore correct dimension; $p(t)$ has dimension [1/s] and we want signal $x_s(t)$ to have the same dimension as $x(t)$ itself).

Now, what happens to the spectrum $X(f)$ of signal $x(t)$ when it gets impulse train-sampled? What is $X_s(f)$ of $x_s(t)$?

**Derivation**

**MUDE Exam Information**

This derivation is provided for additional insight and will not be part of the exam.

The Fourier transform of $x_s(t)$ is:

$$X_s(f) = \int_{-\infty}^{\infty} \Delta t x(t) p(t) e^{-j2\pi ft} dt$$

Using the Fourier series of $p(t)$:

$$X_s(f) = \Delta t \int_{-\infty}^{\infty} \left( f_s \sum_{k=-\infty}^{\infty} e^{j2\pi k f_s t} \right) x(t) e^{-j2\pi ft} dt$$

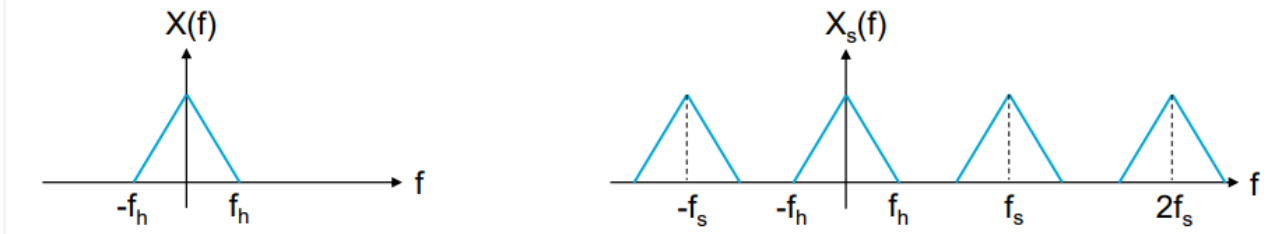
Interchanging integration and summation:

$$\begin{aligned} X_s(f) &= \underbrace{\Delta t f_s}_{1} \sum_{k=-\infty}^{\infty} \int_{-\infty}^{\infty} x(t) e^{j2\pi k f_s t} e^{-j2\pi ft} dt = \\ &= \sum_{k=-\infty}^{\infty} \underbrace{\int_{-\infty}^{\infty} x(t) e^{-j2\pi (f - k f_s) t} dt}_{X(f - k f_s)} \end{aligned}$$

Finally, the Fourier transform of the *sampled* signal becomes:

$$X_s(f) = \sum_{k=-\infty}^{\infty} X(f - k f_s), \quad \text{with } k \in \mathbb{Z}$$

so, the spectrum of the sampled signal is the spectrum of the original signal, but repeated with “period” $f_s$ (in the frequency domain); copies of spectrum are called *aliases.*




*Spectrum of an assumed original signal $x(t)$ (left) and that of its sampled equivalent (right)*

# Sampling Theory

In practice, we should choose a *sample rate* that is capable of representing the frequency components of a continuous signal as a discrete sequence. The term *rate* is quantified in units of samples per second, whereas a *frequency* component of the signal is described using cycles per second; both are represented by units of frequency, Hertz (Hz).

Consider a **band-limited** signal, $x(t)$, which has no frequency components above $f_h$ Hz. Such a signal can be *completely specified* by samples taken at a *uniform* sample rate greater than $2f_h$ Hz.

**Definition: Nyquist *rate* and Nyquist *frequency.***

If $f_h$ is the maximum frequency of a band-limited signal, the quantity $2f_h$ is called the **Nyquist rate.**

If the same signal is sampled at rate $f_s$, the quantity $f_s/2$ is called the **Nyquist frequency.**

Note that the Nyquist *rate* is related to the *frequency* of a signal, and the Nyquist *frequency* is related to the sampling. In other words, the Nyquist rate is characteristic of the signal, whereas the Nyquist frequency is characteristic of the sampling system!

In practice we consider only the domain $-\frac{f_s}{2} < f < \frac{f_s}{2}$ of the spectrum obtained from the sampled signal.

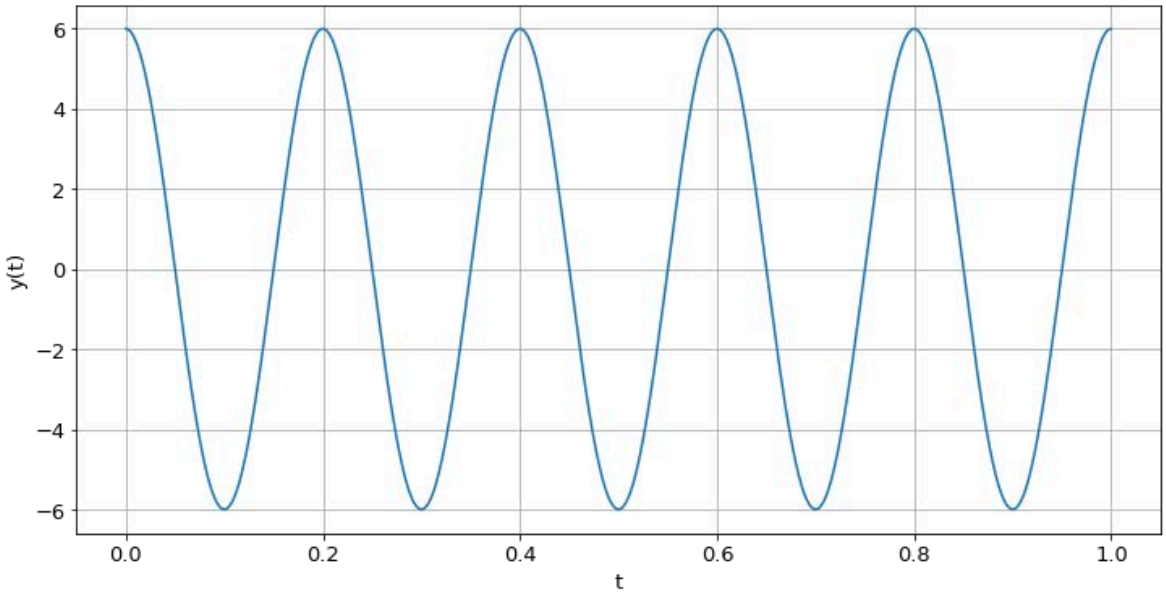
Although many signals in practice are not band-limited, by considering the maximum frequency of interest for a particular application, the Nyquist rate can be applied to determine a suitable sampling strategy; specifically, to avoid aliasing.

# Aliasing

Note that in order to reconstruct original continuous-time signal from samples, it is crucial to **sample the signal at a rate larger than Nyquist rate**. When sampling is below this rate, the adjacent spectra (aliases) will overlap, and it will be impossible to reconstruct the signal from its samples. This phenomenon is called **aliasing**, and is illustrated in the following two examples.

### Theoretical Example

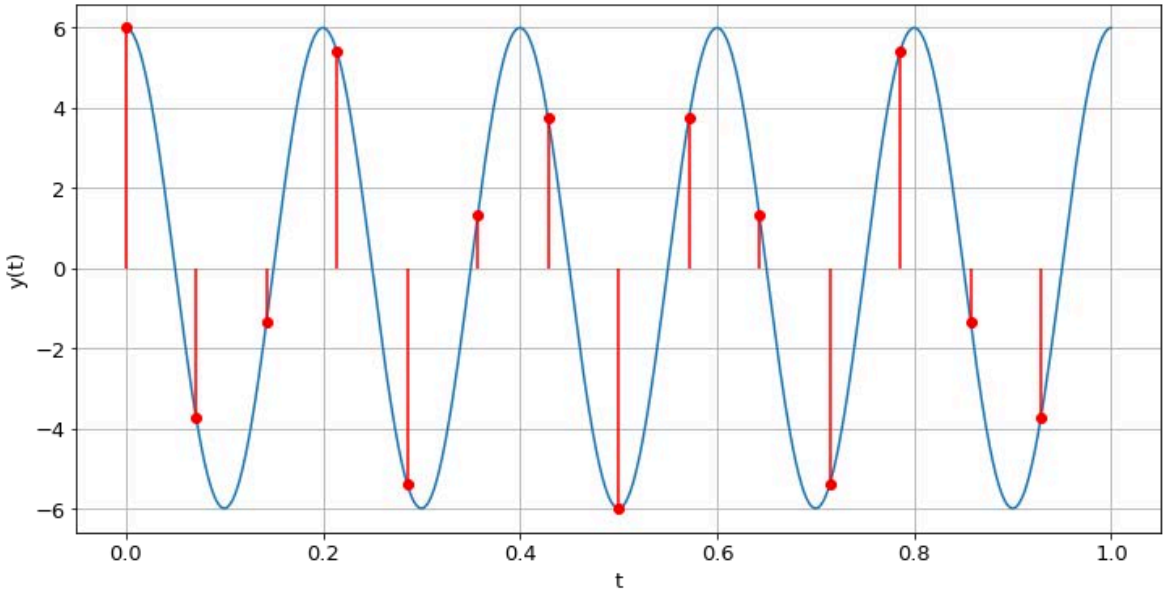
As an example, we will study the effect of sampling a sinusoidal signal with frequency $f_c = 5$ Hz we have $x(t) = 6\cos(10\pi t)$




First, we look at the *correctly* sampled signal, with $f_s = 14$ Hz ($f_s > 2f_c$). The spectrum (which is real, because $x(t)$ is even) of original continuous-time signal will have two Dirac-functions with weight 3, at $f = 5$ Hz and $f = -5$ Hz, i.e.

$$X(f) = \frac{6}{2}(\delta(f-5) + \delta(f+5))$$

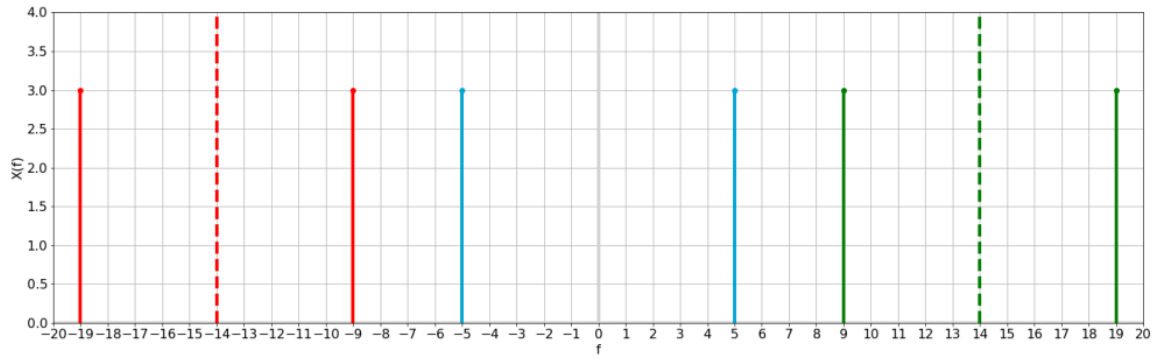
Sampled at 14 Hz, we find:




### Spectrum

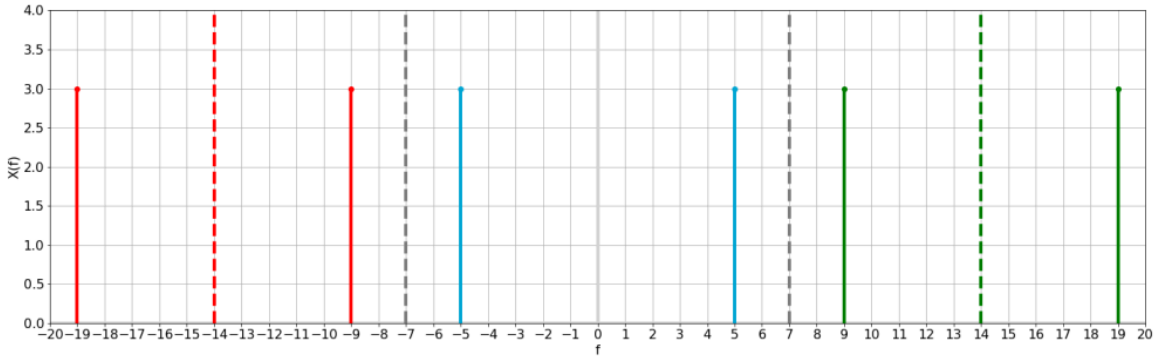
With $x(t) = 6\cos(10\pi t)$ and $f_s = 14$ Hz:

$$X_s(f) = 3\sum_{k=-\infty}^{\infty}(\delta(f+5-kf_s)+\delta(f-5-kf_s)), \text{ with } k \in \mathbb{Z}$$

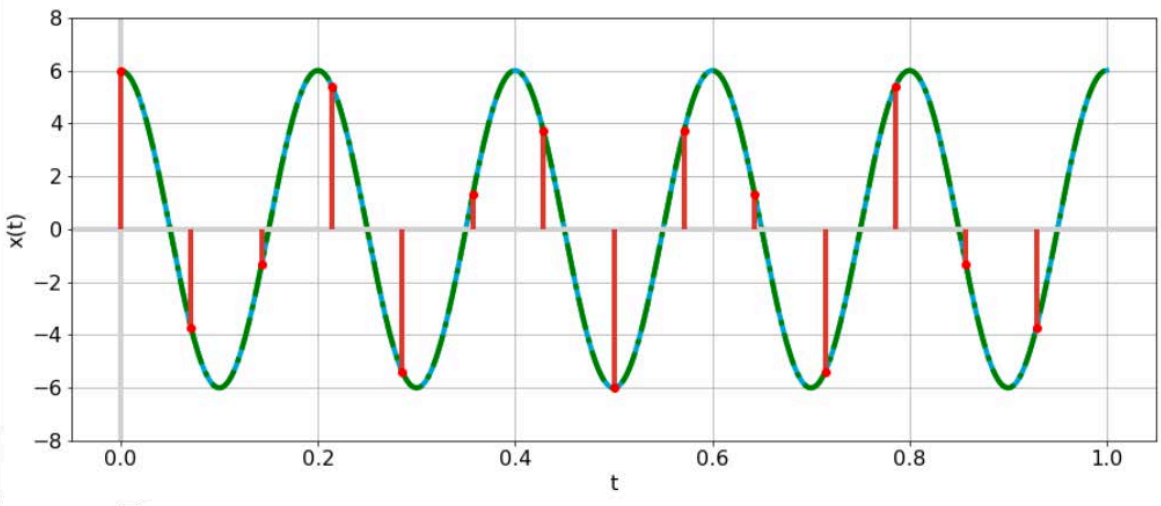



$k = -1$ $k = 0$ $k = 1$

The dashed lines indicate integer multiples of the sampling frequency. As we are only considering the domain $-\frac{f_s}{2} < f \leq \frac{f_s}{2}$:

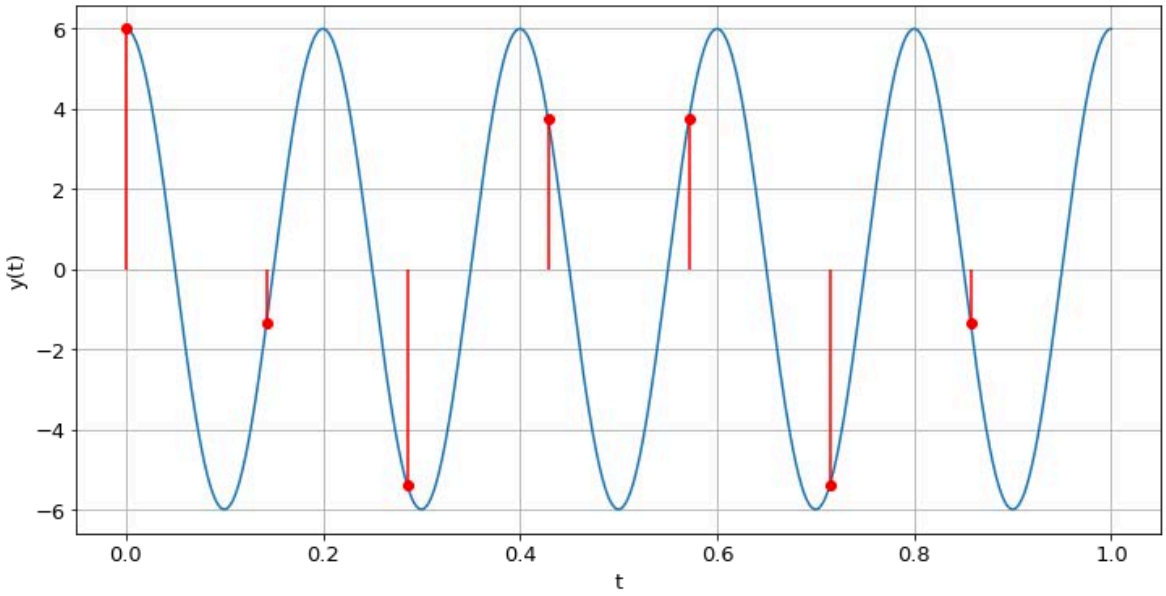



And we see that with this sampling we get a **correct result** (just the two original Dirac delta pulses in blue)!

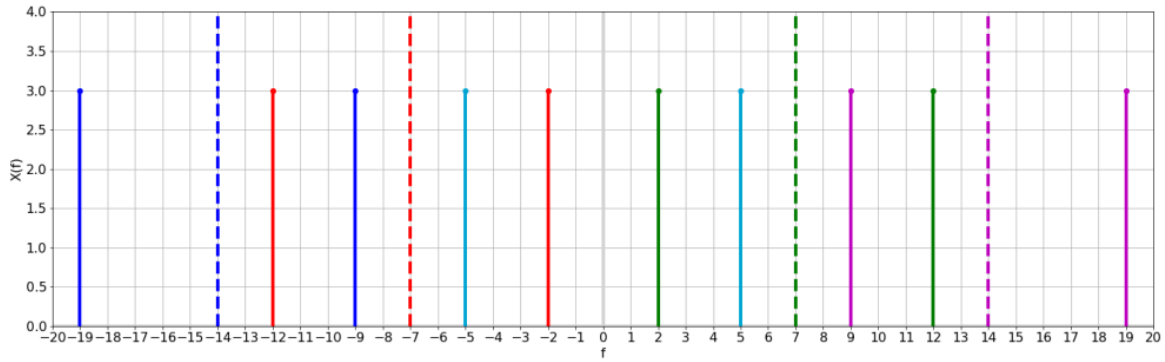



*But what if we try to sample our signal using a sampling frequency of 7 Hz?*

Well, with this we violate the sampling theorem, as $f_s = 7$ Hz and, therefore, $f_s \ngeq 2f_c$. We find:

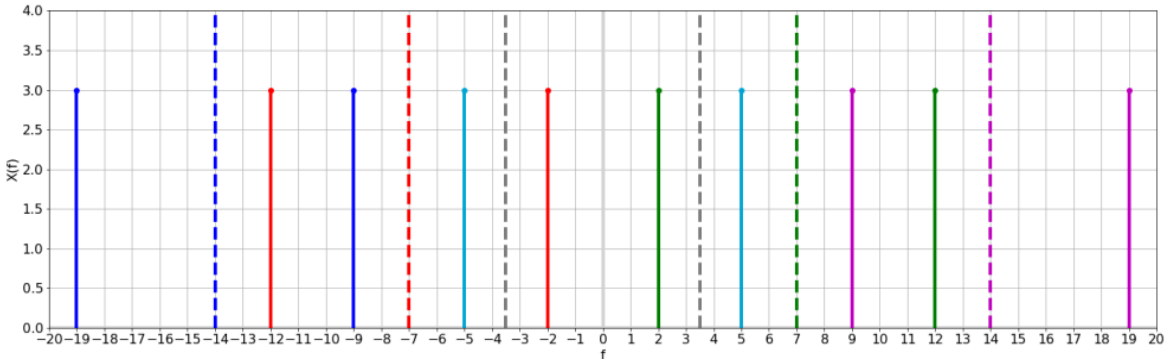



With this, we find the following spectrum:

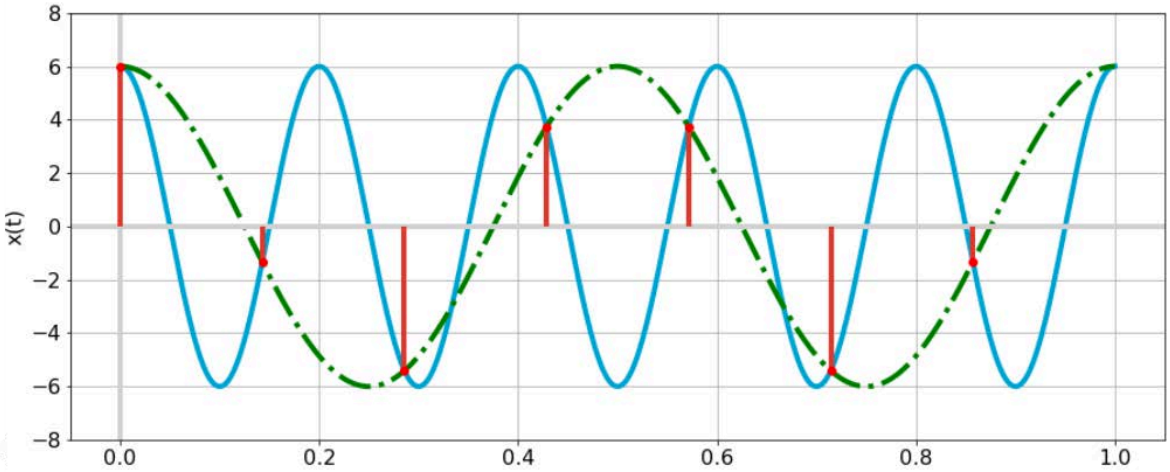



$k = -2$ $k = -1$ $k = 0$ $k = 1$ $k = 2$

Once again, considering only the domain $-\frac{f_s}{2} < f \leq \frac{f_s}{2}$ represented in grey, we find:




And we see we find an **incorrect result**, as the sampled and reconstructed signal (in dashed green) presents a lower frequency than the original signal (in blue) we encounter Dirac delta pulses in green and red at 2 Hz and -2 Hz:


8
6
4
2
0
−2
−4
−6
−8
x(t)
0.0
0.2
0.4
0.6
0.8
1.0

**Example in Practice**

The phenomenon illustrated in this video is known as the “wagon wheel effect,” and, more generally as the *stroboscopic effect* (and you can find loads of videos on it in addition to the one her). This phenomenon is a result of aliasing.

Suppose you have a wheel with, hypothetically, only one spoke connecting the rim to the axle. By watching the spoke we can tell whether, and how, the wheel is turning. If this wheel would be turning at a frequency of $f_c$ = 1 cycle per second (Hz), and we would capture frames with a video camera at exactly $f_s$ = 1 Hz (one frame per second), then we would see the spoke in exactly the same position in all frames recorded by the camera; hence, it looks like the wheel is standing still in our video! Of course in reality the wheel is turning. In this case the video, a series of images taken at discrete time instants, gives us a false impression about reality.

Connecting this to Nyquist sampling: the wheel rotation frequency is $f_c$ = 1 Hz, and hence the Nyquist rate is 2 Hz, and our sampling frequency $f_s$ = 1 Hz is clearly below it (we’re sampling too slowly). The spectrum (with frequencies $f_c$ = -1 and +1 Hz) gets copied to integer multiples of the sampling frequency $f_s$ (e.g. at 1 Hz), hence a copy of $f_c$ = -1 Hz, shows up as an alias at $f$ = 0 Hz.

The example in the video is slightly more involved as the wheel has 7 identical spokes, causing the frequency of interest ($f_c$) to be actually 7 times as large. We achieve an identically looking image already after 1/7th turn of the wheel.

In the video, at some point, the wheel is turning backward: not in reality, but in our observations based on the sampled signal. At this point we were driving slightly over 30 km/h, which is 8.9 m/s. With a wheel circumference of 1.92 m, the turning

frequency is 4.6 Hz. With 7 identical spokes, the periodic signal has a frequency of 7 times 4.6 Hz, which yields $f_c$ = 32.4 Hz. The camera on the smartphone captures the video at 30 frames per second (fps), hence $f_s$ = 30 Hz. So, we can expect an alias of $f_c$ at a frequency of -2.4 Hz (a negative frequency, materialized by the apparently a backward turning wheel).

# Summary

The Fourier transform of a **sampled signal**, $x_s(t)$, is given as:

$$X_s(f) = \sum_{k=\infty}^{\infty} X(f - kf_s)$$

where $f_s$ is the sampling frequency.

To prevent aliases, this frequency $f_s$ should be larger than $2f_h$, where $f_h$ is the highest frequency occurring in the signal!

# Exercises

Given a signal $x(t)$ that is sampled at frequency $f_s$, what does $X_s(f)$ look like? The following quiz questions will test your knowledge.

Sinusoidal signal $x(t) = 10\cos(4\pi t)$ is (impulse train) sampled at 5 Hz. By looking at the Fourier transform of the sampled signal, at which frequency, starting from zero, for positive frequencies, do we encounter the first peak (Dirac delta pulse)? And, does aliasing occur in this case? The first peak occurs at:

- ○ 2 Hz; yes, aliasing occurs
- ○ 3 Hz; yes, aliasing occurs
- ○ 1 Hz; yes, aliasing occurs
- ○ 2 Hz; there is no aliasing

Check

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

# Discrete Fourier Transform

## Contents


- Discrete time Fourier Transform (DTFT)
- Discrete Fourier Transform (DFT)
- Inverse Fourier Transform (IDFT)
- Summary


## Discrete time Fourier Transform (DTFT)

We discretize Fourier transform of $x_s(t)$

$$X_s(f) = \int_{-\infty}^{\infty} x_s(t) e^{-j2\pi ft} dt$$

by replacing the integral by summation over time:

$$X_s(f) = \sum_{n=-\infty}^{\infty} \Delta t x_n e^{-j2\pi fn\Delta t}$$

with $x_n = x(n\Delta t)$ and where complex exponential is also evaluated at times $t = n\Delta t$, and $n \in \mathbb{Z}$.

This is still a **continuous function** of frequency $f$ ($f \in \mathbb{R}$), periodic with period $f_s$, exactly as we got with impulse train sampling, and known as Discrete Time Fourier Transform (DTFT).

## Discrete Fourier Transform (DFT)

We want to analyze spectrum $X_s(f)$ of *sampled* signal $x_s(t)$ using a computer, i.e. by Digital Signal Processing (DSP). Two issues remain, however:

$$X_s(f) = \sum_{n=-\infty}^{\infty} \Delta t x_n e^{-j2\pi f n \Delta t}$$

- we cannot measure the signal forever, so we cannot have $n \to \infty$
- once sampled in time domain, still continuous function (of frequency) does not lend itself to DSP - algorithm requires *discrete* data points as input, and delivers *discrete* data points as output

**Action item 1 - we cannot measure signal forever**

Sampled signal $x_s(t)$ is a sequence $x_n$ with $n = \{-\infty, \ldots, \infty\}$, so values $x_n = x(n\Delta t)$ at discrete times $t = n\Delta t$ (with $n \in \mathbb{Z}$).

We consider it only for time duration $T = N\Delta t$, resulting in just N samples; effectively this means applying a window $\Pi\left(\frac{t}{T}\right)$; in practice we set the signal to zero outside window, hence: $x_{sw}(t) = \Pi\left(\frac{t}{T}\right) x_s(t)$

This means $x_n$ with $n = 0, \ldots, N-1$ has **finite length**

**Action item 2 - continuous function does not lend itself to DSP**

Sample frequency spectrum: we will evaluate it only at *discrete* frequencies.

As we only use piece of $T = N\Delta t$ of the signal, the *smallest* resulting frequency is known as frequency (or spectral) **resolution** and it is given by

$$f_0 = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{f_s}{N}$$

and the *largest* frequency is related to the sampling frequency by $f_s = \frac{1}{\Delta t}$. Hence, the spectrum will be computed at frequencies

$$f = 0, \frac{1}{N} f_s, \frac{2}{N} f_s, \ldots, \frac{N-1}{N} f_s$$

This results in the so-called **Discrete Fourier Transform (DFT)**. DFT turns $N$ samples of signal $x(t)$ into $N$ samples of spectrum $X_{sw}(f)$:

$$x(n\Delta t) \leftrightarrow X_{sw}(k f_0)$$

with both $n$ and $k \in \{0, 1, \ldots, N-1\}$.

**Frequency sampling (background information)**

By default, analysis frequencies are $f = 0, \frac{1}{N} f_s, \frac{2}{N} f_s, \ldots, \frac{N-1}{N} f_s$ or, in terms of the sampling duration, $T$: $f = 0, \frac{1}{T}, \frac{2}{T}, \ldots, \frac{N-1}{T}$

Sampling the spectrum at interval of $\frac{1}{T}$ Hz turns sampled time signal into *periodic* (instead of windowed) signal with period $T$. This choice causes DFT to turn $N$ samples of $x(t)$ into $N$ samples of $X_{sw}(f)$.

Sampling frequency at *higher* rate (smaller interval in frequency) is possible. On the other hand, sampling at *lower* rate is **not allowed**. A longer interval of, e.g. $\frac{2}{T}$ Hz would cause the signal to repeat every $\frac{T}{2}$, thus causing **aliasing** in the time domain

Going back to our action items:

**Action item 1**:

Sampled, windowed signal $x_{sw}(t)$ equals sequence $x_n$ with $n = 0, 1, \ldots, N-1$. Then, $X_{sw}(f) = \int_{-\infty}^{\infty} x_{sw}(t) e^{-j2\pi ft} dt$, which we discretize with step size $\Delta t$ into:

$$X_{sw}(f) = \sum_{n=0}^{N-1} \Delta t \, x_n e^{-j2\pi fn\Delta t}$$

**Action item 2**:

Finally, we sample the frequency spectrum, turning $X_{sw}(f)$ into $X_{sws}(f)$ by considering only $f = k\Delta f$ with $\Delta f = f_0 = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{f_s}{n}$ and $k = 0, 1, \ldots, N-1$:

$$X_{sws}(k\Delta f) = \Delta t \sum_{n=0}^{N-1} x_n e^{-j2\pi k\Delta fn\Delta t} = \Delta t \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}, \text{ with } k \in \mathbb{Z}$$

Hence sequence $X_k$ equals $X_{sws}(f)$ at $f = k\Delta f$ for $k = 0, 1, \ldots, N-1$:

$$X_k = \Delta t \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}$$

This is the discrete Fourier transform (DFT), typically implemented in software packages as `fft` (in Python, we will use `numpy.fft.fft`).

# Inverse Fourier Transform (IDFT)

The Inverse Fourier Transform reads $x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df$. Applying it to the sequence $X_k$ and discretizing the integral yields:

$$x_{sws}(t) = \sum_{k=0}^{N-1} \Delta f X_k e^{j2\pi k \Delta f t}$$

Considering sampling in the time domain as $t = n\Delta t$:

$$x_n = \sum_{k=0}^{N-1} \Delta f X_k e^{j2\pi k \Delta f n \Delta t} = \Delta f \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kn} = \frac{1}{N\Delta t} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kn}$$

# Summary

$$X_k = \Delta t \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}$$
$$x_n = \frac{1}{N\Delta t} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kn}$$

with both $k$ and $n \in \{0, 1, \ldots, N-1\}$

With $X_k$, we consider function $X(k\Delta f)$ by restoring *frequency dimension*, frequency resolution, $\Delta f = f_0 = \frac{1}{T} = \frac{1}{N\Delta t} = \frac{f_s}{N}$

With $x_n$, we consider function $x(n\Delta t)$ by restoring time dimension, with time resolution $\Delta t = \frac{1}{f_s}$

In many textbooks we also find DFT as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}$$
$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kn}$$

with both $k$ and $n \in \{0, 1, \ldots, N-1\}$

Hence, *without* factors $\Delta t$ and $\frac{1}{\Delta t}$. This is also how DFT is implemented in programming languages like Matlab and Python; the user has to restore time and frequency dimension!

> **Attribution**
>
> This chapter is written by Christian Tiberius. Find out more here.

# Spectral Estimation

## Contents


- Energy and power signals
- Parseval's theorem
- Power Spectral Density (PSD)
- Summary


## Energy and power signals

The amplitude of an harmonic wave is related to its energy (denoted by $E$) and power (denoted by $P$). In this chapter we introduce the Power Spectral Density (PSD) which describes how the signal power is distributed over frequency. For later analyses it is relevant to know how much power is contained at what frequencies.

The PSD is used with many different observed signals (e.g. displacement in [m], and acceleration in [m/s$^2$]), maintaining the notions of power and energy, but the classical development originates from electrical engineering, starting with the fact that an electrical sensor delivers a voltage, which hopefully is proportional to the observed physical phenomenon, in which we're interested.

We'll start with a flashback to high-school physics, and use Ohm's law. Suppose $u(t)$ is voltage across a resistor $R$ producing a time-varying current $i(t)$.

Instantaneous power is defined as $p(t) = u(t)i(t)$, and $u(t) = i(t)R$, so $p(t) = i^2(t)R = \frac{u^2(t)}{R}$. With $R = 1\Omega$, the instantaneous power, in Ohm, is given as:

$$p(t) = u(t)i(t) = i^2(t) = u^2(t)$$

For 1 Ohm resistor, power $P$ in Watt [W], equals the square of the voltage $u(t)$ [V$^2$]. The total energy $E$, in Joule [J], is obtained by integrating the power $P$ (power equals energy per unit time [W = J/s]).

Integrating over $|t| \leq T$, we define **total energy** and **average power** as

$$E = \lim_{T\to\infty} \int_{-T}^{T} u^2(t)dt \ \ [J]$$
$$P = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} u^2(t)dt \ \ [W]$$

For signal $x(t)$, the **total energy**, normalized to a unit resistance, is defined similarly as:

$$E = \lim_{T\to\infty} \int_{-T}^{T} |x(t)|^2 dt$$

and **average power**, normalized to a unit resistance, as:

$$P = \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} |x(t)|^2 dt$$

For real signals, the modulus signs can be removed from the two equations above.

# Parseval's theorem

**Definition**

Average (normalized) power of signal in **time domain** is the same as the average (normalized) power of corresponding a **signal in the frequency domain**; and similarly for total (normalized) energy.

**Derivation**

**MUDE Exam Information**

This derivation is provided for additional insight and will not be part of the exam.

## Fourier Series

**Average power** of **periodic** waveform $x(t)$ with period $T_0$ can be written as:

$$P = \frac{1}{T_0}\int_{T_0} |x(t)|^2 dt$$

Since $|x(t)|^2 = x(t)x^*(t)$, we can replace $x^*(t)$ with its complex exponential Fourier series:

$$P = \frac{1}{T_0}\int_{T_0} x(t)\left(\sum_{k=-\infty}^{\infty} X_k^* e^{-j2\pi k f_0 t}\right)dt = \sum_{k=-\infty}^{\infty} X_k^* \underbrace{\frac{1}{T_0}\int_{T_0} x(t)e^{-j2\pi k f_0 t}\,dt}_{X_k}$$

Where we just interchanged the order of summation and integration.

Now, we may write:

$$P = \frac{1}{T_0}\int_{T_0} |x(t)|^2 dt = \sum_{k=-\infty}^{\infty} |X_k|^2 = X_0^2 + 2\sum_{k=1}^{\infty} |X_k|^2$$

because $|X_k|$ is even.

In other words, **average power** of a **periodic** signal is simply the sum of powers in phasors of its Fourier Series, or just the sum of the squared moduli of its complex Fourier series coefficients.

> **Note**
>
> $X_0$ is signal average and $X_k$ are harmonic components

We derived that, for **periodic** signals $x(t)$, the **average power** in the time domain equals the average power in the frequency domain:

$$P = \frac{1}{T_0}\int_{T_0} |x(t)|^2 dt = \sum_{k=-\infty}^{\infty} |X_k|^2$$

This is known as **Parseval's theorem**.

# Fourier Transform

Now we can also show that a similar theorem exists to express the total energy in *aperiodic* signals! We start with expressing the **total energy** of the **aperiodic** signal in time and frequency domain using an inverse Fourier transform:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} x^*(t) \left( \int_{-\infty}^{\infty} X(f) e^{j2\pi ft} df \right) dt$$

Reversing the integration order:

$$E = \int_{-\infty}^{\infty} X(f) \left( \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt \right)^* df = \int_{-\infty}^{\infty} X(f) X^*(f) df$$

We obtain Parseval's theorem for Fourier transforms that can be expressed as:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(f)|^2 \, df$$

Energy in the timedomain equals energy in the frequency domain.

# Discrete Fourier Transform

Starting from a discrete, finite sequence of samples, $x_n$, in the time domain, the **average power** start as:

$$P = \frac{1}{T} \sum_{n=0}^{N-1} |x_n|^2 \, \Delta t$$

where we have discretized the integral over $x^2(t)$, which then becomes:

$$P = \frac{1}{N} \sum_{n=0}^{N-1} |x_n|^2 = \frac{1}{N^2 \Delta t^2} \sum_{k=0}^{N-1} |X_k|^2 = \frac{1}{T^2} \sum_{k=0}^{N-1} |X_k|^2$$

Note that $X_k$ denotes DFT-coefficients, with $\Delta t$ included ($X_k = \Delta t \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}$ ). The full proof is ommitted (if you think this optional derivation box is long, you don't want to see that one!).

For a sampled signal, with the coefficients $X_k$, obtained through the DFT (with the $\Delta t$ included), the power of the signal, contained in a frequency band of width $\Delta f = \frac{1}{T}$, at frequency $f = k\Delta f$ is:

$$S(k\Delta f) = \frac{1}{T}|X_k|^2 \quad \text{for} \quad k = 0, \ldots, N-1$$

and this is actually the **power density** [W/Hz].

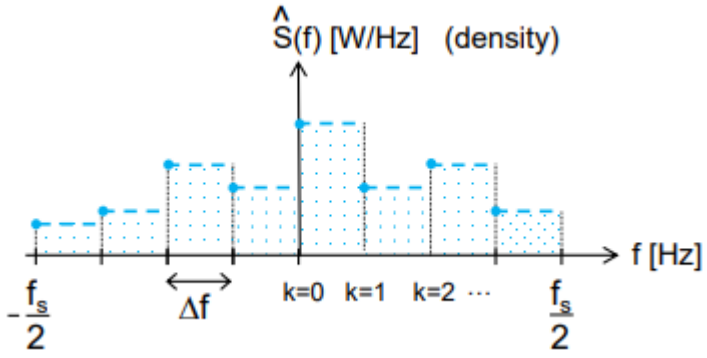
Returning to a voltage signal $x(t)$, or its sampled, discrete time version $x_n$, we have, see the previous chapter on the DFT, that $X_k$ is in [Vs] or equivalently in [V/Hz]. Then the unit of $S$ equals [$V^2$ s], which equals, with the above exposition on the 1 Ohm resistor, [W/Hz].

# Periodogram

This turns out to be an *estimate* for the power spectral density (PSD), and it is referred to as a **periodogram** (estimate may be indicated by a hat-symbol, hence $\hat{S}$).

The product $\Delta f S(k\Delta f)$ will, therefore, represent the contribution by frequency band with width $\Delta f$ at a frequency $f = k\Delta f$, to a power $P$ of the signal.

The periodogram, $S(f)$, defined for $0 \leq f < f_s$ or, equivalently, $-\frac{f_s}{2} < f \leq \frac{f_s}{2}$, with $f_s = \frac{1}{\Delta t}$ will be given by:




# Power Spectral Density (PSD)

$S(f)$ represents the **power spectral density**, with $P$ being given trivially (considering the definition of power spectral **density**) by:

$$P = \int_{-\infty}^{\infty} S(f)df$$

Integrating the PSD over frequency yields the power contained in the signal.

# Summary

The **energy** and the **power** of a signal are given by:

$$E = \lim_{T \to \infty} \int_{-T}^{T} |x(t)|^2 dt$$
$$P = \lim_{T \to \infty} \frac{1}{2T} \int_{-T}^{T} |x(t)|^2 dt$$

Finally, we derived and gave a formal definition for **Parseval's theorem** which reads (for an a-periodic signal) as:

$$E = \int_{-\infty}^{\infty} |x(t)|^2 dt = \int_{-\infty}^{\infty} |X(f)|^2 df$$

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

# Supplementary Videos

The story of Signal Processing is told once again on this page through a series of videos. Besides an introduction and conclusion, the videos listed below have a one-to-one correspondence with the chapters in this book.

- Introduction
- Fourier Series
- Complex Fourier Series
- Fourier Transform
- Sampling
- Discrete Fourier Transform
- Spectral Estimation
- Conclusions

*In case small errors are present in the videos, please refer to the appropriate book chapter.*

**MUDE exam information**

These videos overlap to some extent with the theory presented in the book, and are meant to provide additional perspective on the same topic. Additional material that is presented in these videos is *not* part of the exam; in other words, the exam scope is limited to contents that appear in the previous pages.

**Introduction**

## Fourier Series

## Complex Fourier Series

**Fourier Transform**

**Sampling**

## Discrete Fourier Transform

## Spectral Estimation

**Conclusions**

> **Attribution**
>
> This chapter is written by Christiaan Tiberius. Find out more here.

# Time Series Analysis

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

In this chapter, we will first introduce the components to describe a time series: trend, signal, offsets, irregularities and noise. It will then be shown how to estimate the signal-of-interest (everything except noise).

Next, we will consider stationary time series, meaning that the statistical properties do not depend on the time when the time series was observed. The stationary time series describe an underlying stochastic process, which can then be modelled, for instance using an Autoregressive (AR) model. The final goal is to use the time series for estimating the components such as trend and seasonality, as well as to predict future values, for which we do need to take into account the stochastic process.

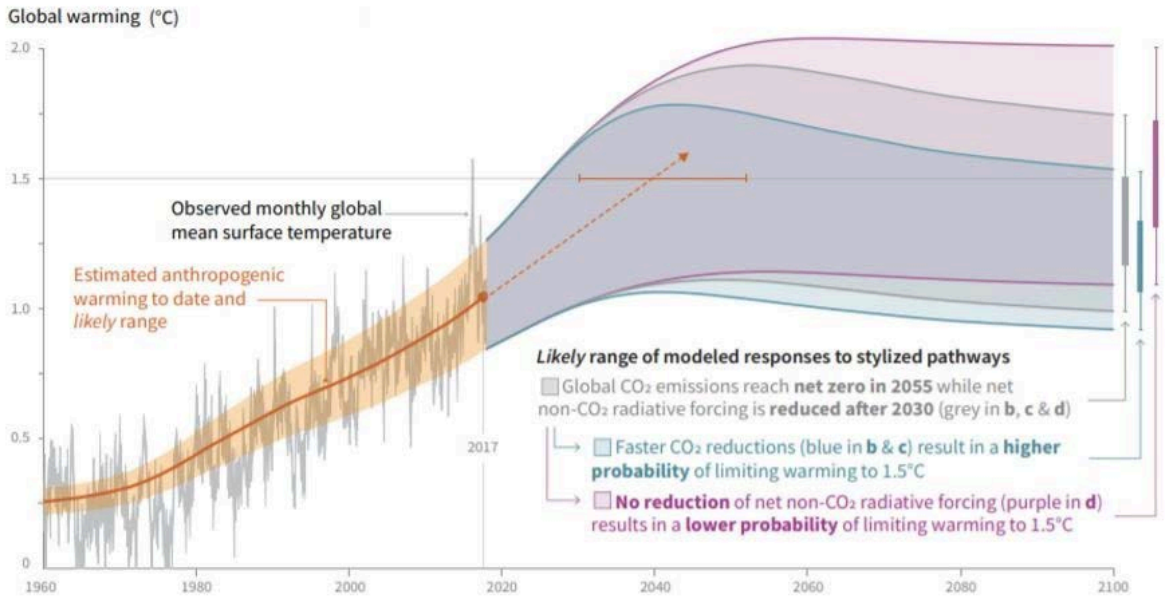



*Fig. 4.17* Recorded and expected global warming from 1960 to 2100, from IPCC (2018).

# Components of time series

## Contents


- Trend
- Seasonality
- Offset (jump)
- Noise


A time series is a discrete time sequence of data points indexed in time which can be used to study a phenomenon. It is a record of the data collected at different points in time, it consists of discrete time samples of typically a continuous-time phenomenon in reality. The data are usually collected at fixed time intervals rather than just recording them intermittently or randomly. The fixed interval $\Delta t$, in the time domain, is defined as ‘sampling interval’ and, in the frequency domain, is defined as ‘sampling rate’ or ‘sampling frequency’, expressed for example in Hz.

$$\Delta t = \frac{1}{f_s}$$

A time series is denoted as

$$Y(t) = [Y(t_1), Y(t_2), \dots, Y(t_m)]^T$$

The $Y(t_i)$ are random variables, since the data is affected by noise.

The time instants, also defined as epochs, are $t_i = i\Delta t$, indicating that the samples are equally spaced in time intervals of $\Delta t$. Assuming a unit time interval (i.e., $\Delta t = 1$), then $t_i = i$ and we can write the time series as

$$Y(t) = [Y(1), Y(2), \dots, Y(m)]^T = [Y_1, Y_2, \dots, Y_m]^T$$

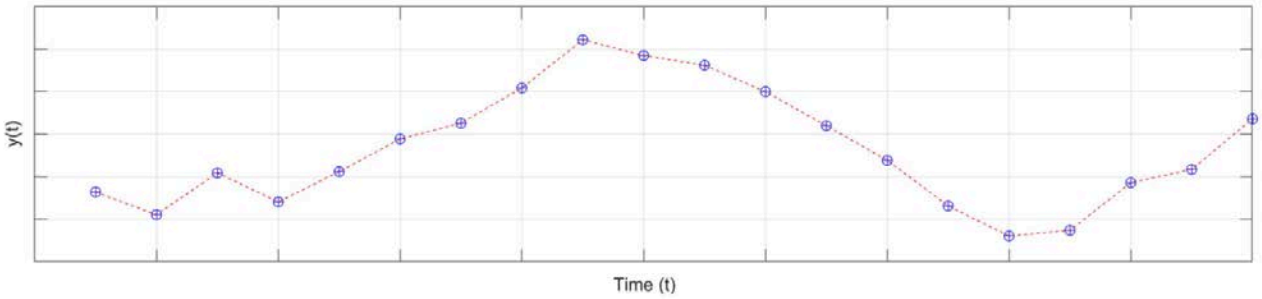



*Fig. 4.18* Example of time series with equally spaced time interval $\Delta t$

A time series can be decomposed as follows:

$$Y(t) = tr(t) + s(t) + o(t) + b(t) + \epsilon(t)$$

where we distinguish the following components:

1. $tr(t)$ = trend, provides the general behavior and variation of the process
2. $s(t)$ = seasonality, shows the regular seasonal variations
3. $o(t)$ = offset, is a discontinuity (or jump) in the data
4. $b(t)$ = irregularities and outliers (also referred to as biases), due to unexpected reasons. Irregularities will not be considered in this book.
5. $\epsilon(t)$ = noise process, can be white or colored noise.

# Trend

The trend is the general pattern of the time series and shows its long-term changes. The trend can be linear, however higher order polynomials are also possible.

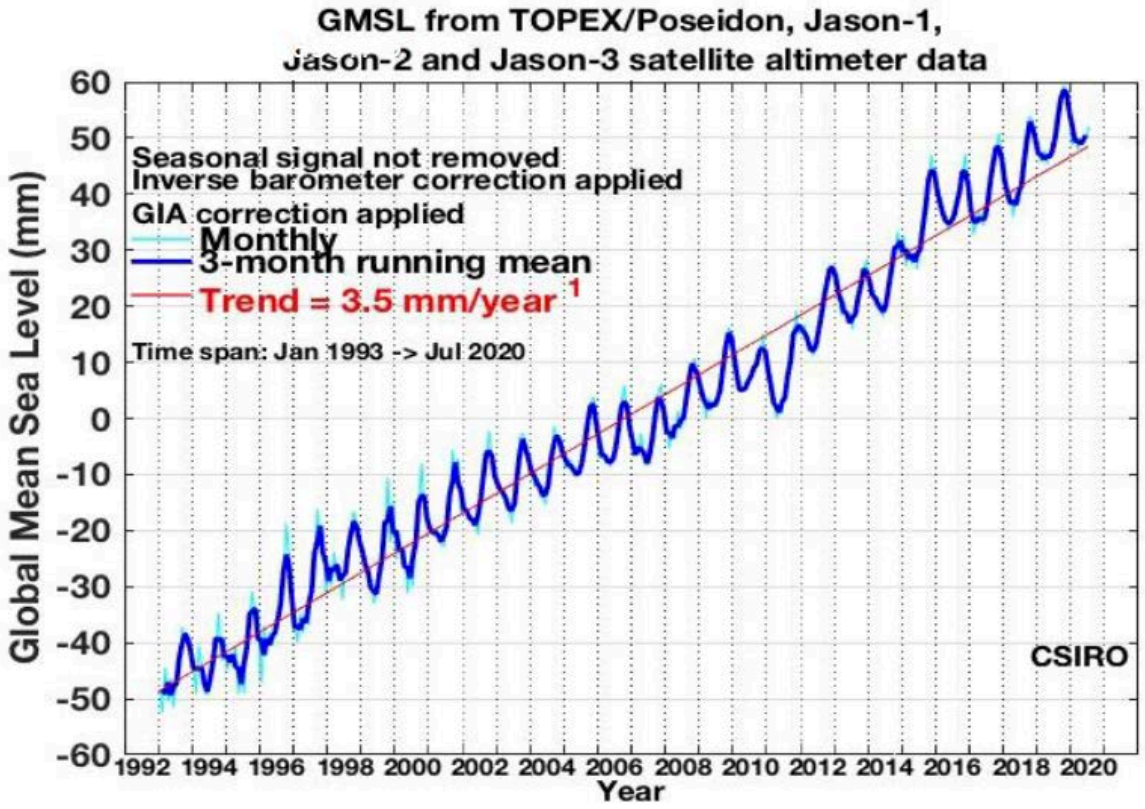



*Fig. 4.19* Monthly time series of global mean sea level measurements using Satellite Altimetry technique. Source image: CSIRO (n.d.).

Fig. 4.19 shows a positive trend (red line) of around $3.5$ mm/year, which in this case indicates sea level rise. This however needs to be further investigated and tested statistically (see Hypothesis testing for Sensing and Monitoring and also Modelling and Estimation).

Trend analysis expresses the changes of the variable of interest with respect to time $t$. Different types of trend are possible and for now we will mainly focus on linear trend, i.e. the time-dependent variable $Y(t)$ changes at a (constant) linear rate over time: $Y_t = y_0 + rt + \epsilon_t$. Other trends are however also possible, for example, quadratic, which includes $ct^2$, or log linear $\log(Y_t) = y_0 + rt + \epsilon_t$.

# Seasonality

Seasonal variations explain regular fluctuations in a certain period of time (e.g. a year), usually caused by climate and weather conditions (e.g. temperature, rainfall), cycles of seasons, customs, traditional habits, weekends, or holidays. For example, the weekly signal is usually evident in the volume of people engaged in shopping (likely more people prefer going shopping in the weekends)

From Fig. 4.19 it is also possible to see the seasonal variations: in fact sea levels are higher in summer and lower in winter. The annual warming/cooling cycle is the main contributor to

these seasonal variations.

Regular seasonal variations in a time series might be handled by using a sinusoidal model with one or more sinusoids with frequency that may be known or unknown depending on the context. In fig Fig. 4.19, cyclical behavior with a period of 1 year can be observed. A harmonic model for seasonal variation can be of the following two equivalent forms (using that $\cos(u+v) = \cos u \cos v - \sin u \sin v$):

$$\begin{aligned} Y(t) &= \sum_{k=1}^{p} A_k \cos(k\omega_0 t + \theta_k) + \epsilon_t \\ &= \sum_{k=1}^{p} (a_k \cos(k\omega_0 t) + b_k \sin(k\omega_0 t)) + \epsilon_t \end{aligned}$$

With the coefficients $a_k = A_k \cos\theta_k$ and $b_k = -A_k \sin\theta_k$, and where $\omega_0$ is the base (fundamental) frequency of the seasonal variation and is fixed or is determined by Spectral Analysis. To be more specific, we can use the Power Spectral Density (PSD) to determine the unknown frequencies.

Once $\omega_0$ is set, the coefficients $a_k$ and $b_k$ can be determined using the least-squares method, since the equation is linear in $a_k$ and $b_k$. From this the original sinusoids can be obtained using:

$$A_k = \sqrt{a_k^2 + b_k^2}, \qquad \theta_k = \arctan(-\frac{b_k}{a_k}), \qquad k = 1, \ldots, p$$

**Note**

This transformation is necessary to make the seasonal component phase-independent. Using regular estimation methods, we cannot linearly estimate the phase of the sinusoidal function. However by transforming the sinusoidal function into a linear combination of sine and cosine functions, we can estimate the phase of the seasonal component.

**Worked example - seasonality signal**

Show that the time series

$$Y(t) = A\cos(\omega_0 t + \theta)$$

with given $\omega_0$, can be rewritten as

$$Y(t) = a\cos(\omega_0 t) + b\sin(\omega_0 t)$$

and derive the formulation of $A$ and $\theta$.

Hint: you might need to know trigonometric identity
$\cos(u+v) = \cos(u)\cos(v) - \sin(u)\sin(v)$

**Solution**

Using the trigonometric identity to rewrite:

$$Y(t) = A\cos(\omega_0 t + \theta) = A(\cos(\omega_0 t)\cos(\theta) - \sin(\omega_0 t)\sin(\theta))$$

Retrieving the functions for a and b

$$a = A\cos(\theta) \qquad b = -A\sin(\theta)$$

Squaring both functions in order to get rid of the sin and cos

$$a^2 = A^2\cos^2(\theta) \qquad b^2 = A^2\sin^2(\theta)$$

Adding both functions together

$$a^2 + b^2 = A^2(\cos^2(\theta) + \sin^2(\theta))$$

Using this property to simplify:

$$\cos^2(\theta) + \sin^2(\theta) = 1$$

$$a^2 + b^2 = A^2$$

Take square root to find A

$$\sqrt{a^2 + b^2} = A$$

For $\theta$ we rewrite the second function

$$a = A\cos(\theta) \qquad -b = A\sin(\theta)$$

$$\frac{-b}{a} = \frac{\sin(\theta)}{\cos(\theta)} = \tan(\theta)$$

$$\theta = \arctan(\frac{-b}{a})$$

This video includes the solution to this exercise.

# Offset (jump)

Offsets are sudden changes or shifts in time series. There are different underlying reasons why we encounter offsets in time series.

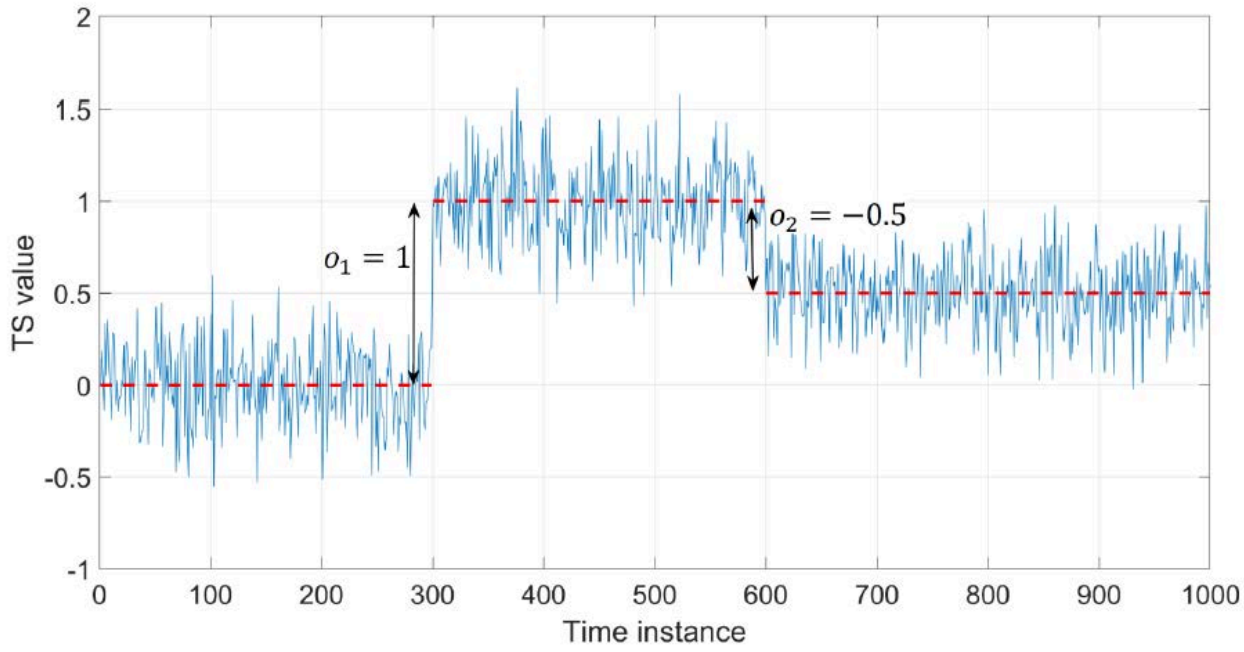



*Fig. 4.20* Example of time series with two offsets.

As a deterministic sudden change, offsets can be handled by a step function such as a heaviside step function with an epoch (time instant) that can be known or unknown (to be detected) depending on the time series.

In this case the time series is written as:

$$Y(t) = \sum_{k=1}^{q} o_k u_k(t) + \epsilon_t$$

where $q$ is the series of offsets (in Fig. 4.20 there are two offsets, hence $q = 2$) and each of them is expressed as a Heaviside step function

$$u_k(t) = \begin{cases} 0 & \text{if } \ t < t_k \\ 1 & \text{if } \ t \geq t_k \end{cases}$$

Once the time instant ($t_k$) of the offset is known, the amplitude can be estimated using least-squares.

# Noise

Noise simply refers to random fluctuations in the time series about its typical pattern. In general we can talk about white and colored noise in time series analysis. The following characteristics are associated with noise:

- Noise is not synonymous with error, although random variation, including measurement errors, contributes to noise. Essentially, noise represents the unpredictable fluctuations in data, while errors encompass any inaccuracies that may arise from a range of factors, including both random variations and systematic issues.
- It is required to filter out unwanted random variations, and detect meaningful information (i.e., a signal) from noise processes.
- Transforming data from the time domain to the frequency domain allows to filter out the frequencies that pollute the data.
- White noise can be decomposed into its constituent components (frequencies). In principle, white noise contains all wavelengths/colors (like white light), each contributing equally to the fluctuations observed in the data.
- Colored noise can seriously affect the analysis of time series, and their parameters of interest. Short-term colored noise has also predictive property (used for forecasting).

A purely stationary random process (or white noise process) yields a sequence of uncorrelated zero-mean random variables. This zero-mean random process is of the form

$$Y(t) = Y_t = \epsilon_t$$

where $\epsilon_t$ is the independent identically distributed (i.i.d.) error at epoch $t$. Therefore, the observation/noise at time $t$ is not dependent on any of the previous observations $Y_t$.

# Stochastic model

A stationary zero-mean random process has an expectation of zero (functional model), and a scaled identity matrix as its covariance matrix (stochastic model). The functional and stochastic models of white noise are of the form

$$\mathbb{E}(Y) = \mathbb{E}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

and

$$\mathbb{D}(Y) = \Sigma_Y = \sigma^2 \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

The noise can be represented with, for example, a Gaussian distribution with mean $\mu = 0$ and variance $\sigma^2$, that is $\epsilon(t) \sim \mathbf{N}(0, \sigma^2)$.

**Example - time series consisting of a trend, annual signal (seasonality), an offset and pure random noise (white noise)**

It can be written as

$$Y(t) = y_0 + rt + a\cos(\omega_0 t) + b\sin(\omega_0 t) + ou_k(t) + \epsilon(t)$$

where

- $y_0$ is the intercept (e.g. in mm)
- $r$ is the rate (e.g. in mm/year)
- $a$ and $b$ are the coefficients of the signal, (e.g. annual signal)
- $\omega_0$ is the frequency (e.g. 1 cycle/year)
- $o$ is the offset starting at time $t_k$
- $u_k(t)$ is the Heaviside step function
- $\epsilon(t)$ is the i.i.d. random Gaussian noise, i.e. $\epsilon(t) \sim \mathbf{N}(0, \sigma^2)$.

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Components of time series

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

**Introduction:**

The four components of time series, we will consider here, are the trend, seasonality, offset, and noise (white/colored). We use simulated data to show these components here. The observation equation of time series should have the following mathematical representation:

$$Y(t) = y_0 + rt + a\cos(\omega_0 t) + b\sin(\omega_0 t) + ou_k(t) + \epsilon(t) = y_0 + rt + A\cos(\omega_0 t + \phi_0)$$

where

- $y_0$: intercept (e.g. in mm)
- $r$: is the rate (e.g. in mm/day)
- $a$ and $b$ are the coefficients of the periodic signal
- $\omega_0$ is the frequency of signal (e.g. cycle/ day)
- $o$ is the size of the offset at time instant $t_k$
- $u_k(t)$ is the unit step function which is 1 if $t_k \leq t$ and 0 otherwise
- $\epsilon(t)$ is the random noise with a given variance which follows a Normal distribution: $\epsilon(t) \sim \mathbf{N}(0, \sigma^2)$

Here, we are assuming only a single seasonality and offset component. However, in many practical scenarios, there could be multiple components related to these.

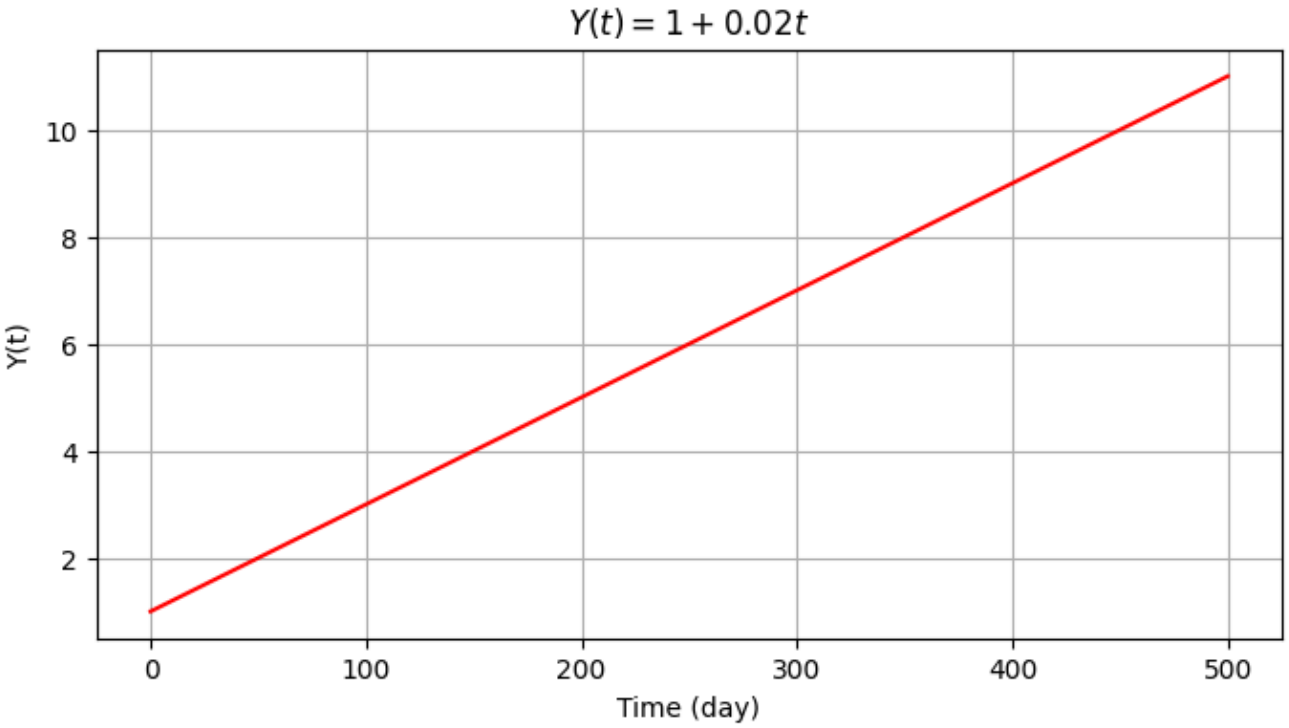
**Exercise:**

You can simulate your time series based on the priori information provided in the scripts. Plot your results and change the input variables to see the effect.

*The noise follows a normal distribution: use np.random.normal in order to draw random samples from a normal (Gaussian) distribution. Study more for this function here.*

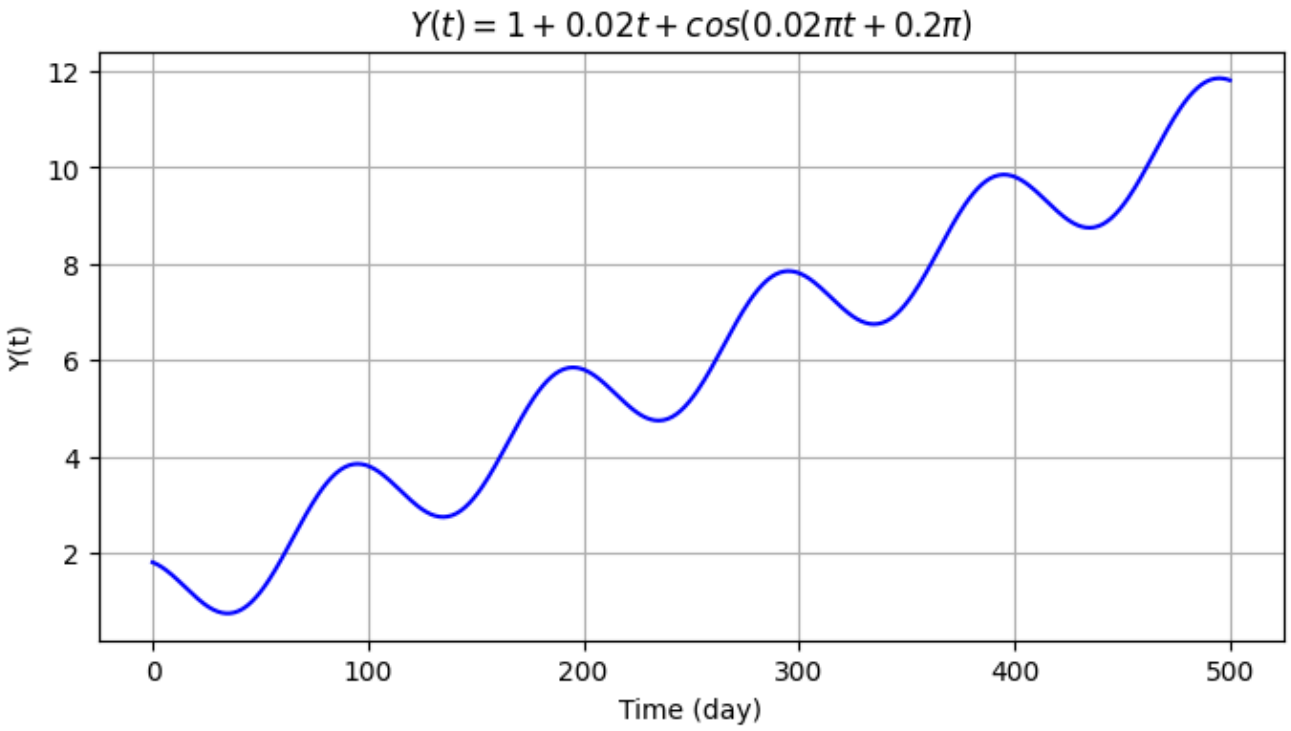
We will first create the time series $Y(t) = y_0 + rt$, with $y_0 = 1$ mm and $r = 0.02$ mm/day with a duration of 500 days (i.e., time series consists of 500 observations).

```
np.random.seed(0)  # For reproducibility

time = np.arange(501)
m = len(time)
y_0 = 1
r = 0.02
y1 = y_0 + r*time

plt.figure(figsize=(8,4))
plt.grid()
plt.plot(time, y1, color='red')
plt.ylabel('Y(t)')
plt.xlabel('Time (day)')
plt.title('$Y(t) = 1 + 0.02 t $');
```



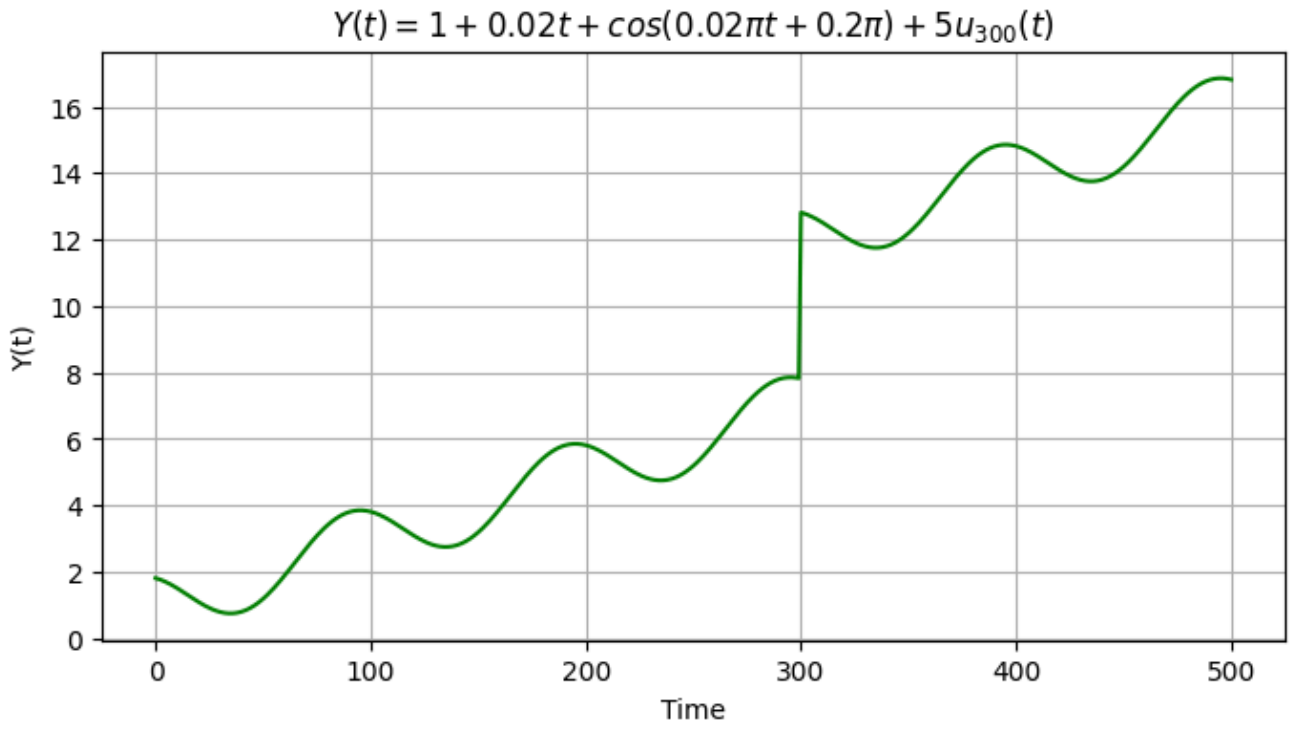

We then introduce seasonality to the data with a cosine signal $s(t) = A\cos(\omega_0 t + \phi_0)$ with $\omega_0 = 2\pi f$, frequency $f = 0.01$ cycle/day (i.e., 1 cycle per 100 days), amplitude $A = 1$ mm and initial phase $\phi_0 = 0.2\pi$ rad.

```
omega_0 = 2 * np.pi/100
A = 1
phi_0 = 0.2*np.pi
y2 = y1 + A*np.cos(omega_0 * time + phi_0)

plt.figure(figsize=(8,4))
plt.grid()
plt.plot(time, y2, color='blue')
plt.ylabel('Y(t)')
plt.xlabel('Time (day)')
plt.title('$Y(t) = 1 + 0.02 t + cos(0.02πt + 0.2π)$');
```



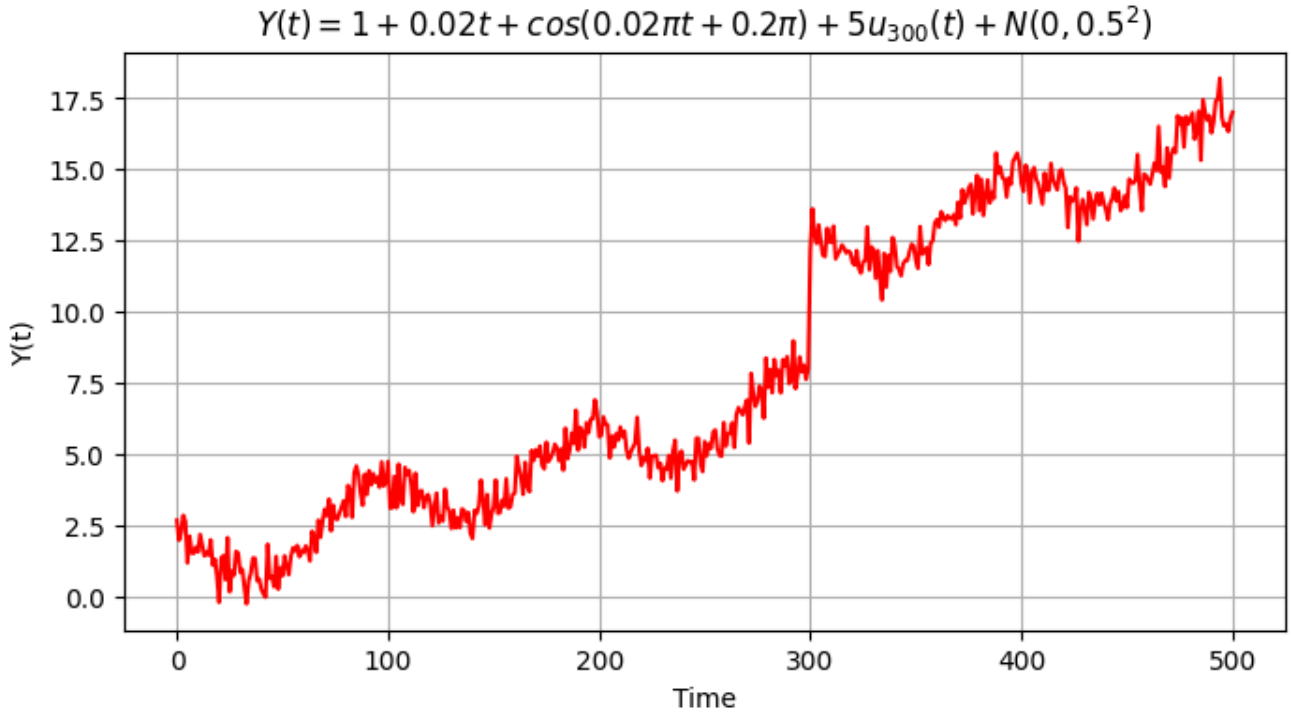

We now add an offset $o_k = 5$ at $t = 300$ days. We therefore create a copy of the previous signal and store it into a new array.

```
t_k = 300
O_k = 5
y3 = y2.copy()
y3[t_k:] = y3[t_k:] + O_k

plt.figure(figsize=(8,4))
plt.grid()
plt.plot(time, y3, color='g')
plt.ylabel('Y(t)')
plt.xlabel('Time')
plt.title('$Y(t) = 1 + 0.02 t + cos(0.02πt + 0.2π) + 5 u_{300}(t)$');
```




Eventually we include the random error $\epsilon \sim \mathbf{N}(\mu, \sigma_e)$ with $\mu = 0$ mm and $\sigma_e = 0.5$ mm.

```
mean = 0
sigma = 0.5
et = np.random.normal(loc = mean, scale = sigma, size = m)
y4 = y3 + et

plt.figure(figsize=(8,4))
plt.grid()
plt.plot(time, y4, color='red')
plt.ylabel('Y(t)')
plt.xlabel('Time')
plt.title('$Y(t) = 1 + 0.02 t + cos(0.02πt + 0.2π) + 5 u_{300}(t) + N(0,0.5^2)$');
```



> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Noise and stochastic model

## Contents


- Additional concepts
- Different types of noise


The code on this page can be used interactively: click 🚀 –> **Live Code** in the top right corner, then wait until the message **Python interaction ready!** appears.

In the previous section you have learned about the different components that can be present in a time series. Removing all these component, i.e. the functional model, we will be left with the residual term $\epsilon(t)$. In this section we will take a closer look at the difference between signal and noise, and introduce other types of noise than the traditional white noise.

## Additional concepts

In Signal Processing the data is just considered to be the signal of interest, whereas here we assume the data is "contaminated" with noise, i.e.

$$Y = \text{signal} + \text{noise}$$

Time series analysis means understanding patterns and, hence, extracting the **signal of interest** from the noisy data.

## Signal and noise

How can we describe both signal and noise?

- **Signal** - the meaningful information that we want to detect: deterministic characteristics by means of mathematical expressions to capture for example trend, seasonality and offsets.
- **Noise** - random and undesired fluctuation that interferes with the signal: stochastic process are needed to describe this. Parts of the time-correlated noise needs to be

accounted for in predictions, see later Autoregressive process.

The example in Fig. 4.21 shows that the *signal* can be described by $\cos(2\pi t f) + \sin(2\pi t f)$. The stochastic model (assuming independent normally distributed observations) would be a scaled identity matrix with variance equal to 1 (middle panel) and 9 (bottom panel), respectively. The signal of interest has been entirely hidden in the background noise in the bottom panel. Techniques from signal processing can be used to detect the frequency.

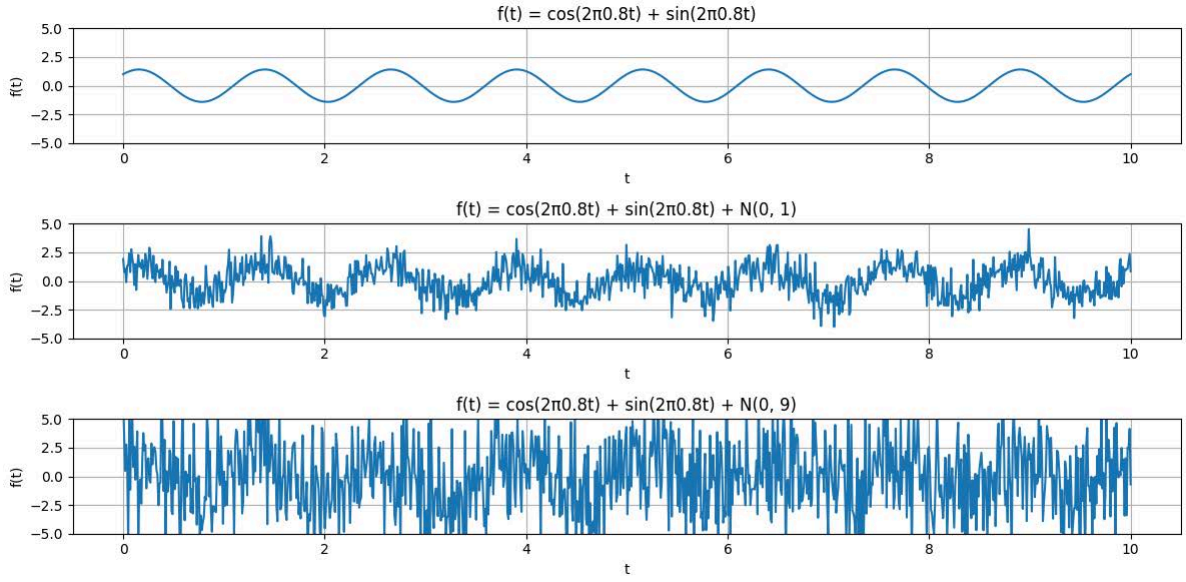



*Fig. 4.21* Example of a time series (top graph) affected by noise with different strengths (middle and bottom figures). Note the different scales on the vertical axes.

## Signal to noise ratio

In signal processing the signal to noise ratio is commonly used to report on the amount of noise present in the model. If we analyze the model $Y = signal + noise$, then Y is a random variable with $E[Y] = E[signal] = \mu$, and its variance $D(Y) = D(noise) = \sigma^2$. Using this the signal to noise ratio is often defined as:

$$SNR = \frac{\mu}{\sigma}$$

The signal to noise ratio is a measure of how much the signal stands out from the noise. The higher the signal to noise ratio, the more the signal stands out from the noise. Better equipment or more data can increase the signal to noise ratio.

# Different types of noise

In the ideal case, when the signal is removed, you are left with white noise. A white noise stochastic model has the following properties:

$$\mathbb{E}(Y) = \mathbb{E}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

and

$$\mathbb{D}(Y) = \Sigma_Y = \sigma^2 \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

Most notable, all observations are uncorrelated (off-diagonal elements of the covariance matrix are equal to 0). When we compute the PSD, the resulting density will be flat over the entire range of frequencies. In other words, a white noise process has equal energy over all frequencies, just like white light. We will show this in the interactive plot at the bottom of this page.

# Colored noise

In time series it is not guarantied that the individual observations are uncorrelated. At the bottom of this page you will find an interactive plot. You can select four different types of noise: white, pink, red and blue. The noise processes are plotted in combination with the PSD. The PSD is a measure of the power of the signal at different frequencies. The white noise process has a flat PSD, while the other noise processes have a different shape. The pink noise process has a PSD that decreases with frequency, the red noise process has a PSD that decreases quadratically with frequency, and the blue noise process has a PSD that increases with frequency.

**Note**

If you are interested, you can read more about the different types of noise in the Wikipedia article. In here you can also listen to the different types of noise, which might give you a better understanding of the differences.

**Attribution**

This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Modelling and Estimation

## Contents


- Components of time series
- Best Linear Unbiased Estimation (BLUE)
- Model identification


The goal is now to:

- estimate parameters of interest (i.e., components of time series) using **Best Linear Unbiased Estimation (BLUE)**;
- evaluate the confidence intervals of the estimators for the parameters of interest;

## Components of time series

As already discussed, we will distinguish the following components in a time series:

- **Trend:** General behavior and variation of the process. This often is a linear trend with an unknown intercept $y_0$ and a rate $r$.
- **Seasonality:** Regular seasonal variations, which can be expressed as sine functions with (un)known angular frequency $\omega_0$, and unknown amplitude $A$ and phase $\theta$.
- **Offset:** A jump of size $o$ in a time series starting at epoch $t_k$.
- **Noise:** White or colored noise (e.g., AR process).

## Best Linear Unbiased Estimation (BLUE)

If the components of time series are known, we may use the linear model of observation equations to estimate those components. We will use the theory from the chapter on observation theory to estimate the parameters of interest.

Consider the linear model of observation equations as

$$Y = \mathrm{A}\mathrm{x} + \epsilon, \;\; \mathbb{D}(Y) = \Sigma_Y$$

Recall that the BLUE of $\mathrm{x}$ is:

$$\hat{X} = (\mathrm{A}^T \Sigma_Y^{-1} \mathrm{A})^{-1} \mathrm{A}^T \Sigma_Y^{-1} Y, \;\; \Sigma_{\hat{X}} = (\mathrm{A}^T \Sigma_Y^{-1} \mathrm{A})^{-1}$$

The BLUE of $Y$ is:

$$\hat{Y} = \mathrm{A}\hat{X}, \;\; \Sigma_{\hat{Y}} = \mathrm{A}\Sigma_{\hat{X}}\mathrm{A}^T$$

and $\epsilon$ is:

$$\hat{\epsilon} = Y - \hat{Y}, \;\; \Sigma_{\hat{\epsilon}} = \Sigma_Y - \Sigma_{\hat{Y}}$$

# Model of observation equations

The linear model, consisting of the above three components plus noise, is of the form

$$Y_t = y_0 + rt + a\cos\omega_0 t + b\sin\omega_0 t + ou_k(t) + \epsilon_t$$

The linear model should indeed be written for all time instances $t_1, \ldots, t_m$, resulting in $m$ equations as:

$$\begin{aligned}
Y(t_1) &= y_0 + rt_1 + a\cos\omega_0 t_1 + b\sin\omega_0 t_1 + ou_k(t_1) + \epsilon(t_1) \\
Y(t_2) &= y_0 + rt_2 + a\cos\omega_0 t_2 + b\sin\omega_0 t_2 + ou_k(t_2) + \epsilon(t_2) \\
&\vdots \\
Y(t_k) &= y_0 + rt_k + a\cos\omega_0 t_k + b\sin\omega_0 t_k + ou_k(t_k) + \epsilon(t_k) \\
&\vdots \\
Y(t_m) &= y_0 + rt_m + a\cos\omega_0 t_m + b\sin\omega_0 t_m + ou_k(t_m) + \epsilon(t_m)
\end{aligned}$$

These equations can be written in a compact matrix notation as

$$Y = \mathrm{A}\mathrm{x} + \epsilon$$

where

$$\overbrace{\begin{bmatrix} Y_1 \\ \vdots \\ Y_{k-1} \\ Y_k \\ \vdots \\ Y_m \end{bmatrix}}^{Y} = \overbrace{\begin{bmatrix} 1 & t_1 & \cos\omega_0 t_1 & \sin\omega_0 t_1 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{k-1} & \cos\omega_0 t_{k-1} & \sin\omega_0 t_{k-1} & 0 \\ 1 & t_k & \cos\omega_0 t_k & \sin\omega_0 t_k & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_m & \cos\omega_0 t_m & \sin\omega_0 t_m & 1 \end{bmatrix}}^{\mathrm{A}} \overbrace{\begin{bmatrix} y_0 \\ r \\ a \\ b \\ o \end{bmatrix}}^{\mathrm{x}} + \overbrace{\begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_{k-1} \\ \epsilon_k \\ \vdots \\ \epsilon_m \end{bmatrix}}^{\epsilon}$$

with the $m \times m$ covariance matrix

$$\Sigma_Y = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \dots & \sigma_{1m} \\ \sigma_{21} & \sigma_2^2 & & \\ \vdots & \vdots & \ddots & \\ \sigma_{m1} & \sigma_{m2} & \dots & \sigma_m^2 \end{bmatrix}$$

**Exercise**

A time series exhibits a linear regression model $Y(t) = y_0 + rt + \epsilon(t)$. The measurements have also been taken at a measurement frequency of 10 Hz, producing epochs of $t = 0.1, 0.2, \dots, 100$ seconds, so $m = 1000$. Later an offset was also detected at epoch 260 using statistical hypothesis testing. For the linear model $Y = \mathrm{A}\mathrm{x} + \epsilon$, establish an appropriate design matrix that can capture all the above effects.

**Solution**

In the linear regression case, the design matrix consists of two columns, one for the unknown $y_0$ (a column on ones), and the other for $r$ (a column of time, $t$). Due to the presence of an offset, the mathematical model should be modified to:

$$Y(t) = y_0 + rt + o_k u_k(t) + \epsilon(t)$$

where $u_k(t)$ is the Heaviside step function:

$$u_k(t) = \begin{cases} 0, & \text{if } t < t_k = 26 \\ 1, & \text{if } t \geq t_k = 26 \end{cases}$$

and $o_k$ is the magnitude of the offset. This means that we have to add a third column to the design matrix having zeros before the epoch 260, and ones afterwards. Since epoch 260 corresponds to $t = 26$ s (noting that we have 10 Hz data), in $\mathrm{A}$ we begin to have 1's in that row:

$$\mathrm{A} = \begin{bmatrix} 1 & t_1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & t_{259} & 1 \\ 1 & t_{260} & 1 \\ \vdots & \vdots & \vdots \\ 1 & t_{1000} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & 25.9 & 0 \\ 1 & 26 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 100 & 1 \end{bmatrix}$$

# Estimation of parameters

If we assume the covariance matrix, $\Sigma_Y$, is known, we can estimate $\mathrm{x}$ using BLUE:

$$\hat{X} = \begin{bmatrix} \hat{y_0} \\ \hat{r} \\ \hat{a} \\ \hat{b} \\ \hat{o} \end{bmatrix}, \quad \Sigma_{\hat{X}} = \begin{bmatrix} \sigma^2_{\hat{y}_0} & \sigma_{\hat{y}_0\hat{r}} & \sigma_{\hat{y}_0\hat{a}} & \sigma_{\hat{y}_0\hat{b}} & \sigma_{\hat{y}_0\hat{o}} \\ \sigma_{\hat{r}\hat{y}_0} & \sigma^2_{\hat{r}} & \sigma_{\hat{r}\hat{a}} & \sigma_{\hat{r}\hat{b}} & \sigma_{\hat{r}\hat{o}} \\ \sigma_{\hat{a}\hat{y}_0} & \sigma_{\hat{a}\hat{r}} & \sigma^2_{\hat{a}} & \sigma_{\hat{a}\hat{b}} & \sigma_{\hat{a}\hat{o}} \\ \sigma_{\hat{b}\hat{y}_0} & \sigma_{\hat{b}\hat{r}} & \sigma_{\hat{b}\hat{a}} & \sigma^2_{\hat{b}} & \sigma_{\hat{b}\hat{o}} \\ \sigma_{\hat{o}\hat{y}_0} & \sigma_{\hat{o}\hat{r}} & \sigma_{\hat{o}\hat{a}} & \sigma_{\hat{o}\hat{b}} & \sigma^2_{\hat{o}} \end{bmatrix}$$

Given $\hat{X}$ and $\Sigma_{\hat{X}}$, we can obtain the confidence region for the parameters. For example, assuming the observations are normally distributed, a 99% **confidence interval** for the rate $r$ is ($\alpha = 0.01$):

$$\hat{r} \pm k\sigma_{\hat{r}}$$

where $\sigma_{\hat{r}} = \sqrt{(\Sigma_{\hat{X}})_{22}}$ is the standard deviation of $\hat{r}$ and $k = 2.58$ is the critical value obtained from the standard normal distribution (using $0.5\alpha$).

# Model identification

The design matrix $\mathrm{A}$ is usually assumed to be known. So far, we have assumed the frequency $\omega_0$ of the periodic pattern (seasonality, for example) in a $a\cos\omega_0 t + b\sin\omega_0 t$ is known, so the design matrix $\mathrm{A}$ can be directly obtained. In some applications, however, such information is hidden in the data, and needs to be identified/detected. Linear model identification is a way to reach this goal.

***How to determine $\omega_0$ if it is unknown a priori?***

# Example power spectral density

Fig. 4.22 shows on the left the original time series as well as the estimated linear trend and seasonal signal. The sine wave has a period ($T = 1/f$) of 100. Indeed the PSD as function of period on the right shows a peak at a period of 100.

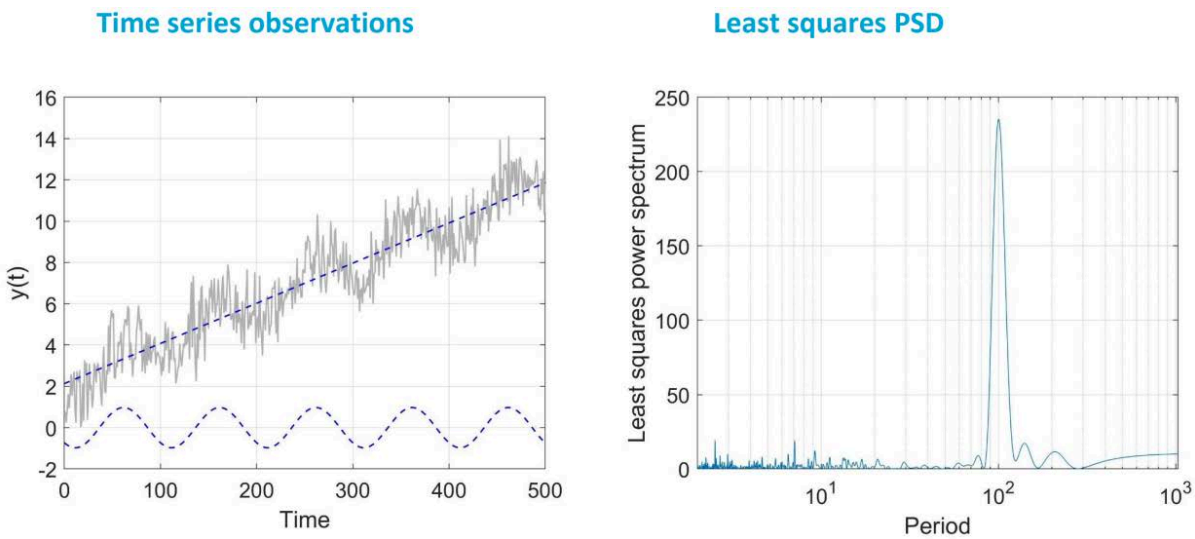



*Fig. 4.22* Left: time series (grey) and estimated linear trend and sine wave with period of 100. Right: estimated PSD.

This means we can estimate the frequency $\omega_0$ of the periodic pattern using the techniques discussed in the chapter on signal processing. Once we have the frequency, we can construct the design matrix $\mathrm{A}$.

It is also possible to infer the frequency of the periodic pattern by reasoning. For example, if we know our model depends on temperature, we can assume that the frequency of the seasonal pattern is related to the temperature cycle (e.g., 24 hours). However, this is a more qualitative approach and should be used with caution. Best practice is to use the DFT or PSD to estimate the frequency.

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Time series modelling

We have a time series with a clear trend, seasonality and jump. The researcher knows that there is jump present at $t = 270$ due to a change in the data collection method. First we will load the data and plot it. After that, we will use the PSD to identify the frequency of the seasonality and the trend.

```
# Plot the data
plt.figure(figsize=(12, 3))
plt.plot(t, data, label="y_values")
plt.xlabel("Time")
plt.ylabel("y_values")
plt.title("y_values over time")
plt.grid()
```

run | run all | add cell | clear

We can now inspect the time series and identify the jump at $t = 270$. We can also see that there is a clear trend and seasonality. We will now use the PSD to identify the frequency of the seasonality and the trend.

remember that to compute a PSD we need to make our time series stationary. In the cell below I have shown how not to do it. without stationarity, the PSD is not useful.

```
1 plt.psd(data, Fs=1);
```

run | run all | add cell | clear

Since we already now there is a linear trend and a jump we can remove them from the time series. We can do this by fitting a linear model to the time series and subtracting it. The resulting residuals will be stationary and we can compute the PSD.

Matrix A is defined as follows:

$$A = \begin{bmatrix} 1 & t_1 & 0 \\ \vdots & \vdots & \vdots \\ 1 & t_{269} & 0 \\ 1 & t_{270} & 1 \\ 1 & t_{271} & 1 \\ \vdots & \vdots & \vdots \\ 1 & t_{500} & 1 \end{bmatrix}$$

```
offset = np.zeros_like(data)
offset[270:] = 1

A = np.column_stack((np.ones_like(t), t, offset))
xhat = np.linalg.solve(A.T @ A, A.T @ data)
yhat = A @ xhat
ehat = data - yhat

fig, axs = plt.subplots(2, 1, figsize=(12, 5))

# Plot data and fit
axs[0].plot(t, data, label='Data')
axs[0].plot(t, yhat, label='Fit', linestyle='--')
axs[0].set_xlabel('Time')
axs[0].set_ylabel('y values')
```

run | run all | add cell | clear

Using peak detection we can identify the frequency of the seasonality and the trend.

Finally, we can use the identified frequencies to model the time series.

For this, we need to expand our matrix A to include the seasonality and the trend. The new matrix A is defined as follows:

$$A = \begin{bmatrix} 1 & t_1 & 0 & \cos(2\pi t_1 f) & \sin(2\pi t_1 f) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{269} & 0 & \cos(2\pi t_{269} f) & \sin(2\pi t_{269} f) \\ 1 & t_{270} & 1 & \cos(2\pi t_{270_f}) & \sin(2\pi t_{270} f) \\ 1 & t_{271} & 1 & \cos(2\pi t_{271_f}) & \sin(2\pi t_{271} f) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_{500} & 1 & \cos(2\pi t_{500_f}) & \sin(2\pi t_{500} f) \end{bmatrix}$$

```
f = highest_peak_freq

A2 = np.column_stack((np.ones_like(t), t, offset, np.cos(2*np.pi*f*t)

xhat2 = np.linalg.solve(A2.T @ A2, A2.T @ data)
yhat2 = A2 @ xhat2
ehat2 = data - yhat2 # removing the functional components from the da

fig, axs = plt.subplots(2, 1, figsize=(12, 5))

# Plot data and fit
axs[0].plot(t, data, label='Data')
axs[0].plot(t, yhat2, label='Fit', linestyle='--')
axs[0].set_xlabel('Time')
```

run | run all | add cell | clear

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Time Series Stationarity

## Contents


- Why stationary time series?
- How to “stationarize” a time series?
- … and then what?


> **Definition**
>
> A stationary time series $S(t)$ is a stochastic process whose statistical properties do not depend on the time at which it is observed.

This means that parameters such as *mean* and *(co)variance* should remain constant over time and not follow any trend, seasonality or irregularity.

- Mean of the process is time-independent

$$\mathbb{E}(S(t)) = \mathbb{E}(S_t) = \mu$$

- Covariance of the process is independent of $t$ for each time shift $\tau$ (so only a function of τ and not t):

$$Cov(S_t, S_{t-\tau}) = Cov(S_t, S_{t+\tau}) = \mathbb{E}((S_t - \mu)(S_{t-\tau} - \mu)) = c_\tau$$

- The variance (i.e., $\tau = 0$) is then also constant with respect to time :

$$Var(S_t) = \mathbb{E}((S_t - \mu)^2) = c_0 = \sigma^2$$

Notice that we have introduced the new notation $S_t$ to denote a stationary time series. The time series $Y_t$ is then a non-stationary time series.

# Why stationary time series?

Stationarity is important if we want to use a time series for forecasting (predicting future behaviour), which is not possible if the statistical properties change over time.

In practice, we may in fact be interested in for instance the trend and seasonality of a time series. Also, many real-world time series are of course non-stationary. Therefore the approach is to first “stationarize” the time series (e.g., remove the trend), use this stationary time series to predict future states based on the statistical properties (stochastic process), and then apply a back-transformation to account for the non-stationarity (e.g., add back the trend).

# How to “stationarize” a time series?

There are several ways to make a time series stationary. In this course we will focus on detrending the data using least-squares fit.

## Least-squares fit

If we can express the time series $Y = [Y_1, \ldots, Y_m]^T$ with a linear model of observation equations as $Y = \mathrm{A}\mathrm{x} + \epsilon$, we can apply best linear unbiased estimation to estimate the parameters $\mathrm{x}$ that describe e.g. the trend and seasonality:

$$\hat{X} = (\mathrm{A}^T \Sigma_Y^{-1} \mathrm{A})^{-1} \mathrm{A}^T \Sigma_Y^{-1} Y$$

A detrended time series is obtained in the form of the residuals

$$\hat{\epsilon} = Y - \mathrm{A}\hat{X}$$

The **detrended** $\hat{\epsilon}$ **is assumed to be stationary** for further **stochastic processing**. This is also an admissible transformation because $Y$ can uniquely be reconstructed as $Y = \mathrm{A}\hat{X} + \hat{\epsilon}$.

Let us take a look into an example:

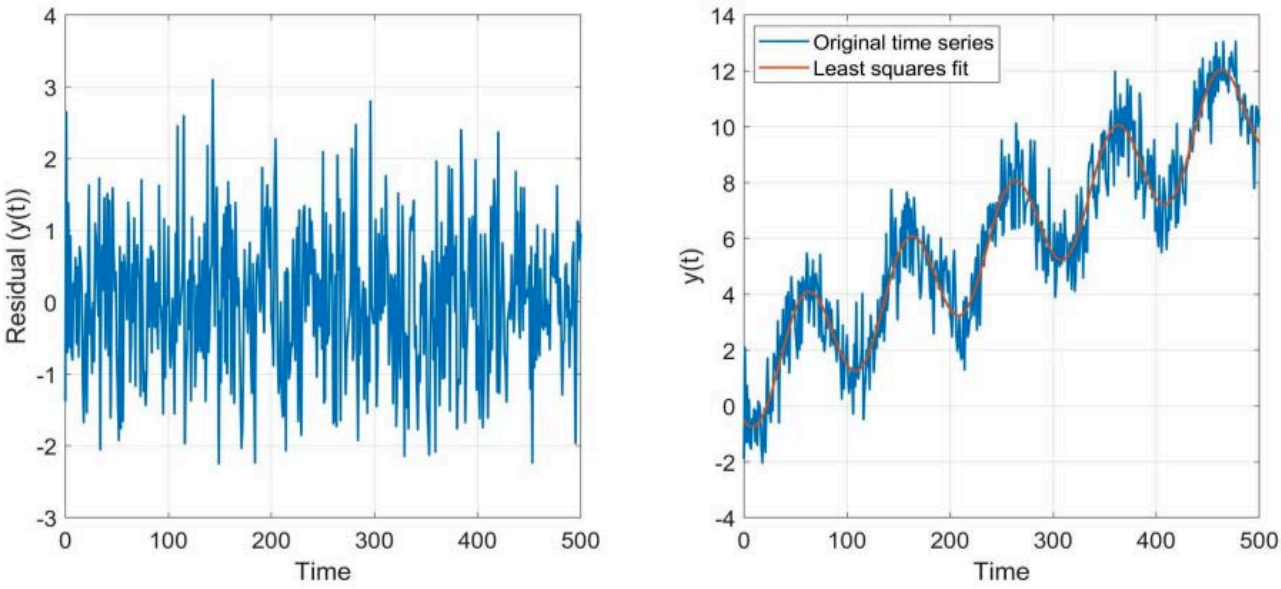



*Fig. 4.23* Example of a time series (right graph) with linear and seasonal trend. The residuals (= stationary time series) after applying BLUE are shown on the left.

In the example above, for each observation $Y_i = y_0 + rt_i + a\cos\omega_0 t_i + b\sin\omega_0 t_i + \epsilon_i$, where $a$ and $b$ describe the seasonality and $y_0$ and $r$ the trend. The time series then is:

$$\begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_m \end{bmatrix} = \begin{bmatrix} 1 & t_1 & \cos\omega_0 t_1 & \sin\omega_0 t_1 \\ 1 & t_2 & \cos\omega_0 t_2 & \sin\omega_0 t_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & t_m & \cos\omega_0 t_m & \sin\omega_0 t_m \end{bmatrix} \begin{bmatrix} y_0 \\ r \\ a \\ b \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{bmatrix}$$

The time series of residuals (left panel) is indeed a stationary time series.

**Question Stationary Time Series**

Which of the four options is a stationary time series?

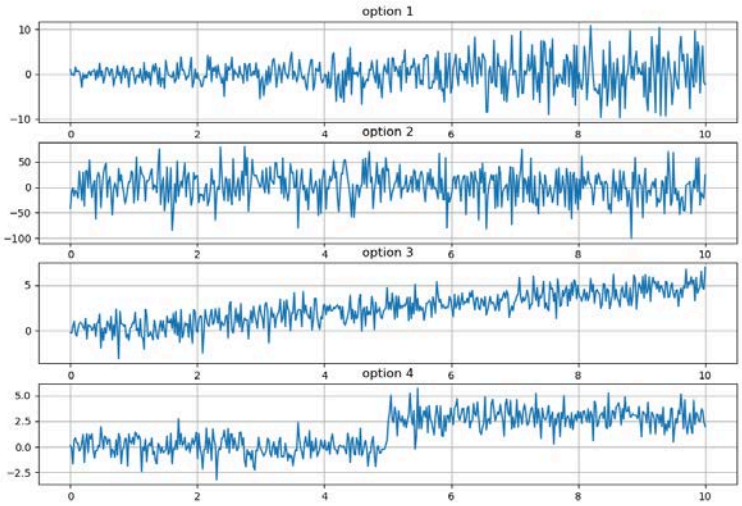



*Fig. 4.24* Example of a stationary time series.

**Solution**

The time series in the second panel is stationary. The mean and variance are constant over time.

# Other ways to make a time series stationary

When model specification is not straightforward, other methods can be used to make a time series stationary. Two common methods are single differencing and moving average. Single differencing of $Y = [Y_1, \ldots, Y_m]^T$ makes a time series $\Delta Y_t = Y_t - Y_{t-1}$. Another way to create an (almost) stationary time series is by taking the moving average of the time series. Where we apply a moving average of $k$ observations to the time series $Y$ to create a new time series $\bar{Y}_t = \frac{1}{k} \sum_{i=1}^{k} Y_{t-i}$, and then take the difference between the original time series and the moving average to obtain a (nearly) stationary time series $\Delta Y_t = Y_t - \bar{Y}_t$.

Both these methods do not require a model specification. So in cases where the model is not known, these methods can be used to make the time series stationary.

# ... and then what?

We have seen different ways of obtaining a stationary time series from the original time series. The reason is that in order to make predictions (forecasting future values, beyond the

time of the last observation in the time series) we need to account for both the **signal-of-interest** and the **noise**. Estimating the signal-of-interest was covered in the previous section. In the next sections we will show how the noise can be modelled as a stochastic process. Given a time series $Y = \mathrm{A}\mathrm{x} + \epsilon$, the workflow is as follows:

1. Estimate the signal-of-interest $\hat{X} = (\mathrm{A}^T\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^T\Sigma_Y^{-1}Y$ (Section Modelling and estimation).
2. Model the noise using the Autoregressive (AR) model, using the stationary time series $S := \hat{\epsilon} = Y - \mathrm{A}\hat{X}$ (Section AR).
3. Predict the signal-of-interest: $\hat{Y}_{signal} = \mathrm{A}_p\hat{X}$, where $\mathrm{A}_p$ is the design matrix describing the functional relationship between the future values $Y_p$ and $\mathrm{x}$ (Section Forecasting).
4. Predict the noise $\hat{\epsilon}_p$ based on the AR model.
5. Predict future values of the time series: $\hat{Y}_p = \mathrm{A}_p\hat{X} + \hat{\epsilon}_p$ (Section Forecasting).

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Autocovariance function

## Contents


- Autocovariance and autocorrelation
- Normalized ACF


Before we can look into the modelling of a stochastic process using an Autoregressive (AR) model, we first need to introduce the autocovariance function (ACF) for a stationary time series, and describe the relationship between ACF and a power spectral density (PSD).

As in Observation theory, the variance component is often determined based on the precision of an observation (at a given epoch), and the covariance components quantitatively indicate the statistical dependence (or independence) between observations. In this case, dependence is inherently introduced by the physical processes that produce the signal (of which our time series is a sample), and in fact our time series methods seek to (mathematically) account for this.

## Autocovariance and autocorrelation

Let us assume an arbitrary (discrete) stationary time series, $S = [S_1, S_2, \ldots, S_m]^T$, with mean $\mathbb{E}(S) = \mu$ and variance $Var(S_i) = \sigma^2$. Remember that stationarity implies that the statistical properties of the time series do not depend on the time at which it is observed, i.e. expectation and variance are constant over time.

The *formal* (or: theoretical) autocovariance is defined as

$$Cov(S_{t+\tau}, S_t) = \mathbb{E}(S_{t+\tau}S_t) - \mu^2 = c_\tau$$

We have that $Cov(S_{t+\tau}, S_t) = Cov(S_t, S_{t-\tau})$.

**Exercise covariance**

Show that the covariance can be written as:

$$Cov(S_{t+\tau}, S_t) = \mathbb{E}(S_{t+\tau}S_t) - \mu^2 = c_\tau$$

**Solution**

$$\begin{aligned} Cov(S_{t+\tau}, S_t) &= \mathbb{E}[(S_{t+\tau} - \mathbb{E}(S_{t+\tau}))(S_t - \mathbb{E}(S_t))] \\ &= \mathbb{E}((S_{t+\tau} - \mu)(S_t - \mu)) \\ &= \mathbb{E}(S_{t+\tau}S_t - \mu S_t - \mu S_{t+\tau} + \mu^2) \\ &= \mathbb{E}(S_{t+\tau}S_t) - \mu\mathbb{E}(S_t) - \mu\mathbb{E}(S_{t+\tau}) + \mu^2 \\ &= \mathbb{E}(S_{t+\tau}S_t) - 2\mu^2 + \mu^2 \\ &= \mathbb{E}(S_{t+\tau}S_t) - \mu^2 \end{aligned}$$

**Exercise covariance**

Prove that $Cov(S_t, S_{t-\tau}) = Cov(S_{t+\tau}, S_t)$:

**Solution**

From the definition of covariance, we know that

$$Cov(a, b) = Cov(b, a)$$

Hence, we have that

$$Cov(S_t, S_{t-\tau}) = Cov(S_{t-\tau}, S_t)$$

Due to the stationarity of the time series, we have that

$$Cov(S_t, S_{t-\tau}) = Cov(S_{t+\tau}, S_t)$$

Therefore, we have that

$$Cov(S_t, S_{t-\tau}) = Cov(S_{t+\tau}, S_t)$$

The *formal* autocorrelation is defined as

$$r_\tau = \mathbb{E}(S_{t+\tau} S_t)$$

> **Note**
>
> When we have a zero-mean time series, $\mu = 0$, it follows that $c_\tau = r_\tau$

# Empirical autocovariance

The autocovariance function of a time series is not known beforehand, and hence needs to be estimated based on the actual observed values. The least-squares method or maximum likelihood method can be used to estimate this *empirical* autocovariance function of a time series. Let us see how!

### Least-squares estimation

For a given stationary time series $S = [S_1, S_2, \ldots, S_m]^T$, the least-squares estimator of the **autocovariance function** is given by

$$\hat{C}_\tau = \frac{1}{m-\tau} \sum_{i=1}^{m-\tau} (S_{i+\tau} - \mu)(S_i - \mu), \qquad \tau = 0, 1, \ldots, m-1$$

The least-squares estimator of **autocorrelation** (also called empirical autocorrelation function) is then

$$\hat{R}_\tau = \frac{1}{m-\tau} \sum_{i=1}^{m-\tau} S_{i+\tau} S_i, \qquad \tau = 0, 1, \ldots, m-1$$

### Maximum likelihood estimations

The maximum likelihood estimator of **autocovariance** is given by

$$\hat{C}_\tau = \frac{1}{m} \sum_{i=1}^{m-\tau} (S_{i+\tau} - \mu)(S_i - \mu), \qquad \tau = 0, 1, \ldots, m-1$$

Note that this is a biased estimator, $\mathbb{E}(\hat{C}_\tau) \neq c_\tau$.

Similarly, the maximum likelihood estimate of the autocorrelation follows as:

$$\hat{R}_\tau = \frac{1}{m}\sum_{i=1}^{m-\tau} S_{i+\tau}S_i, \qquad \tau = 0, 1, \ldots, m-1$$

> **Note**
>
> Here we use capitals for $\hat{C}_\tau$ and $\hat{R}_\tau$ since **estimators** are always a function of the random observables $S_t$.

# Covariance matrix based on autocovariance

The structure of a covariance matrix for a stationary time series is purely symmetric and it looks like

$$\Sigma_S = \begin{bmatrix} \sigma^2 & c_1 & c_2 & \cdots & c_{m-1} \\ c_1 & \sigma^2 & c_1 & \ddots & \vdots \\ c_2 & c_1 & \sigma^2 & \ddots & c_2 \\ \vdots & \ddots & \ddots & \ddots & c_1 \\ c_{m-1} & & c_2 & c_1 & \sigma^2 \end{bmatrix}$$

There are $m$ (co)variance components - **one** variance component, $\sigma^2 = c_0$, and $m-1$ covariance components, $c_i$.

# Normalized ACF

The least-squares estimator of the autocovariance function (ACF) has some important properties (derivations are outside the scope of MUDE).

The empirical autocovariance function is an unbiased estimator of the formal autocovariance function

$$\mathbb{E}(\hat{C}_\tau) = c_\tau$$

The *normalized* autocovariance estimator can directly be obtained from the autocovariance estimator as

$$\hat{\rho}_\tau = \frac{\hat{C}_\tau}{\hat{C}_0}, \quad \tau = 0, \ldots, m-1 \implies \hat{\rho}_0 = 1$$

> **Note**
>
> The estimated normalized autocovariance is the same as the time dependent Pearson correlation coefficient.
>
> In literature $\hat{\rho}_\tau$ is often referred to as the *autocorrelation function*.

The variance of the normalized ACF can be approximated as

$$\sigma^2_{\hat{\rho}_\tau} = \frac{1}{m-\tau} + \frac{2\hat{\rho}^2_\tau}{m}$$

If $m$ is sufficiently large, $\hat{\rho}_\tau$ is normally distributed:

$$\hat{\rho}_\tau \sim N(\rho_\tau, \sigma^2_{\hat{\rho}_\tau})$$

# Worked example

Let us consider a time series of $m = 100$ observations, such that

$$\hat{\rho}_1 = 0.4$$

We know that

$$\hat{\rho}_1 \sim N(\rho_1, \sigma^2_{\hat{\rho}_1})$$

with $\rho_1$ unknown and

$$\sigma^2_{\hat{\rho}_1} = \frac{1}{m-1} + \frac{2\hat{\rho}^2_1}{m} = 0.0133 \implies \sigma_{\hat{\rho}_1} = 0.1153$$

We will now apply a test whether the estimated autocorrelation is significant. The null hypothesis assumes there is **no** correlation:

- $\mathcal{H}_0$: $\rho_1 = 0$
- $\mathcal{H}_a$: $\rho_1 \neq 0$

Since we know the distribution of $\hat{\rho}_1$, a suitable test statistic would be:

$$T = \frac{\hat{\rho}_1}{\sigma_{\hat{\rho}_1}} \sim N(0,1)$$

where we would reject $\mathcal{H}_0$ if $|T| > k_\alpha$. With a false alarm rate of $\alpha = 0.01$, we find that the critical value can be obtained from the table of the standard normal distribution. Note that we have a 2-sided critical region, hence we need to look up the value for $0.5\alpha$.

In this example, we obtain:

$$(T = 3.47) > (k_\alpha = 2.58)$$

and hence the null hypothesis is rejected, implying that the autocorrelation is significant.

**Exercise**

A zero-mean stationary noise process consists of $m = 5$ observations:

$$S = \begin{bmatrix} 2 & 1 & 0 & -1 & -2 \end{bmatrix}^T$$

What is the *least-squares estimate* of the normalized ACF at $\tau = 1$; so compute $\hat{\rho}_1$?

**Solution**

The normalized autocovariance function (ACF) can be estimated from the auto-covariance function as:

$$\hat{\rho}_\tau = \frac{\hat{C}_\tau}{\hat{C}_0}, \qquad \tau = 0, \ldots, m-1$$

where the least-squares estimate of auto-covariance function is:

$$\hat{C}_\tau = \frac{\sum_{i=1}^{m-\tau}(S_i - \mu)(S_{i+\tau} - \mu)}{m - \tau}, \qquad \tau = 0, \ldots, m-1$$

For our application we have $\mu_y = 0$, as we deal with a zero-mean process. We have to compute $\hat{\sigma}(0)$ and $\hat{\sigma}(1)$ given as:

$$\hat{C}_0 = \frac{\sum_{i=1}^{m} S_i^2}{m - 0} = \frac{10}{5} = 2$$

And

$$\hat{C}_1 = \frac{\sum_{i=1}^{m-1} S_i S_{i+1}}{m - 1} = \frac{2(1) + 1(0) + 0(-1) + (-1)(-2)}{5 - 1} = \frac{2 + 0 + 0 +}{4}$$

Giving

$$\hat{\rho}_1 = \frac{1}{2}$$

**Attribution**

This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Autoregressive process

## Contents


- Process definition
- Simulated example
- Estimation of coefficients of AR process


The code on this page can be used interactively: click 🚀 –> **Live Code** in the top right corner, then wait until the message **Python interaction ready!** appears.

The main goal is to introduce the AutoRegressive (AR) model to describe a **stationary stochastic process**. Hence the AR model can be applied on time series where e.g. trend and seasonality are not present / removed, and only noise remains, or after applying other methods to obtain a stationary time series.

## Process definition

In an AR model, we forecast the variable of interest using a linear combination of its past values. A zero mean AR process of orders $p$ can be written as follows:

$$S_t = \overbrace{\phi_1 S_{t-1} + \ldots + \phi_p S_{t-p}}^{\text{AR process}} + e_t$$

or as

$$S_t = \sum_{i=1}^{p} \phi_i S_{t-i} + e_t$$

Each observation is made up of a **random error** $e_t$ at that epoch, a linear combination of **past observations**. The errors $e_t$ are uncorrelated purely random noise process, known also as white noise. We note the process should still be stationary, satisfying

$$\mathbb{E}(S_t) = 0, \quad \mathbb{D}(S_t) = \sigma^2, \quad \forall t$$

This indicates that parts of the total variability of the process come from the signal and noise of past epochs, and only a (small) portion belongs to the noise of that epoch (denoted as $e_t$ ).

# First-order AR(1) process

We will just focus on explaining $p = 1$, i.e. the AR(1) process. A **zero-mean first order autoregressive** process can be written as follows

$$S_t = \phi S_{t-1} + e_t, \quad -1 \leq \phi < 1, \quad t = 2, \ldots, m$$

where $e_t$ is an i.i.d. noise process, e.g. distributed as $e_t \sim N(0, \sigma_e^2)$. See later the definition of $\sigma_e^2$.

**Exercise**

In a zero-mean first order autoregressive process, abbreviated as AR(1), we have $m = 3$ observations, $\phi = 0.8$, and the generated white noise errors are $e = [e_1, e_2, e_3]^T = [1, 2, -1]^T$. What is the generated AR(1) process $S = [S_1, S_2, S_3]^T$?

a. $S = [1 \quad 2.8 \quad 1.24]^T$
b. $S = [0 \quad 2 \quad 0.6]^T$
c. $S = [1 \quad 2 \quad -1]^T$

**Solution**

The correct answer is **a**. The AR(1) process can be initialized as $S_1 = e_1 = 1$. The next values can be obtained through:

$$S_t = \phi S_{t-1} + e_t$$

Giving $S_2 = 0.8 S_1 + e_2 = 0.8 \cdot 1 + 2 = 2.8$ and $S_3 = 0.8 S_2 + e_3 = 0.8 \cdot 2.8 - 1 = 1.24$, so we have:

$$S = [1 \quad 2.8 \quad 1.24]^T$$

# Formulation

Initializing $S_1 = e_1$, with $\mathbb{E}(S_1) = \mathbb{E}(e_1) = 0$ and $\mathbb{D}(S_1) = \mathbb{D}(e_1) = \sigma^2$. Following this, multiple applications of the above "autoregressive" formula ($S_t = \phi S_{t-1} + e_t$) gives:

$$\begin{aligned} S_1 &= e_1 \\ S_2 &= \phi S_1 + e_2 \\ S_3 &= \phi S_2 + e_3 = \phi^2 S_1 + \phi e_2 + e_3 \\ &\vdots \\ S_m &= \phi S_{m-1} + e_m = \phi^{m-1} S_1 + \phi^{m-2} e_2 + \ldots + \phi e_{m-1} + e_m \end{aligned}$$

of which we still have (in order to impose the *stationarity*):

$$\mathbb{E}(S_t) = 0 \text{ and } \mathbb{D}(S_t) = \sigma^2, \;\; t = 1, \ldots, m$$

All the error components, $e_t$, are uncorrelated such that $Cov(e_t, e_{t+\tau}) = 0$ if $\tau \neq 0$, and with variance $\sigma_e^2$ which still needs to be determined.

# Autocovariance

The mean of the process is zero and, therefore:

$$\mathbb{E}(S_t) = \mathbb{E}(\phi S_{t-1} + e_t) = \phi \mathbb{E}(S_{t-1}) + \mathbb{E}(e_t) = 0$$

The variance of the process should remain constant as:

$$\mathbb{D}(S_t) = \mathbb{D}(\phi S_{t-1} + e_t) \Leftrightarrow \sigma^2 = \phi^2 \sigma^2 + \sigma_e^2, \;\; t \geq 2$$

resulting in

$$\sigma_e^2 = \sigma^2 (1 - \phi^2)$$

indicating that $\sigma_e^2$ is smaller than $\sigma^2$.

The autocovariance (covariance between $S_t$ and $S_{t+\tau}$) is

$$\begin{aligned} c_\tau &= \mathbb{E}(S_t S_{t+\tau}) - \mu^2 = \mathbb{E}(S_t S_{t+\tau}) \\ &= \mathbb{E}(S_t(\phi^\tau S_t + \phi^{\tau-1} e_{t+1} + \ldots)) = \phi^\tau \mathbb{E}(S_t^2) = \sigma^2 \phi^\tau \end{aligned}$$

In the derivation above we used that:

$$S_{t+\tau} = \phi^{\tau} S_t + \phi^{\tau-1} e_{t+1} + \ldots + e_{t+\tau}$$

and the fact that $S_t$ and $e_{t+\tau}$ are uncorrelated for $\tau \neq 0$.

**Derivation (optional)**

$$\begin{aligned} S_{t+\tau} &= \phi^{t+\tau-1} S_1 + \phi^{t+\tau-2} e_2 + \ldots + \phi^{\tau} e_t + \phi^{\tau-1} e_{t+1} + \ldots + e_{t+\tau} \\ &= \phi^{\tau} \left( \phi^{t-1} S_1 + \phi^{t-2} e_2 + \ldots + e_t \right) + \phi^{\tau-1} e_{t+1} + \ldots + e_{t+\tau} \\ &= \phi^{\tau} S_t + \phi^{\tau-1} e_{t+1} + \ldots + e_{t+\tau} \end{aligned}$$

# Model structure of AR(1)

$$\mathbb{E}(S) = \mathbb{E} \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \mathbb{D}(S) = \Sigma_S = \sigma^2 \begin{bmatrix} 1 & \phi & \ldots & \phi^{m-1} \\ \phi & 1 & \ldots & \phi^{m-2} \\ \vdots & \vdots & \ddots & \vdots \\ \phi^{m-1} & \phi^{m-2} & \ldots & 1 \end{bmatrix}$$

- Autocovariance function $\implies c_\tau = \sigma^2 \phi^\tau$
- Normalized autocovariance function (ACF) $\implies \rho_\tau = c_\tau / c_0 = \phi^\tau$
- Larger value of $\phi$ indicates a long-memory random process
- If $\phi = 0$, this is called *purely random process* (white noise)
- ACF is even, $c_\tau = c_{-\tau} = c_{|\tau|}$ and so is $\rho_\tau = \rho_{-\tau} = \rho_{|\tau|}$

Later in this section we will see how the coefficient $\phi$ can be estimated.

# Simulated example

If you have run the python code on this page, an interactive plot will be displayed below. You can change the value of $\phi$ and the number of observations $m$ to see how the AR(1) process changes. At the start, the process is initialized with $\phi = 0.8$. Try moving the slider and see the response of the ACF; pay special attention when $\phi = 0$ and when $\phi$ becomes negative.

Lastly, focus on the case where $\phi = 1$ and $\phi = -1$. What do you observe? You will notice that the function will “explode”. This makes intuitive sense, since the effect of the previous

epoch is not dampened, but rather amplified. This also means that the process is not stationary anymore. So, the AR(1) process is stationary if $|\phi| < 1$.

# Estimation of coefficients of AR process

If the values of $p$ and of the AR($p$) process are known, the question is: **how can we estimate the coefficients** $\phi_1, \ldots, \phi_p$

Here, we only elaborate on AR(1) using best linear unbiased estimation (BLUE) to estimate $\phi_1$. The method can be generalized to estimate the parameters of an AR($p$) process.

**Example: Parameter estimation of AR(1)**

The AR(1) process is of the form

$$S_t = \phi_1 S_{t-1} + e_t$$

In order to estimate the $\phi_i$ we can set up the following linear model of observation equations (starting from $t = 2$):

$$\begin{bmatrix} S_2 \\ S_3 \\ \vdots \\ S_m \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{m-1} \end{bmatrix} [\phi_1] + \begin{bmatrix} e_2 \\ e_3 \\ \vdots \\ e_m \end{bmatrix}$$

The BLUE estimator of $\phi$ is given by:

$$\hat{\phi} = (\mathrm{A}^T \mathrm{A})^{-1} \mathrm{A}^T S$$

Where $\mathrm{A} = \begin{bmatrix} S_1 & S_2 & \cdots & S_{m-1} \end{bmatrix}^T$ and $S = \begin{bmatrix} S_2 & S_3 & \cdots & S_m \end{bmatrix}^T$.

> **Attribution**
>
> This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Fit AR(p) model

In this exercise, we will fit an AR(p) model to a time series.The timeseries is already stationary and we will use two different methods to choose the number of lag (p) to include. First we will start by loading and displaying the data.

```
#plot data
plt.figure(figsize=(12, 4))
plt.plot(data)
plt.xlabel('Time')
plt.ylabel('Value')
plt.title('Generated stationary Time Series')
plt.grid(True)
```

run | run all | add cell | clear

Next we can use `plot_acf` to plot the autocorrelation function of the time series. From the resulting plot we can see that that there is autocorrelation present in the data.

```
# Plot the ACF of the time series
fig, ax = plt.subplots(figsize=(12, 4))
plot_acf(data, lags=40, ax=ax)
ax.grid(True);
```

run | run all | add cell | clear

To make sure that we have selected the correct number of lags, we can test for significance using the likelihood ratio test (LRT). LRT is similar to the GLRT from chapter 3, but now we have made some assumptions on the standard deviations of the observations. We will apply the LRT to the AR(1) vs AR(2) and AR(2) vs AR(3) models.

The AR(1), AR(2) and AR(3) models are defined as follows:

AR(1): $S_t = \phi_1 S_{t-1} + \epsilon_t$

AR(2): $S_t = \phi_1 S_{t-1} + \phi_2 S_{t-2} + \epsilon_t$

AR(3): $S_t = \phi_1 S_{t-1} + \phi_2 S_{t-2} + \phi_3 S_{t-3} + \epsilon_t$

Since we need $S_{t-3}$ we will have $y = S[3:]$

```
from scipy.stats import chi2

y = data[3:]
s1 = data[2:-1]
s2 = data[1:-2]
s3 = data[:-3]
n = len(y)

# AR(1) model
X1 = np.column_stack((s1)).T
phi_ar1 = np.linalg.inv(X1.T @ X1) @ (X1.T @ y)
e1 = y - X1 @ phi_ar1
rss1 = e1.T @ e1

# AR(2) model
```

run | run all | add cell | clear

From the LRT results we can see that an AR(2) model is the best fit for the data. Finally we will fit the AR(2) model to the data and plot the residuals. The residuals should be white noise, which is confirmed by the ACF plots of the residuals. Plotting

```
# plot the residuals

fig, ax = plt.subplots(2,1,figsize=(12, 6))
ax[0].plot(e2)
ax[0].set_title('AR(2) Residuals')
ax[0].grid(True)

plot_acf(e2, lags=40, ax=ax[1])
ax[1].grid(True)

fig.tight_layout()


sigma_e2 = np.std(e2)
var_phi2 = sigma_e2**2 * np.linalg.inv(X2.T @ X2)
```

run | run all | add cell | clear

**Attribution**

This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Forecasting

**MUDE Exam Information**

The material on this page is provided to give you extra insight into time series analysis and how it is used in practice. This material is not part of the exam.

Time series analysis is about analyzing time series of data points of a variable, with the goal to extract meaningful characteristics and statistics of the data, e.g., to study the trend and seasonality. Very often we do so in order to be able to predict future values based on the previously observed ones, which is referred to as **forecasting**.

In this part of the book, you first learned about the components that can be distinguished (Chapter 4.1) in time series. Some of the components may relate to the *signal-of-interest*. However, another important component that we have to deal with is the noise.

Modelling and estimating the signal-of-interest (Chapter 4.3) using the concepts of observation theory. The remainder of this chapter focused on the noise modelling. In order to do so, we need to work with *stationary*, i.e., time series of which the statistical properties do not depend on the time of observation (Chapter 4.4). An example of a stationary time series are the residuals after best linear unbiased estimation. Using these residuals as the input for noise modelling makes sense, since in fact the residuals are estimates of the noise.

A problem with the noise process of a time series is that often there is auto-correlation: in contrast to a white noise signal, the observations with different time lags depend on each other - this is referred to as colored noise. The dependence can be modelled by the autocovariance function (Chapter 4.5). With that, the noise process can be modelled using the Autoregressive model (Chapter 4.6).

Now that we are able to model both signal-of-interest and the noise process, we can start forecasting.

In summary, given a time series $Y = \mathrm{A}\mathrm{x} + \epsilon$, the workflow is as follows:

1. Estimate the signal-of-interest $\hat{X} = (\mathrm{A}^T\Sigma_Y^{-1}\mathrm{A})^{-1}\mathrm{A}^T\Sigma_Y^{-1}Y$.
2. Model the noise using the Autoregressive (AR) model, using the stationary time series $S := \hat{\epsilon} = Y - \mathrm{A}\hat{X}$.

3. Predict the signal-of-interest: $\hat{Y}_{signal} = \mathrm{A}_p \hat{X}$, where $\mathrm{A}_p$ is the design matrix describing the functional relationship between the future values $Y_p$ and $\mathrm{x}$.
4. Predict the noise $\hat{\epsilon}_p$ based on the AR model. $\hat{\epsilon}_p = \Sigma_{Y_p Y} \Sigma_Y^{-1} \hat{\epsilon}$, where $\Sigma_{Y_p Y}$ is the covariance matrix between the future values $Y_p$ and the observed values $Y$.
5. Predict future values of the time series: $\hat{Y}_p = \mathrm{A}_p \hat{X} + \hat{\epsilon}_p$.

**Note**

This procedure is a general approach to forecasting time series data. It resembles the process of stochastic inter- and extrapolation, which is used in many fields of science and engineering.

**Attribution**

This chapter was written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen. Find out more here.

# Optimization

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

In this chapter you will learn all the basics of optimization. The chapter covers the origins of optimization as well as the main methods that exist to solve these problems. You will learn the following:

- Formulate a mathematical program
- Solve a linear program (LP) through the graphical solution method
- Solve a linear program through the SIMPLEX method
- Solve an integer programming problem with the branch and bound method
- Solve an integer programming problem with a genetic algorithm (meta-heuristic method)

Parts of this chapter are optional. In these cases, this will be indicated at the top of the page.

# Optimization origins

## Contents


- The radar
- 1947: the Simplex method is developed


In the UK and the US, scientists started to be called between WWI and WWII to collaborate with the military in doing research on military operations. A second WW was on the horizon and both countries wanted to be prepared by optimizing their logistics.They created a field of applied sciences known as **Operations Research** in which **Optimization** has been placed.

## The radar

Modern operations research originated in the UK in 1937 and was the result of an initiative of the superintendent, A. P. Rowe.

Rowe conceived the idea to analyze and improve the working of the UK's early warning radar system, Chain Home (CH). Initially, he analyzed the operation of the radar equipment and its communication networks to provide a complete vision of the south coast of the UK. How many radars do you need? Where should they be located? The analysis was later expanded to include the operating personnel's behavior to plan the Human Resources (HR) of this system.

This revealed unappreciated limitations of the CH network and allowed remedial action to be taken which helped win the war.

## 1947: the Simplex method is developed

George Dantzig worked on planning methods for the US Army Air Force during World War II using a desk calculator. In 1946 he was challenged to mechanize/automated the planning process that he was using.

Dantzig formulated the planning problem, typically a problem of assigning resources to activities, as linear inequalities (or equalities) inspired by the work of Wassily. Leontief,

however, at that time didn't include an objective as part of his formulation, he was mainly searching for feasible solutions to a problem.

# Leontief's work: The Input Output matrix

In this example a simple economy is described through three main products/activities, Coal, Electricity, and Steel. It's easy to produce linear equations where the interdependencies between these products are apparent once you have the production factors (how much you need from each one of those to produce the other).

Using the right data and this logic it's possible to describe the functioning of many systems and their corresponding "products". This is what has inspired Dantzig for his mathematical structure of an optimization problem, the economy structure generates the constraints of what is possible to produce (the solution space).

$$
\begin{aligned}
p_C &= 0.4p_E + 0.6p_S \\
p_E &= 0.6p_C + 0.1p_E + 0.2p_S \\
p_S &= 0.4p_C + 0.5p_E + 0.2p_S
\end{aligned}
$$

Input/output matrix in a simple economy:

| **Production / Consumption** | **Coal ($p_C$)** | **Electricity ($p_E$)** | **Steel ($p_S$)** |
|---|---|---|---|
| Coal | $0\%$ | $40\%$ | $60\%$ |
| Electricity | $60\%$ | $10\%$ | $20\%$ |
| Steel | $40\%$ | $50\%$ | $20\%$ |
| | | | |
| | $100\%$ | $100\%$ | $100\%$ |

# Adding an objective to the problem

Without an objective, a vast number of solutions can be feasible. To find the "best" feasible solution, we need an expression to describe how goals can be achieved as opposed to specifying a specific value for this goal on itself. For example, it's not about transporting 1000 soldiers but finding a way to transport as many as possible with the existing resources.

Dantzig's core insight was to realize that most such ground objectives can many times be translated into a **linear objective function** that needs to be maximized (or minimized) that measures the quality/performance of the solutions. Development of the final method, the so-called **simplex** method, was evolutionary and happened over a period of about a year.

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Optimization basics

## Contents


- General components of an optimization problem
- Optimization vs Simulation
- Optimization: why and how?


## General components of an optimization problem

In general you have the following linear (simplest) structure for a mathematical programming problem. The first equation is the objective function:

$$\underbrace{\text{Max (Min) } Z}_{\text{objective function}} \quad = \underbrace{c_1x_1 + c_2x_2 + \ldots + c_nx_n}_{\text{objective function coefficients}}$$

this objective function is subject to a set of constraints given by:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n (\leq, =, \geq) b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n (\leq, =, \geq) b_2 \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n (\leq, =, \geq) b_m \end{cases}$$

In this case $a_{ij}$ refer to technology coefficients that measure the usage of a resource, $x$ refers to the decision variables, $b_m$ is an independent term, which usually translates as an available resource, $n$ is the number of variables, and $m$ is the number of constraints.

## Optimization vs Simulation

**Optimization**

Some modeling approaches attempt to provide optimal answers for problems (e.g., mathematical programming) or near optimum answers (e.g., heuristic and meta-heuristic methods). That's what this week is all about.

**Simulation**

A simulation model predicts the performance of a system under a specific set of inputs (experimental parameters).In general, with simulation, we are not searching for an optimal solution but for the system's performance under different scenarios that are selected according to their importance or likelihood.

**Understanding the difference between both**

Imagine the case of planning a bus line through simulation. You have your route defined (the streets where it's going to go through) and the demand around that route is dependent on the frequency of the buses and the bus stop distance. You want to simulate the bus operation in order to maximize your profit.

The variables involved are the bus fleet size, $b$, and the number of stops in the line, $s$. They can be bounded to form a range:

- $5 \leq b \leq 15$ -> we have 11 bus fleet dimensions to test in a simulation model;
- $10 \leq s \leq 30$ -> we have 21 possible stops dimensions

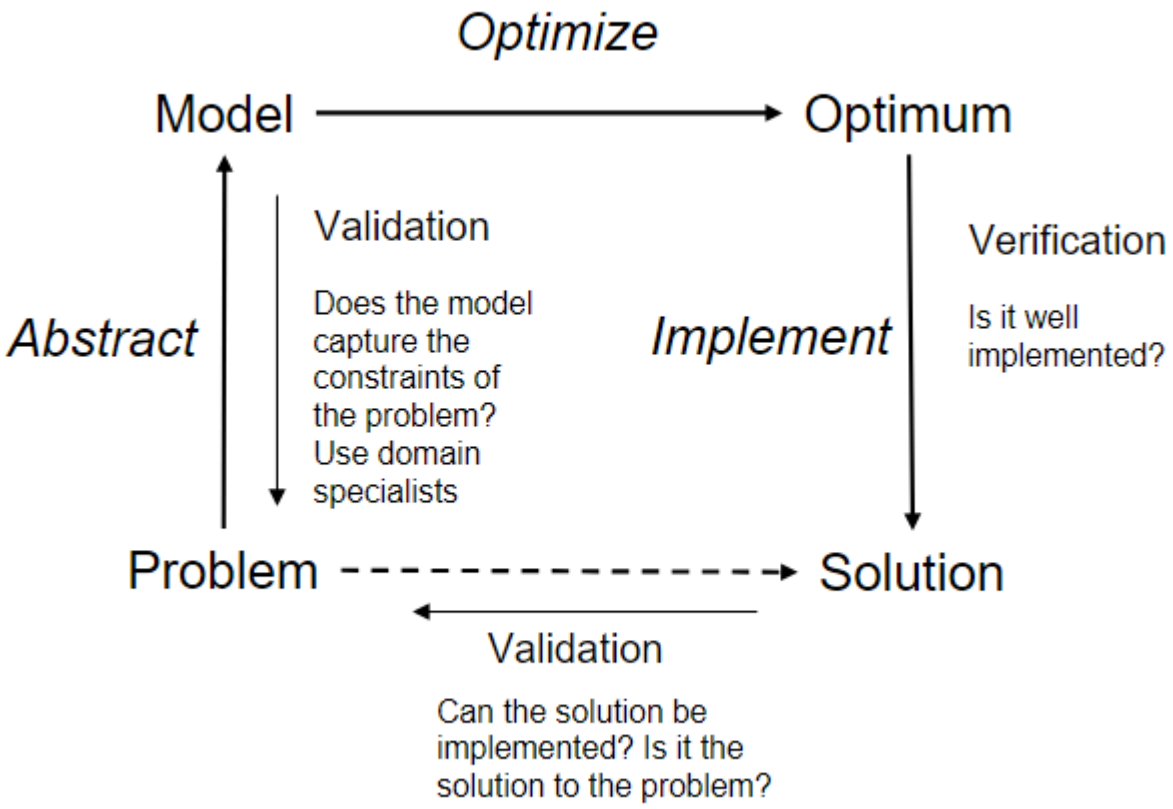
Therefore, we have to test a total of $b \cdot s = 231$ combinations, which is still a manageable number of combinations to simulate.

But what if we are talking about a route line that was not designed yet? Then, for each combination of fleet and number of stops you would have to test the factorial of the number of stops (all the combinations of stops that form a path), $\frac{(n-1)!}{2}$.

This leads to an impractical number of scenarios to test in simulation! This problem would be better studied using optimization techniques in the so-called network design problems which can be solved, to a certain extent, with mathematical programming as well.

# Optimization: why and how?

The typical workflow of an optimization process looks like the schematic shown below:




# Why optimization?

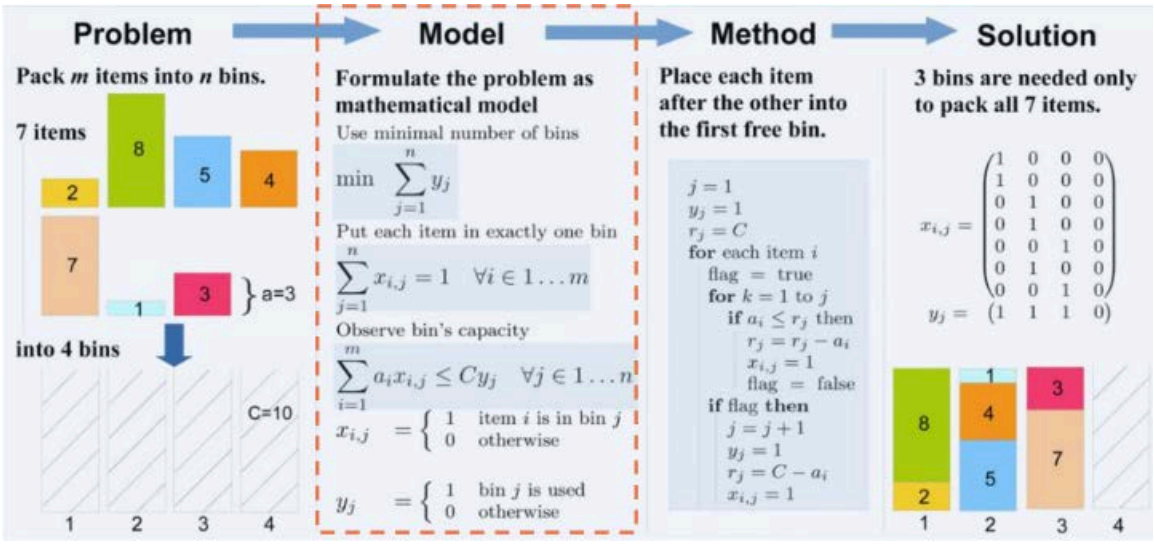
Many problems require the identification of **the best** (or worst) solution, decision or design.

> **Note**
>
> Warning!
>
> This is a concept that is usually misused. Stating something like “optimizing the design of a bridge” to refer to the *improvement* of the design does not mean we are aiming for the best design, just for a better one.

Optimization techniques have multiple applications as part of other mathematical tools (as LS, MLE, or ML), but also direct applications as optimal structural design, resource distribution, logistics, etc. Let us take a deeper look into some of these examples

# What is included under the concept of

optimization?




# Examples of optimization problems

In this section, you can find a few examples of optimization problems with their respective goals.

**Example 1 - Optimizing the layout of the offshore wind farms in Norway**

**Goal**: determine the optimal layout of wind turbines to produce the *highest amount of energy at minimum costs*. Considering different types of turbines, their power curves, physical characteristics, wake effect, etc.

This example is taken from CME4500 Engineering Systems Optimization, final project by Justus Aulbers, Ola Åsbø & Illias Timori

### Example 2 - Optimizing the renewable energy configuration for a completely self-sustaining energy system in the Ijsselmeer area in 2050

**Goal**: determine the most economic configuration of areas for wind turbines, solar panels, algae, and battery storage to meet the hourly energy demand in the future for a certain area. Considering the fluctuating demand, climatic conditions, etc.

This example is taken from CME4500 Engineering Systems Optimization, final project by Max Driessen & Himanshu Patel

### Example 3 - Optimization of electric vehicles charging station locations

**Goal**: identify the optimal distribution of charging stations that maximizes the utility accounting for the population density, existence of other charging stations, and other urban elements (e.g. rivers, parks, etc.)

This example is taken from CME4500 Engineering Systems Optimization, final project by Tomas Raaphorst & Vitali van Elk

### Example 4 - Optimizing the location of new electric vehicle charging stations

**Goal**: identify the most economic location of EV charging stations to satisfy the users' requirements.

This example is taken from the MSc Thesis of Dawei Fu.

### Example 5 - Optimizing monocoque FRP bridges

**Goal:** identify the optimal geometry and plies layout with consideration of the tradeoff between cost and sustainability. The design must guarantee the structural safety conditions.

This example is taken from the MSc Thesis of Ola Åsbø

### Example 6 - Optimal component level construction schedule

**Goal:** determine the most efficient process for building components that have physical interdependencies (fulfilling constructive constraints), with consideration of the tradeoff between the cost and construction duration.

This example is taken from the MSc Thesis of Xinzhi Jiang

### Example 7 - Optimizing the location of shared mobility hubs

**Goal:** identify the optimal locations of mobility hubs that make available several modes, such as shared bikes, shared cars and cargo bikes maximizing accessibility and minimizing costs with consideration of the demand.

This example is taken from the MSc thesis of Stavros Xanthopoulos

### Example 8 - Designing road networks for automated vehicles

**Goal:** identify the best location and time for deploying enhanced roads for automated vehicles, minimizing deployment cost and maximizing efficiency and safety given the uncertain evolution path of automated driving technology and the travelers' mode and route choice behavior.

This example is taken from the PhD thesis of Bahman Madadi

### Attribution

This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Taxonomy of optimization models

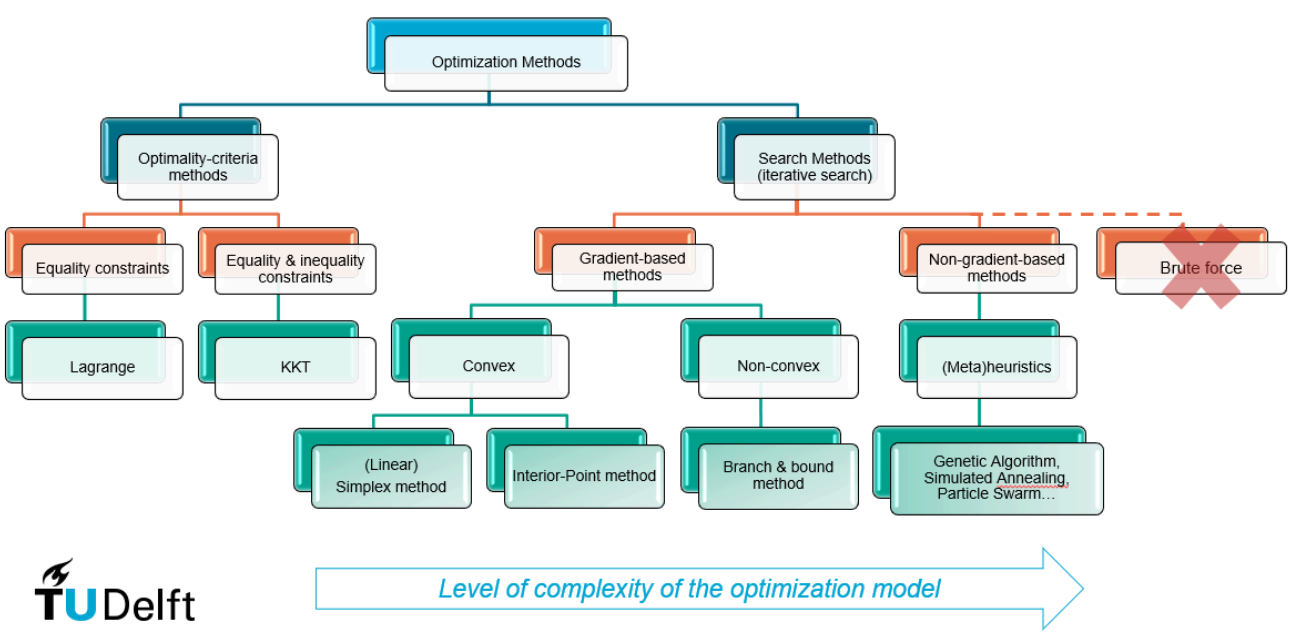
## Contents


- Continuous vs Discrete variables
- Single vs Multi-objective
- Constrained vs Unconstrained problems
- Convex vs Non-convex problems
- Deterministic vs Stochastic


In this section we will discuss the difference between different types of optimization models:

- Continuous vs Discrete variables
- Single vs Multi-objective
- Convex vs Non-convex problems
- Deterministic vs Stochastic
- Constrained vs Unconstrained problems

Below is an overview of the different types of optimization models and their relationship with each other:




# Continuous vs Discrete variables

The name is self-explanatory for the difference between these two types of variables:

- **Continuous variables:** time, distances, physical properties, etc.
- **Discrete variables:** number of wind turbines, decisions such as doing something or not, type of materials, etc.

Imagine we have a problem to find the optimal wind turbine (WT) farm. In this problem, we have two decision variables:

- $n$, the number os WTs ($10 \leq n \leq 50$)
- $d$, the closest distance between two WTs ($15 \leq n \leq 100$)

We can clearly see that $n$ is a discrete variable and $d$ is a continuous variable!

# Single vs Multi-objective

It depends on the number of objective functions we need to define to address our problem. Examples 4 and 6 from the last section are good examples of single and multi-objective problems, respectively.

Consider the example presented below of the optimal WT farm. If you define the objective functions to:

- maximize the annual production (Max$_{n,d} P_{\text{unit}}$ )
- minimize the annual maintenance cost (Min$_{n,d} nC_m$)

These are single objective problems. However, if we consider both objectives at once, we are talking of a multi-objective problem!

$$\text{Min}_{n,d}\{-P_{\text{unit}} \ ; nC_m\}$$

# Constrained vs Unconstrained problems

**Constrained problems:** the solution space is bounded:

- *Feasible region:* only a set of solutions are possible candidates;
- *Unfeasible region:* there is not a possible solution fulfilling all the constraints

**Unconstrained problems:** the solution space is not bounded. All the configurations are possible candidates for being optimal.

Still considering the WT example, if we consider a limited construction budget of 50M euros or a limited annual maintenance cost of 0.7M euros, then we are dealing with a constrained problem. In that case, if the construction or the maintenance costs are higher that the values defined, there is no possible solution!

$$nC \leq 50M$$
$$nC_m \leq 0.7M$$

# Convex vs Non-convex problems

**Definition:** A convex optimization problem is a problem where all the constraints and the objective are convex functions.

taxonomy

Agrawal2021

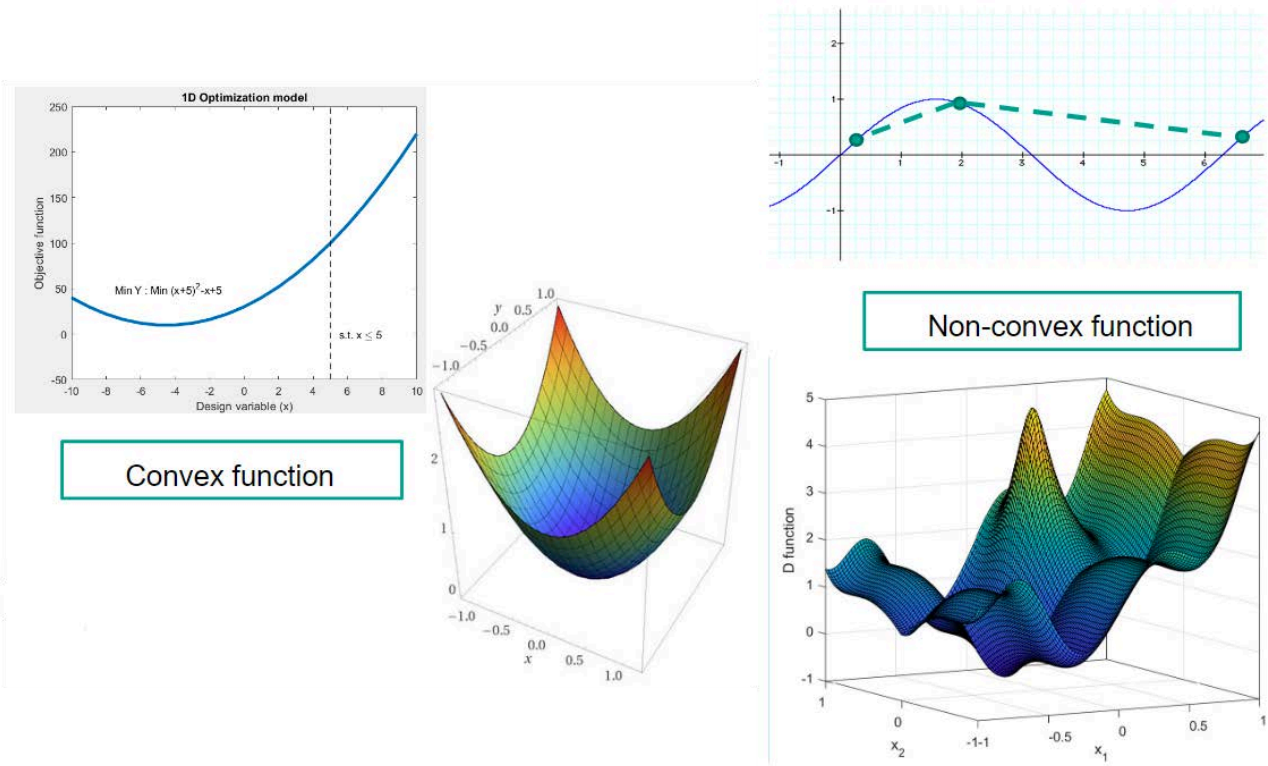



*Fig. 5.31* This figure includes content from Agrawal (2021) and Nogal and Nogal (2021). Find out more here.

In the WT farm case, even though the constraints defined above are linear (convex), the objective function is not. Then, it is considered a non-convex optimization problem.

# Deterministic vs Stochastic

**Deterministic optimization:** all the parameters of the optimization problem are deterministic. There is not variability in the problem definition.

**Stochastic optimization:** the definition of optimization problem presents variability or uncertainty. The optimal solution of a possible scenario is not necessarily the optimal solution of another possible scenario.

The problem as defined with the constraints above is deterministic. However, in case the construction, $C$, and the maintenance costs, $C_m$, present variability, the problem is stochastic and we find, for example:

$$\text{Prob}(nC \leq 50M) \geq 0.9$$
$$\text{Prob}(nC_m \leq 0.7M) \geq 0.95$$

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Example Linear Programming

## Contents


- Videos
- The problem

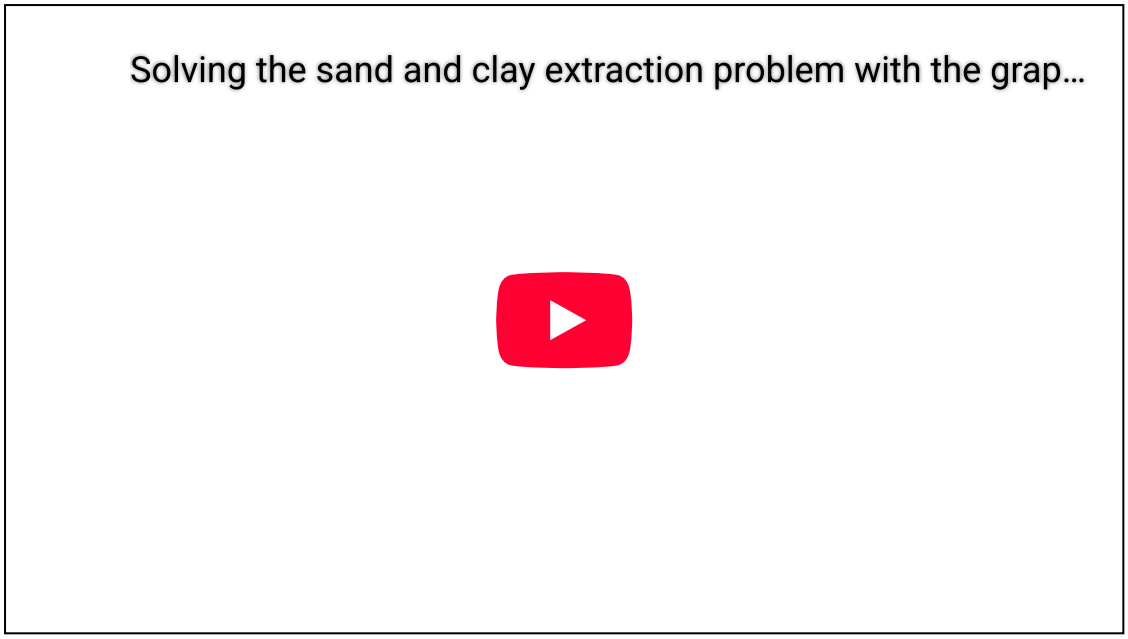

## Videos

The story is told in two videos. The videos have a one-to-one correspondence with this book

### The sand and clay extraction problem

### Solving the sand and clay extraction problem

# with the graphical solution method




# The problem

A company extracts sand and clay from a site which when sold gives a profit of 57 and 60 monetary units per thousand units of product, respectively. For this extraction, sand needs a manpower of 50 workers x hours to extract 1000 units of product whilst clay needs 13 workers x hours for 1000 units.

4 hours of backhoe are needed to extract one thousand units of sand and 5 hours of backhoe work per thousand units of clay. The number of hours needed of truck transport is 8h and 4h, respectively for sand and clay for each 1000 units of product transported.

The company has a work schedule of 40 hours per week for the workers but also for the equipment (truck and backhoe). There are 5 workers who can be used interchangeably between the transport of the two products. There is only one truck and one backhoe.

**Main questions**:

- What should be the extraction plan of this company in a week?
- What should their objective be?
- What decisions need to be made?

# LP formulation of the problem

- $x_1$ - thousands of units of sand to be produced and transported in one week

- $x_2$ - thousands of units of clay to be produced and transported in one week

Our objective function then will look like (with $L$ being the profit in monetary units for one week):

$$\text{Max} L = 57x_1 + 60x_2$$

subject to:

- $8x_1 + 4x_2 \leq 40$: maximum number of hours of the vehicle driving time per week
- $4x_1 + 5x_2 \leq 40$: maximum number of hours of crane work per week
- $50x_1 + 13x_2 \leq 200$: maximum of 5 (workers) x 40 hours of driving time available per week per worker
- $x_1, x_2 \geq 0$: the production must be positive!

**Quiz questions**

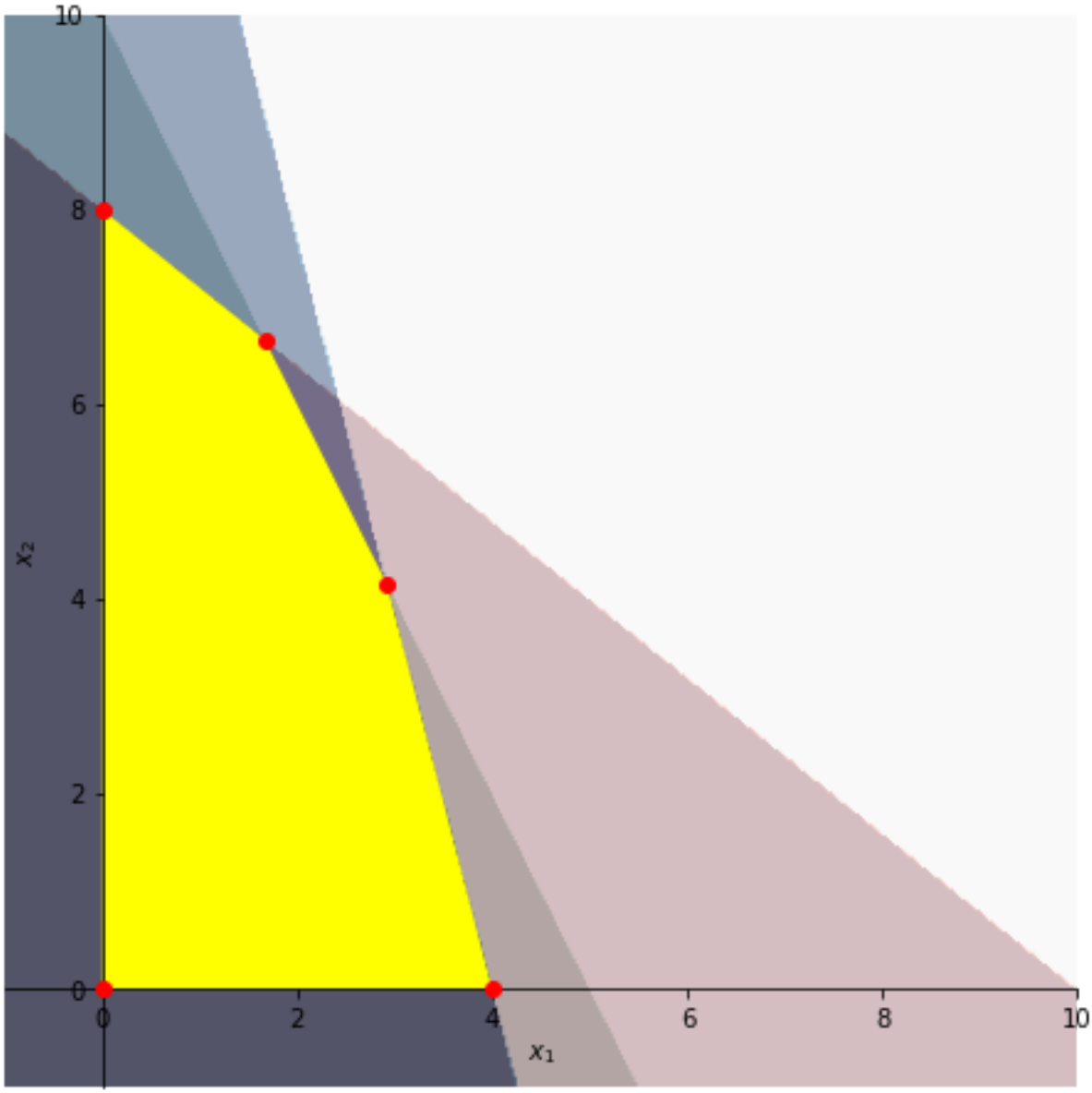
Do you need to know what each specific man does?

- Yes, so each man can be assigned to one particular task.
- Yes, each man is different so we don't want to mistake them.
- No, the men cannot be differentiated, so it doesn't matter.

Check

# Solution

The above mentioned constraints define the feasible region of the solution space. The feasible region is represented by the yellow polygon on the graph below:




The regions at green, blue, and red correspond to the first three constraints defined in the previous subsection, in the same order. The yellow region is a result of the super position of these three regions alongside $x_1 \geq 0$ and $x_2 \geq 0$. All the points inside and in the border of the yellow region are solutions for our problem!

**How can we find the optimal solution now?**

Our solution space, in this case, is a closed polygon, which is always **convex** in LP problems (it is easily seen in the graph represented above that any line connecting two points inside our solution space is always inside the polygon).

**Theorem**

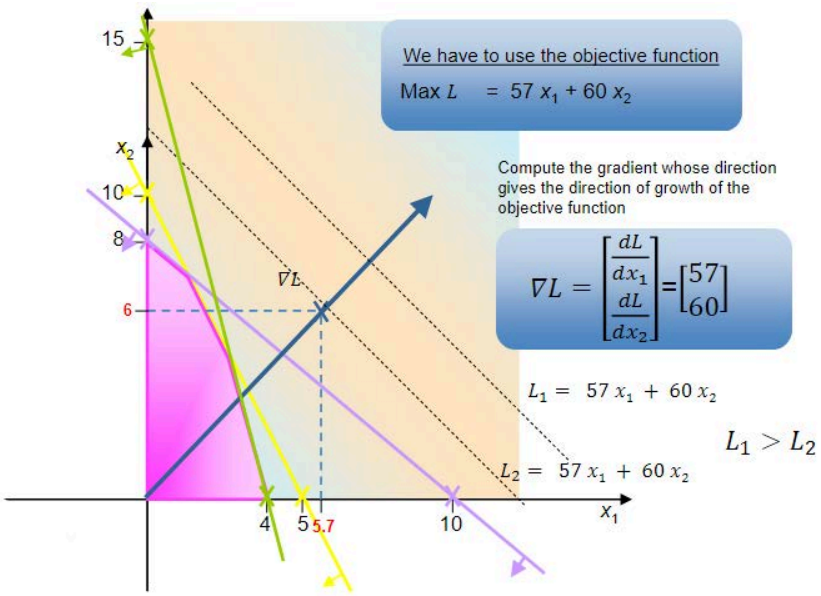
The set of feasible solutions is a convex set whose extremes (vertices) correspond to feasible basic solutions. If there is, at least, a feasible solution to the problem, then there is a feasible basic solution to the problem.

If the objective function has a finite maximum (minimum) there is at least an optimal solution and that is a feasible basic solution (one of those vertices).

We consider as **basic solutions** any of the red vertices of the yellow region, and any other solution that is inside the highlighted yellow region consists of a **non-basic** solution.

To find the optimal solution, we can start now by computing the gradient vector of the objective function whose direction gives us the direction of growth of the objective function, i.e.:

$$\nabla L = \begin{bmatrix} \frac{dL}{dx_1} \\ \frac{dL}{dx_2} \end{bmatrix} = \begin{bmatrix} 57 \\ 60 \end{bmatrix}$$




With this, we just need to find the last point inside the feasible region in the direction of the gradient (maximization) or the first point inside the region (minimization). That point will be one of the vertices (basic solutions), and in this example we get:

$$X = \left(\frac{5}{3}, \frac{20}{3}\right) \implies L = 485$$

If we calculate the value of the objective function for the remaining vertices we confirm that the one we obtained above consists in the one that maximizes $L$:

- $\mu_1 = (0, 8) \implies L = 480$
- $\mu_2 = \left(\frac{35}{12}, \frac{25}{6}\right) \implies L = 416.5$
- $\mu_3 = (4, 0) \implies L = 228$
- $\mu_4 = (0, 0) \implies L = 0$

**Quiz questions**

The intersection of the gradient vector with the feasible solution space polygon determines the optimal solution.

○ True ○ False

Check

**Attribution**

This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Implementation in Python

## Contents


- Implementation in Python
- Exercise: The sand and clay extraction problem


In this Notebook, we show how you can use Gurobi with Python API to solve the clay and sand extraction problem.

> **Note**
>
> Gurobi cannot be loaded in this online book, so download this notebook to work on it with Gurobi locally installed. Instruction on how to do that are in the `README.md` and `PA_2_4_A_gurobilicious.ipynb` of week 2.4.

## Problem description

- A company extracts sand and clay from a site which when sold gives a profit of 57 and 60 monetary units per thousand units of product. For this extraction, sand needs a manpower of 50 men x hours to extract 1000 units of product 1 whilst clay needs 13 men x hours for 1000 units.
- 4h of backhoe work are needed to extract one thousand units of sand and 5h of backhoe work per thousand units of clay. The number of hours needed of truck transport is 8h and 4h respectively for sand and clay for each 1000 units of product transported.
- The company has a work schedule of 40 hours per week for the men but also for the equipment (truck and backhoe). There are 5 men who can be used interchangeably between the transport of the two products. There is only one truck and one backhoe.
- What should be the extraction plan of this company in a week? What should their objective be? What decisions need to be made?

# Gurobi tutorial

- In this notebook, we are going to build a simple model using gurobipy.
- You can turn to the built-in function help() to see how different commands work, for example:

```
help(model.addVar())
```

# Import gurobipy package

After preparing the environment for gurobi+python, you should be able to run the following code to import the gurobipy package.

```
import gurobipy as gp
```

# Construct the model

## Initialization

- First, initialize an empty model.

```
model = gp.Model()
```

## Define attributes

- Now let's store some parameter values we need for constructing the model.
- The multydict() function in gurobipy allows us to store these values conveniently.

```
gurobipy.multidict()
```

```
help(gp.multidict)
```

```
Help on cython_function_or_method in module gurobipy:

multidict(data)
    ROUTINE:
      multidict(data)
    PURPOSE:
      Split a single dictionary into multiple dictionaries.
    ARGUMENTS:
      data: A dictionary that maps each key to a list of 'n' values.
    RETURN VALUE:
      A list of the shared keys, followed by individual tupledicts.
    EXAMPLE:
      (keys, dict1, dict2) = multidict( {
                'key1': [1, 2],
                'key2': [1, 3],
                'key3': [1, 4] } )
```

| | Profit | Manpower | Backhoe work | Truck transport |
|---|---|---|---|---|
| Sand | 57 | 50 | 4 | 8 |
| Clay | 60 | 13 | 5 | 4 |

```
# define attributes
material, profit, manpower, backhoe_work, truck_trans = gp.multidict({
    'sand' :[57, 50, 4, 8], 'clay' :[60, 13, 5, 4]
})
men_equipment, work_hour = gp.multidict({
    'men' :[200], 'truck' :[40], 'backhoe' :[40]
})
```

# Add desicion variables

- This step is to add desicion variables. You can use these two functions to add decision variables to the model.

```
Model.addVar() # add a single variable
Model.addVars() # add multiple variables
```

You can define the type of the variables. Such as:

```
GRB.CONTINUOUS  # continuous variable
GRB.INTEGER # integer variable
```

- Use the built-in function in the following cells to see detailed information about Model.addVar() and variable types in class GRB

---

```
help(gp.Model.addVar)
```

```
Help on cython_function_or_method in module gurobipy:

addVar(self, lb=0.0, ub=1e+100, obj=0.0, vtype='C', name='', column=None)
    ROUTINE:
      addVar(lb, ub, obj, vtype, name, column)

    PURPOSE:
      Add a variable to the model.

    ARGUMENTS:
      lb (float): Lower bound (default is zero)
      ub (float): Upper bound (default is infinite)
      obj (float): Objective coefficient (default is zero)
      vtype (string): Variable type (default is GRB.CONTINUOUS)
      name (string): Variable name (default is no name)
      column (Column): Initial coefficients for column (default is None)

    RETURN VALUE:
      The created Var object.

    EXAMPLE:
      v = model.addVar(ub=2.0, name="NewVar")
```

- Use following cell to add decision variables x into the model. $x_0$ and $x_1$ represent the amount (ton/week) of sand and clay that will be extracted (notice that the index starts from 0).
- Update the model after adding variables.

```
x = model.addVars(2, vtype = gp.GRB.CONTINUOUS, name = 'x')  # x0, x1
model.update() #Remember to update the model after adding decision variables
```

# Add constraints

You can use the following functions to add constrains to the model.

```
Model.addConstr()  # add a single constrain
Model.addConstrs()  # add multiple constrin
```

Besides, for convenience, you can use a function gurobipy.quicksum() to calculate sums and build expressions.

```
help(gp.quicksum)
```

```
Help on cython_function_or_method in module gurobipy:

quicksum(list)
    ROUTINE:
      quicksum(list)

    PURPOSE:
      A quicker version of the Python built-in 'sum' function for building
      Gurobi expressions.

    ARGUMENTS:
      list: A list of terms.

    RETURN VALUE:
      An expression that represents the sum of the input arguments.

    EXAMPLE:
      expr = quicksum([x, y, z])
      expr = quicksum([1.0, 2*y, 3*z*z])
```

Turn to built-in function help() to see detailed information like following.

```
help(gp.Model.addConstr)
```

```
Help on cython_function_or_method in module gurobipy:

addConstr(self, lhs, sense=None, rhs=None, name='')
    ROUTINE:
      addConstr(tc, name)

    PURPOSE:
      Add a constraint to the model.

    ARGUMENTS:
      tc (TempConstr): The constraint to add
      name (string): Constraint name (default is no name)

    RETURN VALUE:
      Depending on the data of 'tc':
        - A Constr object if tc arose from a linear expression
        - A QConstr object if tc arose from a quadratic expression
        - An MConstr object if tc arose from a linear matrix expression
        - A GenConstr object if tc arose form a general constraint expression

    EXAMPLE:
      c = model.addConstr(x + y <= 1)
      c = model.addConstr(x + y + z == [1, 2])
      c = model.addConstr(x*x + y*y <= 1)
      c = model.addConstr(z == and_(y, x, w))
      c = model.addConstr(z == min_(x, y))
      c = model.addConstr((w == 1) >> (x + y <= 1))
      c = model.addConstr(A @ x <= b)
```

In this step, you need to add the following constraints.

$$
\begin{aligned}
8x_0 + 4x_1 &\leq 40 \\
4x_0 + 5x_1 &\leq 40 \\
50x_0 + 13x_1 &\leq 200
\end{aligned}
$$

Note that non-negativity constraints are not always necessary in Gurobi since by default, variables are non-negative in Gurobi.

```
model.addConstr(gp.quicksum(manpower[material[i]] * x[i] for i in range(2)) <= work_hour[
model.addConstr(gp.quicksum(backhoe_work[material[i]] * x[i] for i in range(2)) <= work_h
model.addConstr(gp.quicksum(truck_trans[material[i]] * x[i] for i in range(2)) <= work_ho
model.update()
```

# Set objective function

- You can use the following command to set the objective

```
Model.setObjective()
```

- Use built-in function help() to see detailed information

```
help(gp.Model.setObjective)
```

```
Help on cython_function_or_method in module gurobipy:

setObjective(self, expr, sense=None)
    ROUTINE:
      setObjective(expr, sense=None)

    PURPOSE:
      Set the model objective equal to a LinExpr or QuadExpr

    ARGUMENTS:
      expr: The desired objective function.  The objective can be
            a linear expression (LinExpr) or a quadratic expression (QuadExpr).
            This routine will replace the 'Obj' attribute on model variables
            with the corresponding values from the supplied expression.
      sense (optional): Objective sense.  By default, this method uses the
            modelSense model attribute to determine the sense.  Use
            GRB.MINIMIZE or GRB.MAXIMIZE to ignore modelSense and pick a
            specific sense.

    RETURN VALUE:
      None.

    EXAMPLE:
      model.setObjective(x + y)
      model.setObjective(x + y + 2*z, GRB.MAXIMIZE)
```

This step is to set the objective function shown in the following:

> $$Max \quad L = 57x_0 + 60x_1$$

```
model.setObjective(gp.quicksum(profit[material[i]] * x[i] for i in range(2)), gp.GRB.MAXI
```

# Solve

Use following code to solve the model, and you can also show the optimization process log and limit the calculation time.

You can also print the objective function value and optimal decision variables values.

You can choose the algorithm used, or let Gurobi decide which algorithm is the best. See the following links for detailed information:

https://www.gurobi.com/documentation/9.5/refman/method.html

```
model.Params.LogToConsole = True  # show the optimization process log
model.Params.TimeLimit = 100  # limit the calculation time to 100s
model.Params.MultiObjMethod = 0  # default is -1 (let Gurobi decide)
model.optimize()
```

```
Set parameter TimeLimit to value 100
Set parameter MultiObjMethod to value 0
Gurobi Optimizer version 11.0.0 build v11.0.0rc2 (win64 - Windows 10.0 (19045.2))

CPU model: 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz, instruction set [SSE2|AVX|AVX2
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads

Optimize a model with 3 rows, 2 columns and 6 nonzeros
Model fingerprint: 0xd942f567
Coefficient statistics:
  Matrix range     [4e+00, 5e+01]
  Objective range  [6e+01, 6e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [4e+01, 2e+02]
Presolve time: 0.00s
Presolved: 3 rows, 2 columns, 6 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.4625000e+31   3.609375e+30   1.462500e+01      0s
       2    4.9500000e+02   0.000000e+00   0.000000e+00      0s

Solved in 2 iterations and 0.01 seconds (0.00 work units)
Optimal objective  4.950000000e+02
```

```
print("Optimal Objective function Value", model.objVal)
```

```
Optimal Objective function Value 495.0
```

```
for var in model.getVars():
    print(f"{var.varName}: {round(var.X, 3)}")  # print the optimal decision variable val
```

```
x[0]: 1.667
x[1]: 6.667
```

```
# see the model
model.display()
```

```
Maximize
  57.0 x[0] + 60.0 x[1]
Subject To
  manpower limit: 50.0 x[0] + 13.0 x[1] <= 200
  backhoe limit: 4.0 x[0] + 5.0 x[1] <= 40
  truck limit: 8.0 x[0] + 4.0 x[1] <= 40
```

```
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_8432\17538946.py:2: DeprecationWarni
  model.display()
```

Now that you have seen how to model and solve a simple problem in gurobi, let's try to make some changes in the problem. Imagine in the meantime the company has purchased another truck and has hired an extra (full time) worker to increase its profit. So you have to model and solve a very similar problem with minor changes to reflect the new employee and the new truck.

You need to make changes when defining attributes.

**Solution**

The correct change is:

```
men_equipment, work_hour = gurobipy.multidict({'men' :[240], 'truck' :[80], 'bac
```

Which leads to an Optimal Objective Value of 510.91 with x[0] equal to 3.434 and x[1] equal to 5.253.

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Augmented form of a mathematical program

## Contents

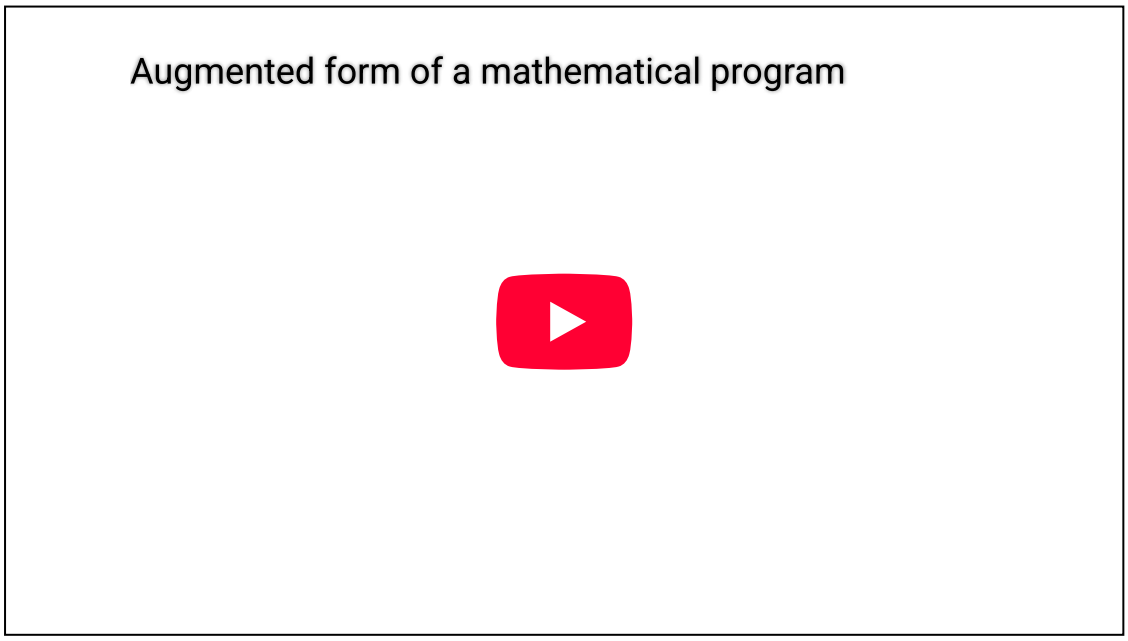

- Video
- Augmented form of a LP problem
- Example: Getting the augmented form of an LP problem


In the augmented form of a mathematical program:

- All the constraints must be equations (equality sign)
- The independent term must always be positive
- All the variables must be positive defined
- The objective function can be of type maximization or minimization

## Video

The story is told in a video. The video has a one-to-one correspondence with this book.



# Augmented form of a LP problem

Let us take a look, considering an objective function given by:

$$\text{Max (Min) } Z = c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

subject to:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\ x_1, x_2, \ldots, x_n \geq 0 \\ b_1, b_2, \ldots, b_m \geq 0 \end{cases}$$

As you can see, all the constraints here presented are equations, all the decision variables $x_i$ are positive and all the independent coefficients $b_i$ are positive as well. Using a more condensed mathematical notation:

$$\text{Max (Min) } Z = \sum_j c_j x_j$$

such that:

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \ (i = 1, 2, \ldots, m)$$

and with $x_j \geq 0 \ (j = 1, 2, \ldots, n)$ and $b_i \geq 0 \ (i = 1, 2, \ldots, m)$. $n$ here is being used to represent the number of variables and $m$ for the number of constraints.

# But what if you have an inequality as a constraint?

To solve those kind of situations you introduce a **slack variable**, let us say $s_1$, that will convert your constraint into an equality:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \rightarrow \underbrace{a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n}_{\text{consumption of a resource}} \quad + \quad \overbrace{s_1}^{\text{available resources whic}}$$

If, on the other hand, instead of $\leq$ we have an inequality dictated by $\geq$, then we write:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \geq b_1 \rightarrow \underbrace{a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n}_{\text{expression of consumption}} \quad \text{resources consumed ab} \quad -$$

# But what if we have negative independent coefficients?

Transform the negative independent coefficients into positive ones:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = -b_1 \rightarrow -a_{11}x_1 - a_{12}x_2 - \ldots - a_{1n}x_n = b_1$$

# And what if we have variables that can take negative values?

$$x_2 \leq 0 \rightarrow x_2 = -x_2'(x_2' \geq 0)$$

And our equation could be written as:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \rightarrow a_{11}x_1 - a_{12}x_2' + \ldots + a_{1n}x_n \leq b_1$$

> **Note**
>
> When one solves the problem, one must not forget to get the original value of $x_2$ in the end by multiplying $x_2'$ by $-1$

# And, finally, what if we have continuous variables defined in ALL the domain?

$$x_2 \in \mathbb{R} \rightarrow x_2 = (x_2' - x_2'')$$

and, in this case, both $x_2'$ and $x_2''$ will be $\geq 0$. Replacing in our equation would give:

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \leq b_1 \rightarrow a_{11}x_1 + a_{12}x_2' - a_{12}x_2'' + \ldots + a_{1n}x_n \leq b_1$$

> **Note**
>
> When one solves the problem, one must not forget to get the original value of $x_2$ in the end by subtracting $x_2''$ to $x_2'$

# Example: Getting the augmented form of an LP problem

Consider the optimization problem defined by:

$$\text{Max } Z = 2x_1 + 6x_2 + 13x_3 + 7x_4$$

such that:

$$\begin{cases} 3x_1 + 4x_2 + 12x_3 \geq 300 \\ 8x_1 + 6x_2 - x_4 \leq 220 \\ 6x_1 + 5x_2 + 3x_3 \leq 150 \\ x_1, x_3, x_4 \geq 0 \\ x_2 \leq 0 \end{cases}$$

We have a problem with the last condition, as we should have $x_2 \geq 0$. Therefore, we introduce a new variable given by $x_2' = -x_2$, and then we write:

$$\text{Max } Z = 2x_1 - 6x_2' + 13x_3 + 7x_4$$

with:

$$\begin{cases} 3x_1 - 4x_2' + 12x_3 - s_1 = 300 \\ 8x_1 - 6x_2' - x_4 + s_2 = 220 \\ 6x_1 - 5x_2' + 3x_3 + s_3 = 150 \\ x_1, x_2', x_3, x_4 \geq 0 \\ s_1, s_2, s_3 \geq 0 \end{cases}$$

And the problem is transformed in its augmented form and ready to be used in the SIMPLEX method!

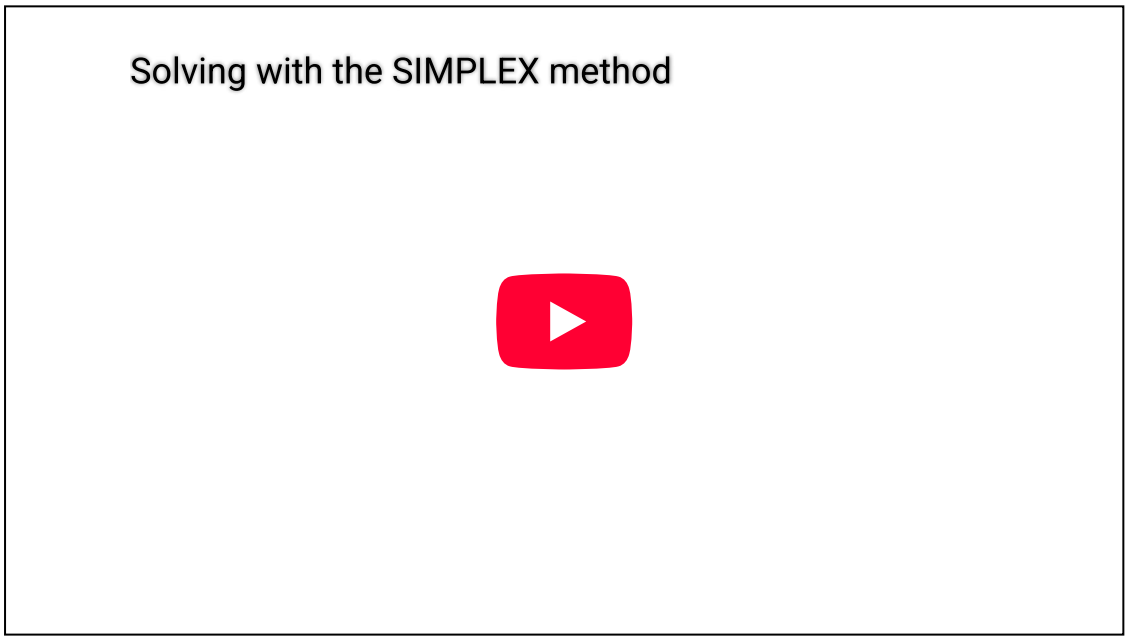
> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# SIMPLEX method

## Contents


- Augmented form
- Setting up the first table
- Optimality condition of the SIMPLEX method
- Summary


The story is told in a video. The video below has a one-to-one correspondence with this book:




**MUDE Exam Information**

For the exam, you are expected to have a clear understanding behind the Simplex method and how to do it. you will not need to do the calculations by hand in the exam or on Friday's assignment.

The SIMPLEX method has been created to automate the solving of the augmented form of a linear programming problem with continuous variables. Consider the following optimization problem:

$$\text{Max } L = 57x_1 + 60x_2$$

such that:

$$\begin{cases} 8x_1 + 4x_2 \leq 40 \\ 4x_1 + 5x_2 \leq 40 \\ 50x_1 + 13x_2 \leq 200 \\ x_1, x_2 \geq 0 \end{cases}$$

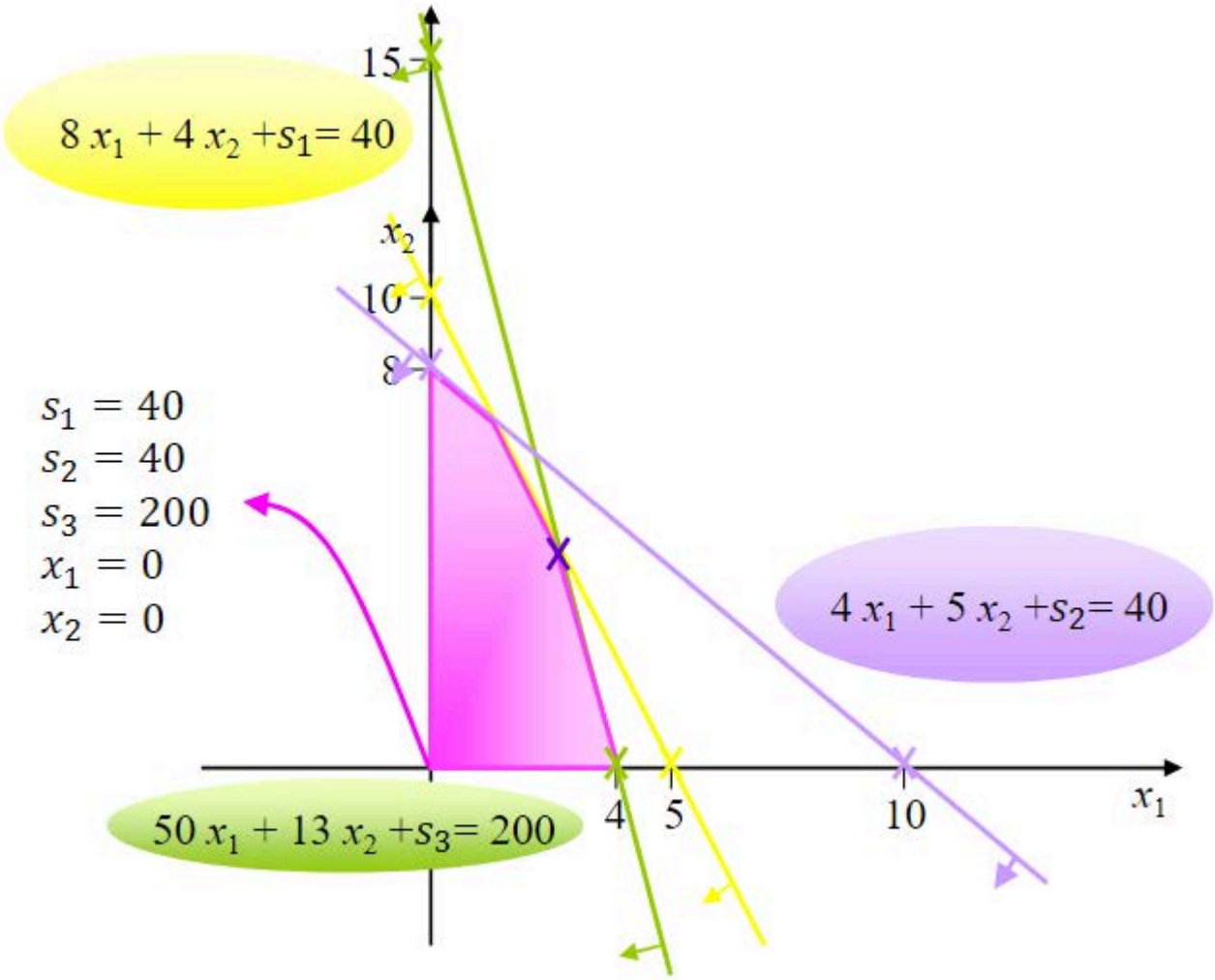
First, you need the problem in its augmented form, i.e., with the constraints given by:

$$\begin{cases} 8x_1 + 4x_2 + s_1 = 40 \\ 4x_1 + 5x_2 + s_2 = 40 \\ 50x_1 + 13x_2 + s_3 = 200 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{cases}$$

The feasible solution space of the problem is defined by all the constraints. We have 3 equations and 5 variables and, therefore, the solution space is not just one point. There are infinite points but just a finite number of feasible basic solutions (vertices).

A basic solution to this problem will have three variables different from zero (named basic variables), one for each constraint (ignore the domain constraints for this), and two that must be non-zero (non-basic). You will see how these match the vertices of the feasible region polygon.

From the three equations, one obvious apparent basic solution is $s_1 = 40$, $s_2 = 40$, $s_3 = 200$, with $x_1 = 0$ and $x_2 = 0$. A basic variable can be recognized in its contraint because it has a positive unitary coefficient in its constraint (+1) and it does not appear (has 0 coefficient) in the other constraints.

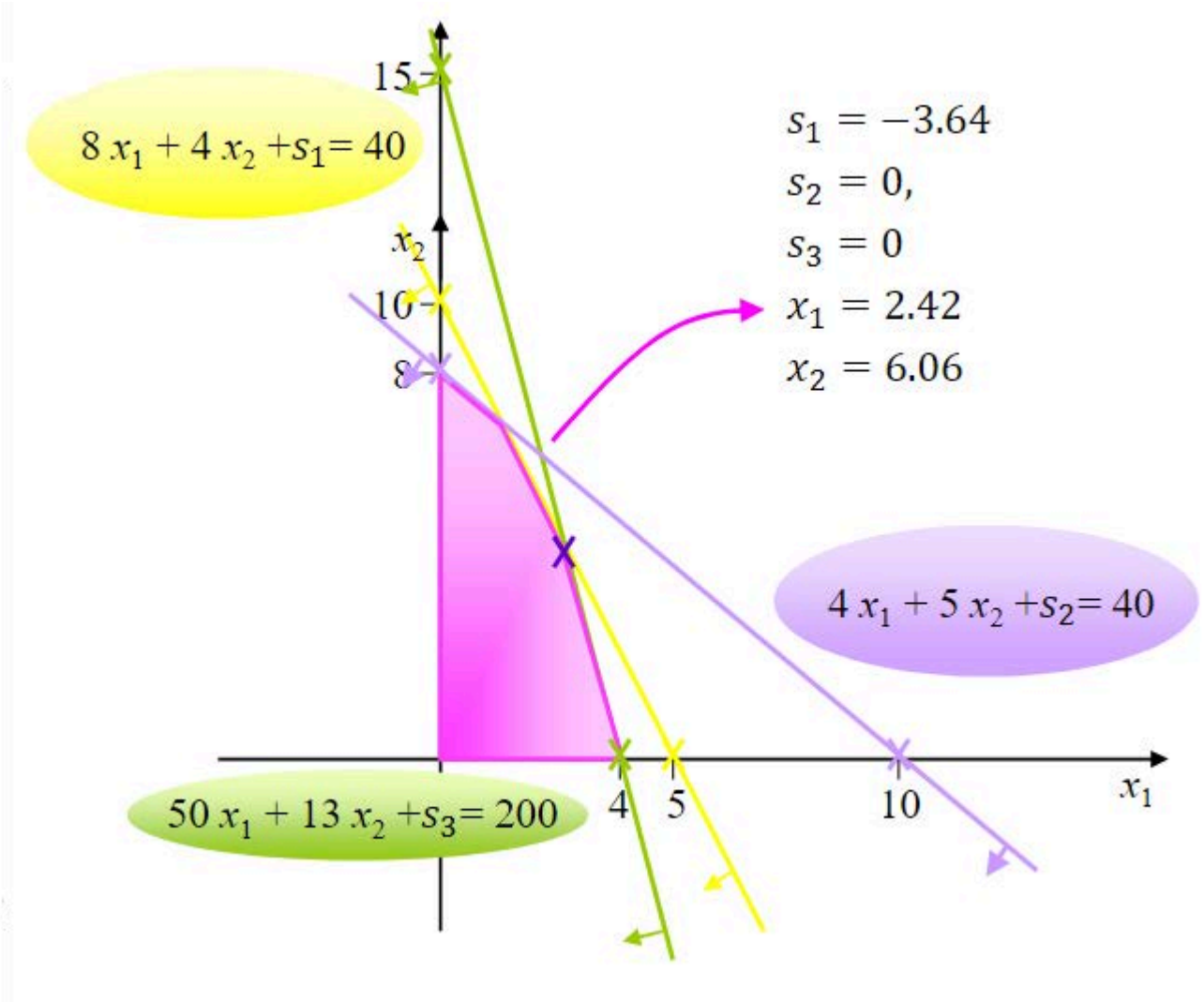



In order to search for the best solution, we should move from one system of equations to another that is equivalent to the first (it has the same solution space) but in which **another apparent basic solution comes up**.

There are two types of operations to transform the system of equations into another equivalent one:

- Multiply any equation by a positive or negative number
- Add to an equation another one which has been multiplied by any number

Graphically:

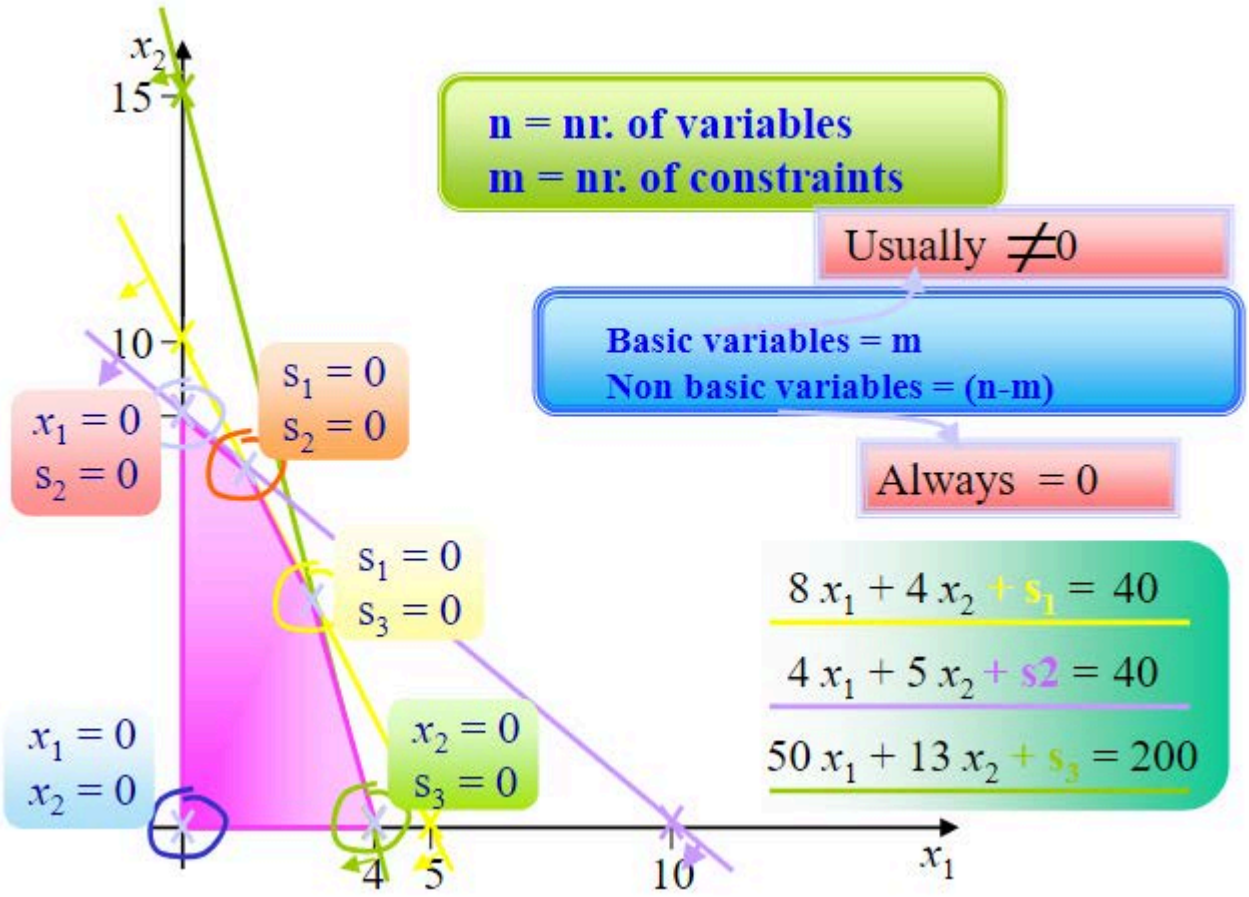



The problem is that even though this is a vertex, it is not a feasible solution - note that $s_1$ is negative!

How many basic solutions are there? We have **5 variables**, that divide into three basic variables ($x_1$, $x_2$, and $s_1$) and two non-basic ($s_2$ and $s_3$). The number of basic variables is always equal to the total number of equations (not including the domain constraints - $x_1 > 0$ - or the objective function). Therefore, the number of feasible and unfeasible basic solutions is given by:

$$C_n^m = \frac{n!}{m!(n-m)!}$$

where $n$ represents the number of variables and $m$ the total number of constraints. The feasible basic solutions are, therefore, given by:




We need a systematic method (algorithm) that applies changes to the system of equations going from feasible basic solution to feasible basic solutions without leaving the feasible space. This is the so-called **SIMPLEX method** (created by Dantzig in the 1950s).

---

# Augmented form

We start with the augmented form, given (as seen before) by:

$$\text{Max } L - 57x_1 - 60x_2 - 0s_1 - 0s_2 - 0s_3 = 0$$

subject to the following constraints:

$$\begin{cases} 8x_1 + 4x_2 + s_1 = 40 \\ 4x_1 + 5x_2 + s_2 = 40 \\ 50x_1 + 13x_2 + s_3 = 200 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{cases}$$

> **Note**
>
> The objective function must be put in the form of an equation with all the variables on the left-hand side and the independent coefficient on the right.

# Setting up the first table

| **Basic Variables** | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $-57$ | $-60$ | $0$ | $0$ | $0$ | $0$ |
| | | | | | | | |
| $s_1$ | $0$ | $8$ | $4$ | $1$ | $0$ | $0$ | $40$ |
| $s_2$ | $0$ | $4$ | $5$ | $0$ | $1$ | $0$ | $40$ |
| $s_3$ | $0$ | $50$ | $13$ | $0$ | $0$ | $1$ | $200$ |

Our current solution right now sets the basic variables $[s_1, s_2, s_3] = [40, 40, 200]$, and the non-basic variables $[x_1, x_2] = [0, 0]$. The value of the objective function in this case is $L = 0$.

# Optimality condition of the SIMPLEX method

The **optimality condition** assures that it is not possible to find a better solution.

In a maximization problem, the current basic feasible solution is optimal if all the coefficients of the objective functions are positive. On the other hand, in a minimization problem, the current solution is optimal if all the coefficients of the objective function are negative.

| Basic Variables | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $-57$ (negative) | $-60$ (negative) | $0$ | $0$ | $0$ | $0$ |
| | | | | | | | |
| $s_1$ | $0$ | $8$ | $4$ | $1$ | $0$ | $0$ | $40$ |
| $s_2$ | $0$ | $4$ | $5$ | $0$ | $1$ | $0$ | $40$ |
| $s_3$ | $0$ | $50$ | $13$ | $0$ | $0$ | $1$ | $200$ |

So it's not optimal

**How to go to the next table of the SIMPLEX method?**

- In a maximization problem, the variable to enter the basis (to become basic) is the one that has the most negative coefficient in the objective funcion
- In a minimization problem, the variable to enter the basis (to become basic) is the one that has the most positive coefficient in the objective function

Therefore, in our problem it will be $x_2$.

**And how to choose the variable to leave the basis?**

The variable to leave the basis is the one whose quotient between the independent coefficient of the constraint (meaning the same line) under the variable that will enter the basis is the lowest positive number including zero (this does not depend on being a maximization or minimization problem at all).

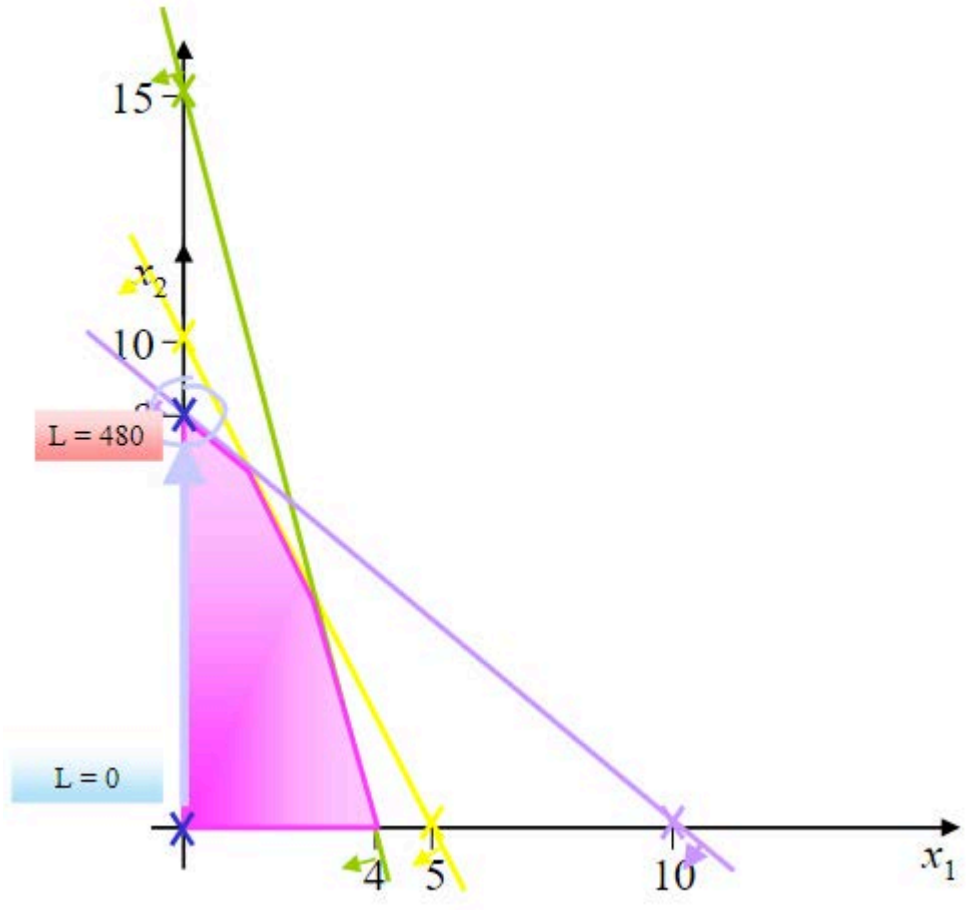
> **Note**
>
> Two special cases exist often:
>
> - if the coefficient of the entering basic variable is zero: enter *no limit* in the minimum ratio test column. That is not a candidate.
> - if the coefficient of the entering basic variable is negative: enter *no limit* in the minimum ratio test column. This is not a candidate.

| **Basic Variables** | $L$ | $x_1$ | $x_2$ **(variable to enter)** | $s_1$ | $s_2$ | $s_3$ | $b$ | | $\Delta$ |
|---|---|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $-57$ | $-60$ (most negative coefficient) | $0$ | $0$ | $0$ | $0$ | | |
| | | | | | | | | | |
| $s_1$ | $0$ | $8$ | $4$ | $1$ | $0$ | $0$ | $40$ | | $10$ |
| $s_2$ (variable to leave) | $0$ | $4$ | $5$ (pivot) | $0$ | $1$ | $0$ | $40$ | | $8$ (lowest positive number) |
| $s_3$ | $0$ | $50$ | $13$ | $0$ | $0$ | $1$ | $200$ | | $15.4$ |

Changing the system, our table will start looking like:

| **Basic Variables** | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $-9$ | $0$ | $0$ | $12$ | $0$ | $480$ |
| | | | | | | | |
| $s_1$ | $0$ | $4.8$ | $0$ | $1$ | $-0.8$ | $0$ | $8$ |
| $x_2$ | $0$ | $0.8$ | $1$ (pivot) | $0$ | $0.2$ | $0$ | $8$ |
| $s_3$ | $0$ | $39.6$ | $0$ | $0$ | $-2.6$ | $1$ | $96$ |

And our current solution has $[s_1, x_2, s_3] = [8, 8, 96]$ as basic variables and $[x_1, s_2] = [0, 0]$ as non-basic variables. With this scenario, the objective function is now $L = 480$. Graphically speaking:




**But is it already the optimal solution?**

The next step will be to check if the solution obtained is the optimal solution and move to the next basic solution if it is not:

| **Basic Variables** | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | 1 | $-9$ (negative) | 0 | 0 | 12 | 0 | 480 |
| | | | | | | | |
| $s_1$ | 0 | 4.8 | 0 | 1 | $-0.8$ | 0 | 8 |
| $x_2$ | 0 | 0.8 | 1 | 0 | 0.2 | 0 | 8 |
| $s_3$ | 0 | 39.6 | 0 | 0 | $-2.6$ | 1 | 96 |

So it’s not optimal

Repeating the same procedure again to identify the variables to enter and to leave:

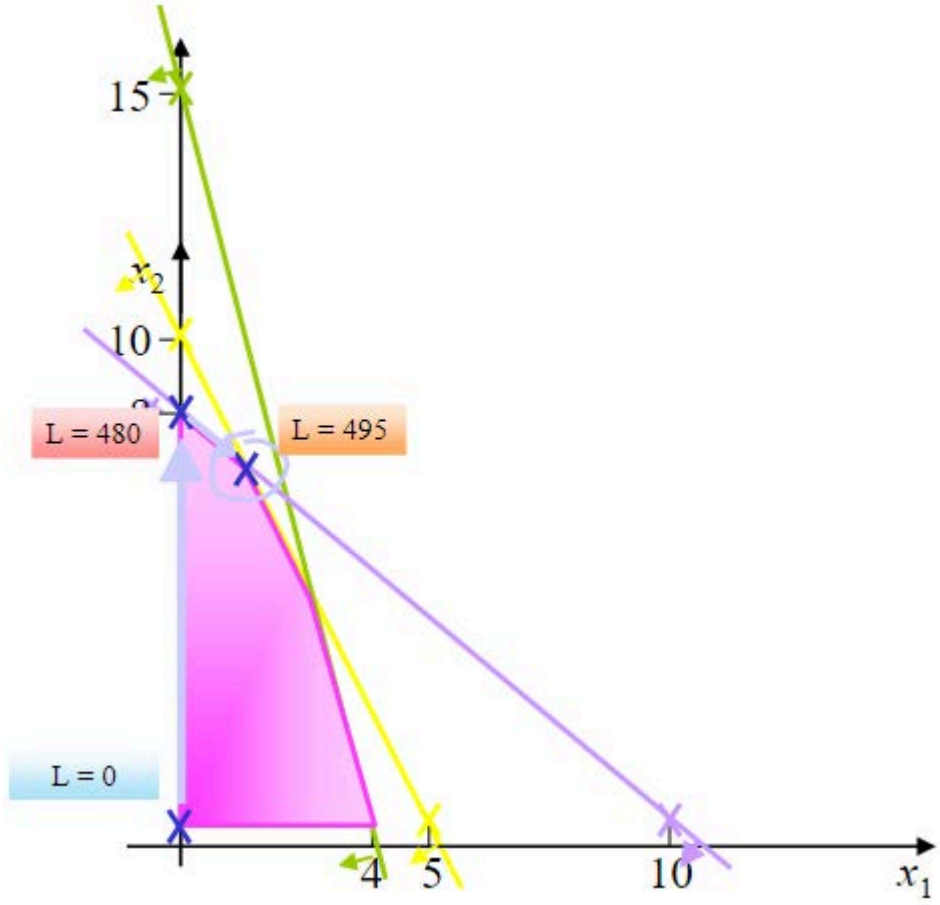
| **Basic Variables** | $L$ | $x_1$ **(variable to enter)** | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ | | $\Delta$ |
|---|---|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $-9$ (most negative coefficient) | $0$ | $0$ | $12$ | $0$ | $480$ | | |
| | | | | | | | | | |
| $s_1$ (variable to leave) | $0$ | $4.8$ (pivot) | $0$ | $1$ | $-0.8$ | $0$ | $8$ | | 1.67 (lowest positive number) |
| $x_2$ | $0$ | $0.8$ | $1$ | $0$ | $0.2$ | $0$ | $8$ | | 10 |
| $s_3$ | $0$ | $39.6$ | $0$ | $0$ | $-2.6$ | $1$ | $96$ | | 2.42 |

So $x_1$ is the variable to enter and $s_1$ the variable to leave.

Transforming the table again, we will obtain:

| **Basic Variables** | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | $1$ | $0$ | $0$ | $1.87$ | $10.5$ | $0$ | $495$ |
| | | | | | | | |
| $x_1$ | $0$ | $1$ (pivot) | $0$ | $0.21$ | $-0.17$ | $0$ | $1.67$ |
| $x_2$ | $0$ | $0$ | $1$ | $-0.17$ | $0.33$ | $0$ | $6.67$ |
| $s_3$ | $0$ | $0$ | $0$ | $-8.25$ | $4$ | $1$ | $30$ |

The current solution is now given by $L = 495$, which results from having $[x_1, x_2, s_3] = [1.67, 6.67, 30]$ as basic variables and $[s_1, s_2] = [0, 0]$ as non-basic variables. Graphically speaking:




Now we just need to check if the solution we have just obtained is optimal and, in this case, it is!

| **Basic Variables** | $L$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $L$ | 1 | 0 (positive) | 0 (positive) | 1.87 (positive) | 10.5 (positive) | 0 (positive) | 495 |
| | | | | | | | |
| $x_1$ | 0 | 1 | 0 | 0.21 | −0.17 | 0 | 1.67 |
| $x_2$ | 0 | 0 | 1 | −0.17 | 0.33 | 0 | 6.67 |
| $s_3$ | 0 | 0 | 0 | −8.25 | 4 | 1 | 30 |

So, this is the optimal solution

**Quiz questions**

| | Z | X | Y | $S_1$ | $S_2$ | $S_3$ | b |
|---|---|---|---|---|---|---|---|
| Z | 1 | -30 | -5 | 0 | 0 | 0 | |
| $S_1$ | 0 | 1 | 0 | 1 | 0 | 0 | 20 |
| $S_2$ | 0 | 0 | 10 | 0 | 1 | 0 | 50 |
| $S_3$ | 0 | 2 | 2 | 0 | 0 | 1 | 8 |

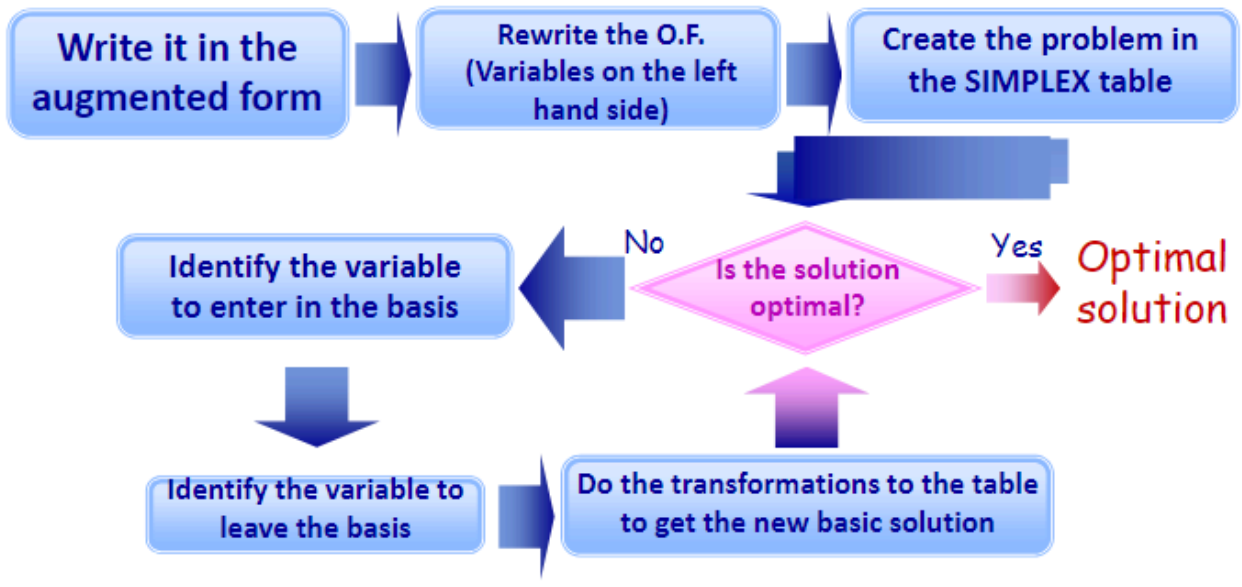
Consider the above table of the Simplex method to solve a maximization problem and select the right answer below.

- ○ The objective function is 1. The basic variables are S1=0, S2=0, S3=0. The non-basic variables are X=0, Y=0. The solution is the optimal solution.
- ○ The objective function is -30. The basic variables are S1=0, S2=0, S3=0. The non-basic variables are X=-30, Y=-5. The solution is the optimal solution.
- ○ The objective function is 0. The basic variables are S1=20, S2=50, S3=8. The non-basic variables are X=0, Y=0. The solution is **not** the optimal solution.
- ○ The objective function is 1. The basic variables are S1=1, S2=1, S3=1. The non-basic variables are X=2, Y=2. The solution is **not** the optimal solution.

Check

# Summary

The SIMPLEX workflow is shown in the scheme below:



> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Exercise 1: Simple Exercise

## Contents


- Task 1
- Task 2
- Task 3
- Task 4


**MUDE Exam Information**

For the exam, you are expected to have a clear understanding behind the Simplex method and how to do it. you will not need to do the calculations by hand in the exam or on Friday's assignment.

Consider the following problem:

$$\max(Z) = 2x_1 + 2x_2$$

such that

$$\begin{cases} 2x_1 + x_2 \leq 4 \\ x_1 + 2x_2 \leq 4 \\ x_1 + x_2 \leq 5 \\ x_1, x_2 \geq 0 \end{cases}$$

## Task 1

Transform the problem into the augmented form

**Solution**

$$Z - 2x_1 - 2x_2 - 0s_1 - 0s_2 - 0s_3 = 0$$

such that

$$\begin{cases} 2x_1 + x_2 + s_1 = 4 \\ x_1 + 2x_2 + s_2 = 4 \\ x_1 + x_2 + s_3 = 5 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{cases}$$

# Task 2

Build the first table of the simplex method.

| | $Z$ | $x_1$ | $x_2$ | $s_1$ | $s_2$ | $s_3$ | $b$ |
|---|---|---|---|---|---|---|---|
| $Z$ | | | | | | | |
| $S_1$ | | | | | | | |
| $S_2$ | | | | | | | |
| $S_3$ | | | | | | | |

# Task 3

Why is your final table the optimal solution?

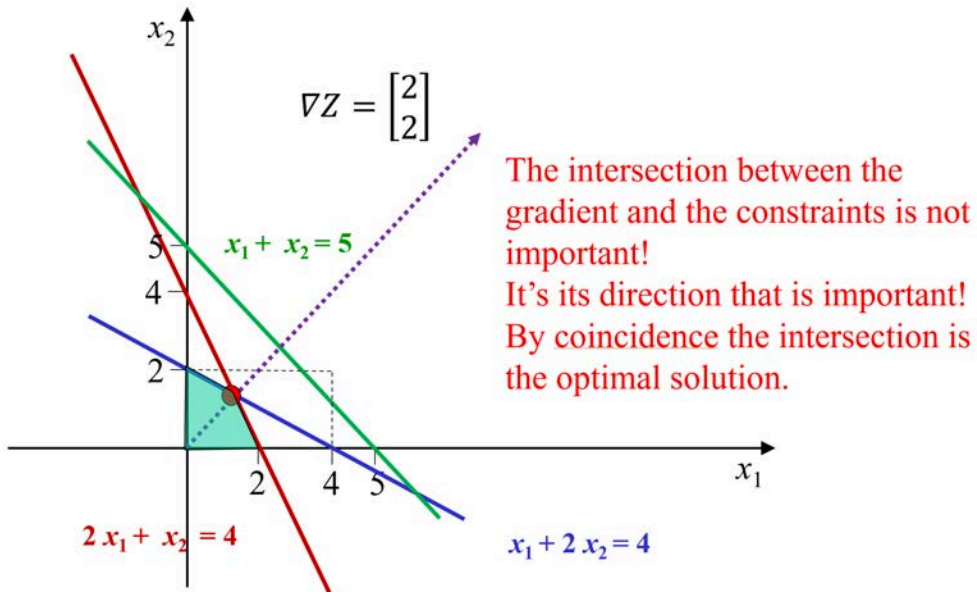
**Solution**

This is the optimal solution because all the coefficients are positive in the row of the objective function and this is a maximization problem.

# Task 4

Find the graphical solution of this problem. Does it give the same solution as using the Simplex method?

**Solution**




**Note:** in this problem and solution the intersection between the gradient and the constraints is not important (see figure), but rather the *direction* that is important! It is a coincidence that the intersection is the optimal solution.

**Attribution**

This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Exercise 2: Cargo airplane (Optional)

## Contents


- Original problem
- Challenges


> **MUDE Exam Information**
>
> For the exam, you are expected to have a clear understanding behind the Simplex method and how to do it. you will not need to do the calculations by hand in the exam or on Friday's assignment.

## Original problem

A cargo airplane has three compartments for storing freight: Front, central and rear. Each one of those compartments has the following limitations in weight and space:

| **Compartment** | **Weight** (ton) | **Volume** ($m^3$) |
|---|---|---|
| Front | 8 | 12000 |
| Central | 12 | 10000 |
| Rear | 14 | 6000 |

There are four cargo types, and the following maximum cargo (of each type) is available to be transported from a warehouse:

| Cargo | Maximum cargo available (ton) | Unit Volume ($\frac{m^3}{ton}$) | Unit profit ($\frac{€}{ton}$) |
|---|---|---|---|
| Type I | 18 | 180 | 590 |
| Type II | 14 | 250 | 580 |
| Type III | 20 | 300 | 580 |
| Type IV | 12 | 80 | 600 |

# Task 1

Build the mathematical programming model in order to define the amount of cargo of each type that should be transported and how it should be distributed in order to be efficient during the flight

## Solution

### Decision variables

| | Front | Center | Rear |
|---|---|---|---|
| Type I | $x_{11}$ | $x_{12}$ | $x_{13}$ |
| Type II | $x_{21}$ | $x_{22}$ | $x_{23}$ |
| Type III | $x_{31}$ | $x_{32}$ | $x_{33}$ |
| Type IV | $x_{41}$ | $x_{42}$ | $x_{43}$ |

$x_{ij}$: weight of cargo of type $i$ in position $i$ in the airplane with $i \in 1, 2, 3, 4$ and $j \in 1, 2, 3$ Note: you can use as unit of the variables the volume too but in that case the constraints are going to be slightly different

### Objective

If we define $P_i$ as the profit obtained for one ton of cargo of type $i \in 1, 2, 3, 4$ then the objective function is to maximize the profit:

$$\max(Z) = \sum_{i=1}^{4}\sum_{j=1}^{3} P_i x_{ij}$$

$$P_i = [590, 580, 580, 600]$$

### Constraints

Weight available to be transported at the warehouses:

$$\begin{aligned} x_{11} + x_{12} + x_{13} &\leq 18 \\ x_{21} + x_{22} + x_{23} &\leq 14 \\ x_{31} + x_{32} + x_{33} &\leq 20 \\ x_{41} + x_{42} + x_{43} &\leq 12 \end{aligned}$$

Weight limits on the compartments:

$$\begin{aligned} x_{11} + x_{21} + x_{31} + x_{41} &\leq 8 \\ x_{12} + x_{22} + x_{32} + x_{42} &\leq 12 \\ x_{13} + x_{23} + x_{33} + x_{43} &\leq 14 \end{aligned}$$

Volume limits on the compartments:

$$
\begin{aligned}
180x_{11} + 250x_{21} + 300x_{31} + 80x_{41} &\leq 12000 \\
180x_{12} + 250x_{22} + 300x_{32} + 80x_{42} &\leq 10000 \\
180x_{13} + 250x_{23} + 300x_{33} + 80x_{43} &\leq 6000
\end{aligned}
$$

**Variables domain**

$$x_{ij} \geq 0$$

# Task 2

Model the problem in Python using Gurobi.

**Solution**

The implementation is shown here.

# Task 3

What is the optimal profit?

€

# Task 4

What is the distribution of cargo for the optimal solution?

**Solution**

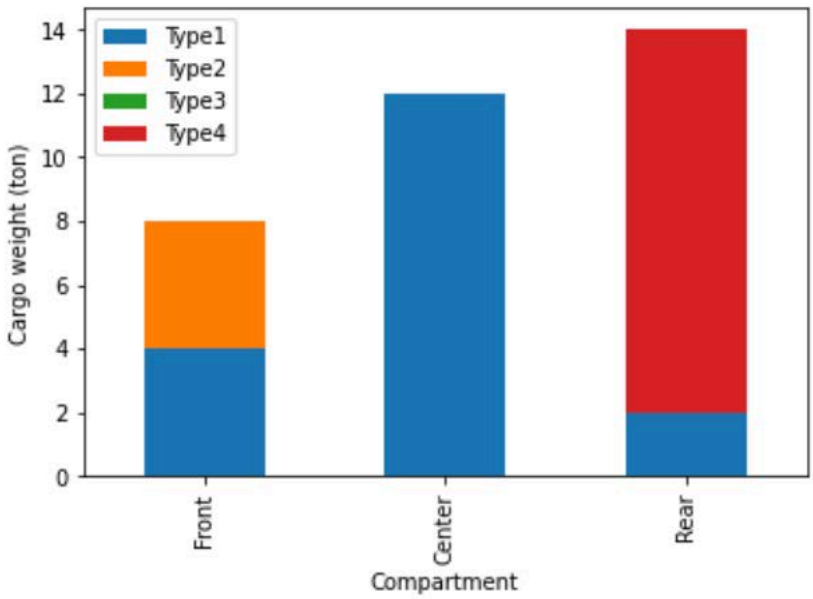



*Fig. 5.32* Distribution airplane problem

# Challenges

## Challenge 1

For efficiency purposes, the total weight transported in all compartments must be the same.

### Task 1

How do you change the model?

**Solution**

Additional constraints:

$$x_{11} + x_{21} + x_{31} + x_{41} = x_{12} + x_{22} + x_{32} + x_{42}$$
$$x_{12} + x_{22} + x_{32} + x_{42} = x_{13} + x_{23} + x_{33} + x_{43}$$

## Task 2

What is the optimal profit?

€

## Task 3

What is the distribution of cargo for the optimal solution?

**Solution**

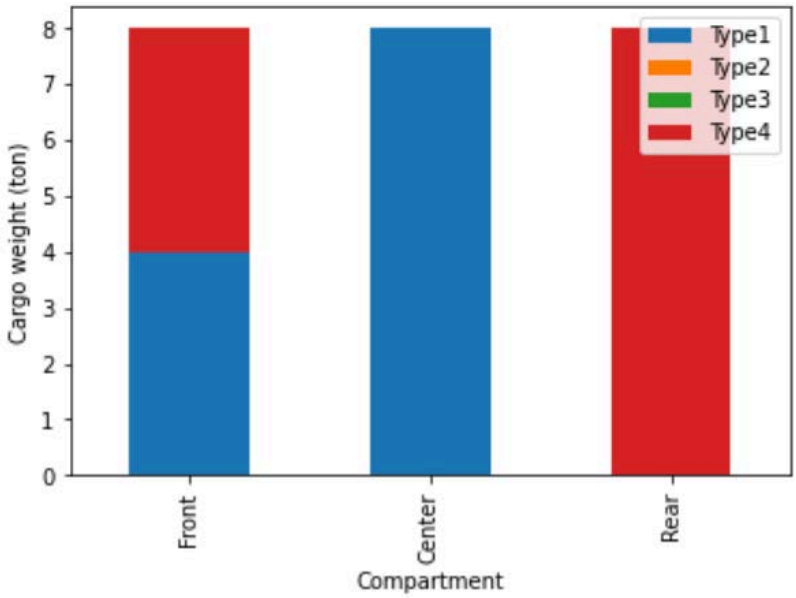



*Fig. 5.33* Distribution airplane problem challenge 1

# Challenge 2

The weight of type I must be double the Type II and III together.

## Task 1

How do you change the model?

**Solution**

Additional constraint:

$$x_{11} + x_{12} + x_{13} = 2\left(x_{21} + x_{22} + x_{23}\right) + 2\left(x_{31} + x_{32} + x_{33}\right)$$

## Task 2

What is the optimal profit? Round your answer to an integer value.

€

## Task 3

What is the distribution of cargo for the optimal solution?

**Solution**

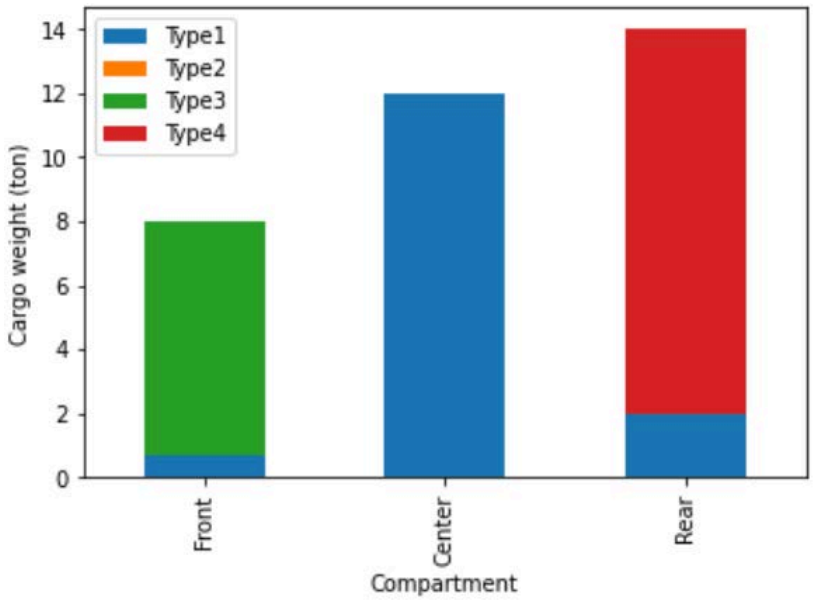



*Fig. 5.34* Distribution airplane problem challenge 2

# Challenge 3:

The weight of cargo in the frontal area must be at least double the cargo in the rear and central areas together.

## Task 1

How do you change the model?

**Solution**

Additional constraint:

$$x_{11} + x_{21} + x_{31} + x_{41} \geq 2\left(x_{12} + x_{22} + x_{32} + x_{42}\right) + 2\left(x_{13} + x_{23} + x_{33} + x_{43}\right)$$

## Task 2

What is the optimal profit?

€

## Task 3

What is the distribution of cargo for the optimal solution?

**Solution**

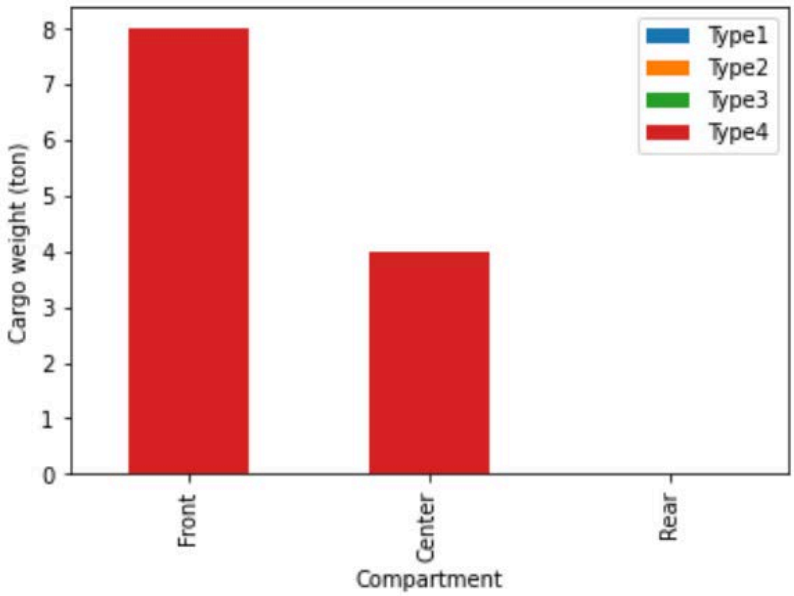



*Fig. 5.35* Distribution airplane problem challenge 3

**Attribution**

This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Cargo Airplane: Implementation in Python (Optional)

**Note**

Gurobi cannot be loaded in this online book, so download this notebook to work on it with Gurobi locally installed. Instruction on how to do that are in the `README.md` and `PA_2_4_A_gurobilicious.ipynb` of week 2.4.

**MUDE Exam Information**

For the exam, you are expected to have a clear understanding behind the Simplex method and how to do it. you will not need to do the calculations by hand in the exam or on Friday's assignment.

```
import gurobipy as gp
import numpy as np
import pandas as pd

model = gp.Model()
```

```
# cargo limits
cargo_type, cargo_weight, cargo_volume, cargo_profit = gp.multidict({
    'Type1': [18, 180, 590],
    'Type2': [14, 250, 580],
    'Type3': [20, 300, 580],
    'Type4': [12, 80,  600]
})

# compartment limits
compartment_type, compartment_weight, compartment_volume = gp. multidict({
    'Front': [8,  12000],
    'Center':[12, 10000],
    'Rear':  [14, 6000]
})
```

```
## decision variables
x = model.addVars(len(cargo_type),len(compartment_type), vtype = gp.GRB.CONTINUOUS, name
model.update()
```

```
# set objectives
objective = gp.quicksum(cargo_profit[cargo_type[i]] * x[i, j]
                                for j in range(len(compartment_type))
                                for i in range(len(cargo_type)))

model.setObjective(objective, gp.GRB.MAXIMIZE)
```

```
## constraints
# 1. cargo type weight limits
model.addConstrs((gp.quicksum(x[i, j] for j in range(len(compartment_type))) <= cargo_wei
                  for i in range(len(cargo_type))), name ='Max available cargo to carry')
```

```
# 2. compartment weight limits
model.addConstrs((gp.quicksum(x[i, j] for i in range(len(cargo_type))) <= compartment_wei
                  for j in range(len(compartment_type))), name ='Weight limits on the com
```

```
# 3. compartment volume limitations
model.addConstrs((gp.quicksum(x[i, j]*cargo_volume[cargo_type[i]] for i in range(len(carg
                  for j in range(len(compartment_type))), name ='Volume limits on the com
```

```
# Complete model
model.update()
model.display()
```

```
Maximize
590.0 x[0,0] + 590.0 x[0,1] + 590.0 x[0,2] + 580.0 x[1,0] + 580.0 x[1,1]
+ 580.0 x[1,2] + 580.0 x[2,0] + 580.0 x[2,1] + 580.0 x[2,2] + 600.0 x[3,0]
+ 600.0 x[3,1] + 600.0 x[3,2]
Subject To
  Max available cargo to carry[0]: x[0,0] + x[0,1] + x[0,2] <= 18
  Max available cargo to carry[1]: x[1,0] + x[1,1] + x[1,2] <= 14
  Max available cargo to carry[2]: x[2,0] + x[2,1] + x[2,2] <= 20
  Max available cargo to carry[3]: x[3,0] + x[3,1] + x[3,2] <= 12
  Weight limits on the compartments[0]: x[0,0] + x[1,0] + x[2,0] + x[3,0] <= 8
  Weight limits on the compartments[1]: x[0,1] + x[1,1] + x[2,1] + x[3,1] <= 12
  Weight limits on the compartments[2]: x[0,2] + x[1,2] + x[2,2] + x[3,2] <= 14
Volume limits on the compartments[0]: 180.0 x[0,0] + 250.0 x[1,0] + 300.0 x[2,0] + 80.0
 x[3,0] <= 12000
Volume limits on the compartments[1]: 180.0 x[0,1] + 250.0 x[1,1] + 300.0 x[2,1] + 80.0
 x[3,1] <= 10000
Volume limits on the compartments[2]: 180.0 x[0,2] + 250.0 x[1,2] + 300.0 x[2,2] + 80.0
 x[3,2] <= 6000
```

```
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7320\1502344741.py:3: DeprecationWar
  model.display()
```

```
# Solve
model.optimize()
```

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Integer problems and solving with Branch-and-Bound

## Contents


- Video
- Integer or binary variables definition

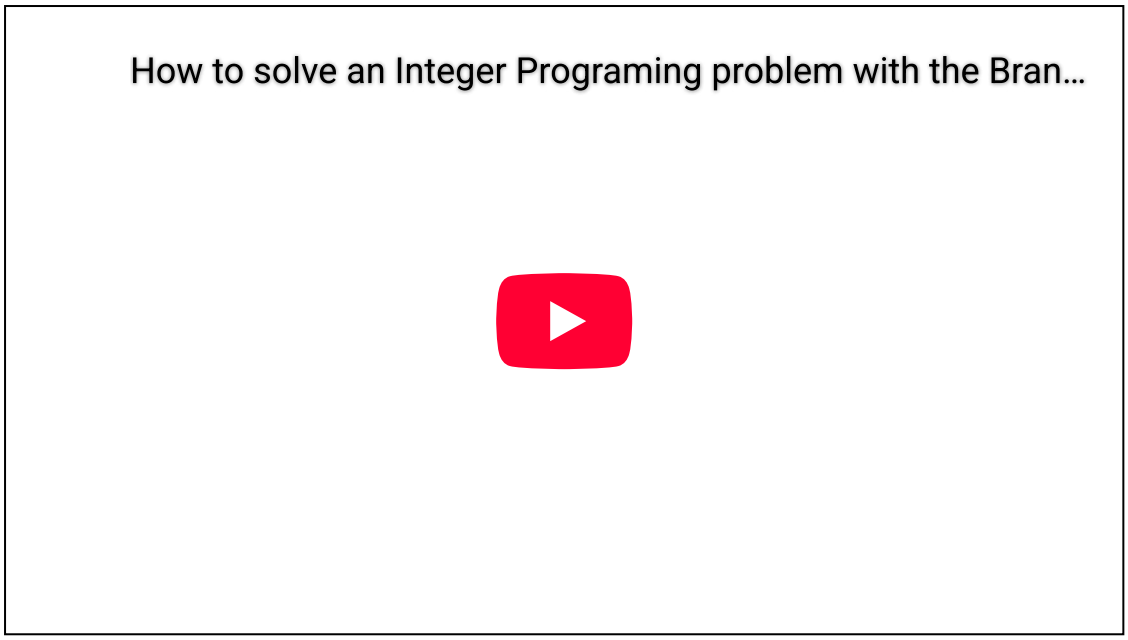

There are many linear programming problems in which all or some of the variables can only take integer values:

- the road network design problem in which a link can only exist or not exist
- location problem in which the variable tells if the facility exists in a certain place or not
- vehicle fleet to be allocated to a service
- number of drivers

These problems are called integer programming problems (IP) or mixed integer programming problems (MIP), if there are some continuous variables. Obtaining a solution for these problems can be quite difficult as the simplex method is not prepared for integer variables.

## Video

The story is told in a video. The video has a one-to-one correspondence with this book




# Integer or binary variables definition

$\mathbb{N}_0$ is the set of natural numbers meaning the integer ones and it includes in this case the zero. You can have:

$$\mathbb{N}^0 = \mathbb{N}_0 = \{0, 1, 2, \dots\}$$
$$\mathbb{N}^* = \mathbb{N}^+ = \mathbb{N}_1 = \mathbb{N}_{>0} = \{1, 2, \dots\}$$

Defining the variable:

$$x \in \mathbb{N}_0$$

for binary variables you define $x$ as:

$$x \in \{0, 1\}$$

Consider the following optimization problem:

$$\text{max } Z = 7x_1 + 9x_2$$

such that:

$$-x_1 + 3x_2 \leq 6$$
$$x_1 + x_2 \leq 8$$
$$x_1, x_2 \in \mathbb{N}_0$$

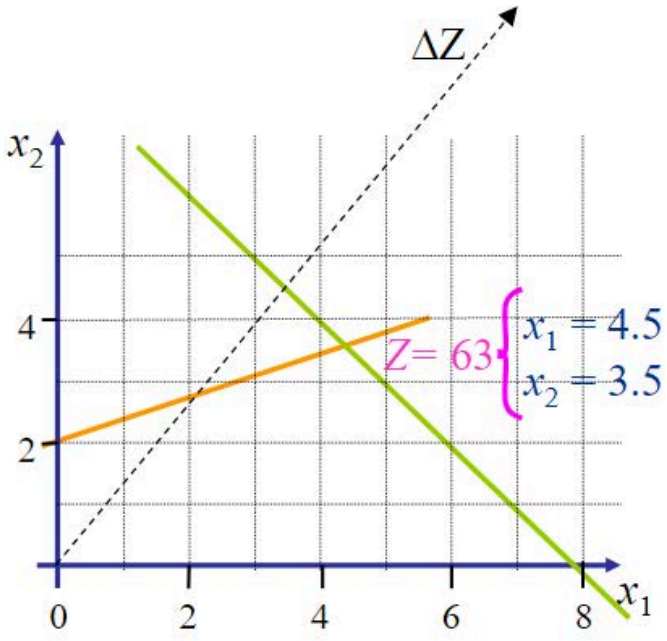
# So how can we solve this?

$$\text{Max } Z = 7x_1 + 9x_2$$

s. t.:

$$-x_1 + 3x_2 \leq 6$$
$$x_1 + x_2 \leq 8$$
$$x_1 , x_2 \in N_0$$

➔ Round the variables?




Rounding with two variables is not such a big problem and we can easily find a feasible integer solution. This solution, however, is not yet proven to be optimal.

---

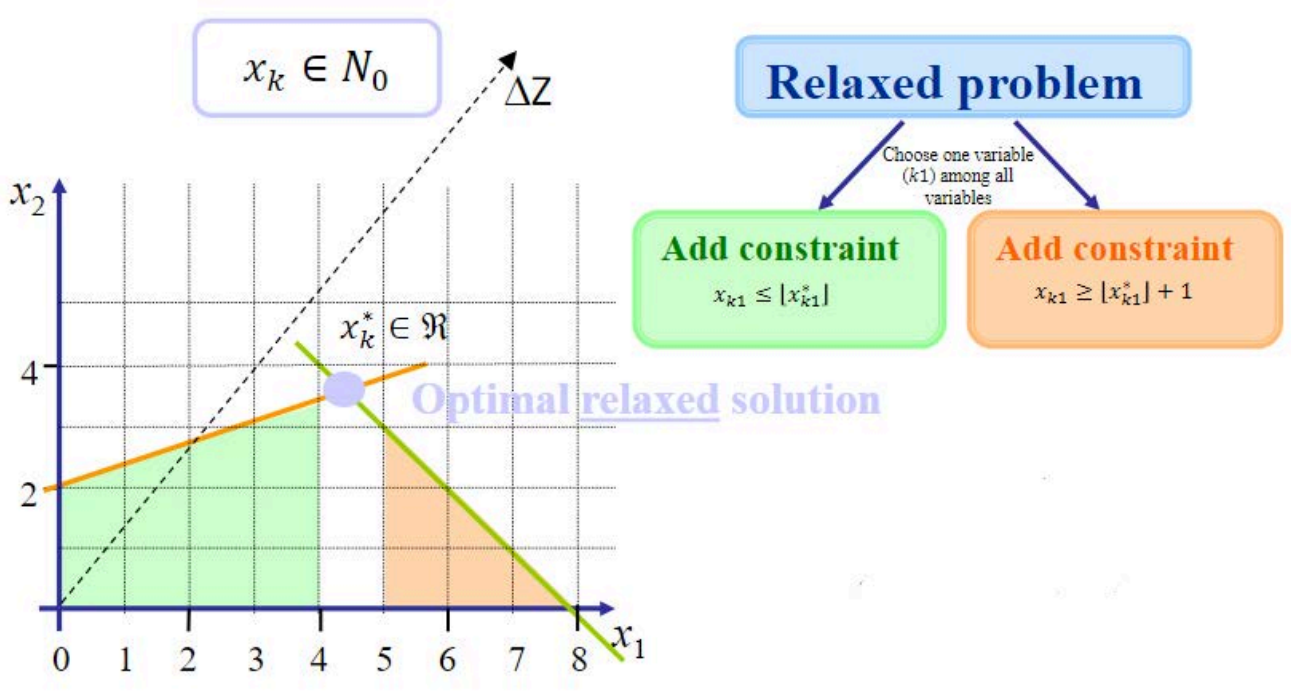
There are problems in which is not acceptable to round, most notably when the variables are binary (can only take the value 0 or 1). Rounding leads to almost always unfeasible solution, especially when you have lots of variables.

The best well-known method to solve integer programming problems is the **branch-and-bound** method. It is based on the idea of enumerating all the solution points of the feasible region, but methods are used so that a lower number of solutions can be tested.

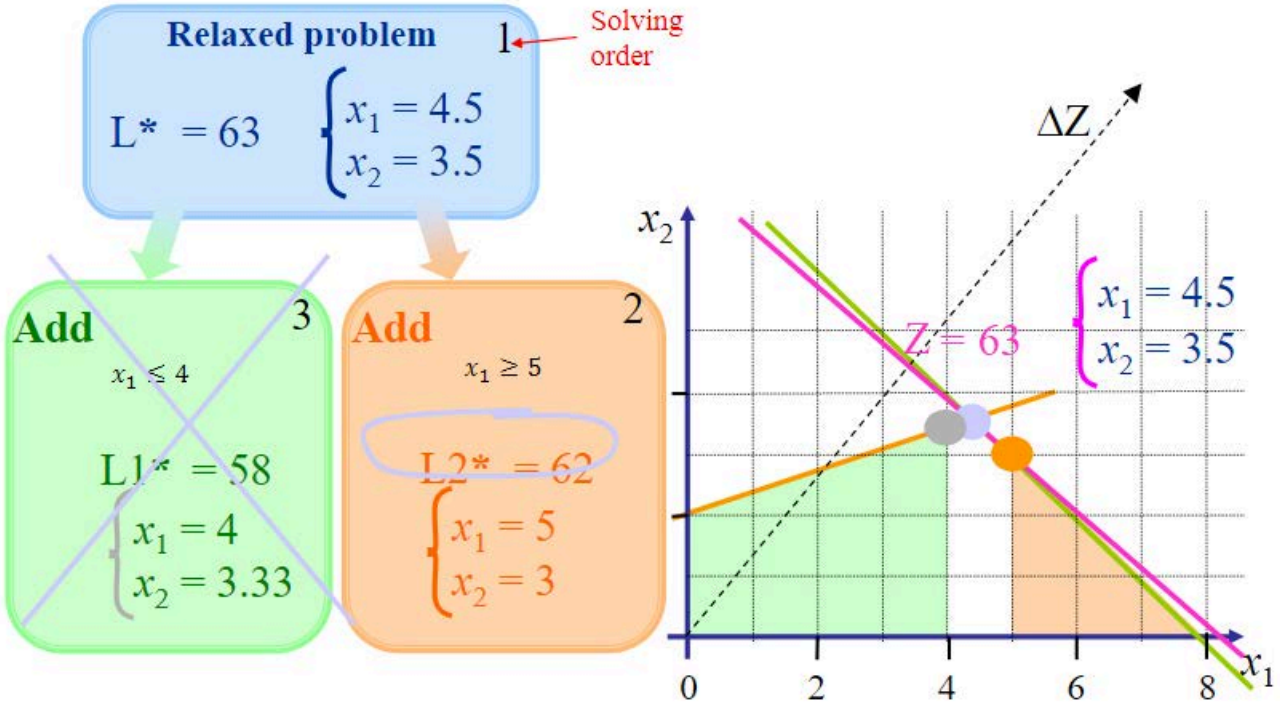
## Branch-and-bound

Solve the original **relaxed problem** (i.e. assuming the variables are all continuous positive) so you can obtain fractional values for the variables. Suppose that $x_k \in \mathbb{N}_0$, but in this case you would obtain for the relaxed problem $x_k^* \in \mathbb{R}$.

From this point onwards, the initial problem is decomposed… Adding new constraints to guarantee the integrality of the variables one by one.




Solve the two problems using the SIMPLEX method. For each solution of those two problems do the same step of dividing the problem but now with another variable. The solution search procedure ends when an integer solution is found ($x_k \in \mathbb{N}_0$) and no better objective function can be found.




The optimal solution is the one with $L2^* = 62$ since in the other branch the value of the objective function is smaller and with $L2^* = 62$, the solution is already an integer like we wanted.

---

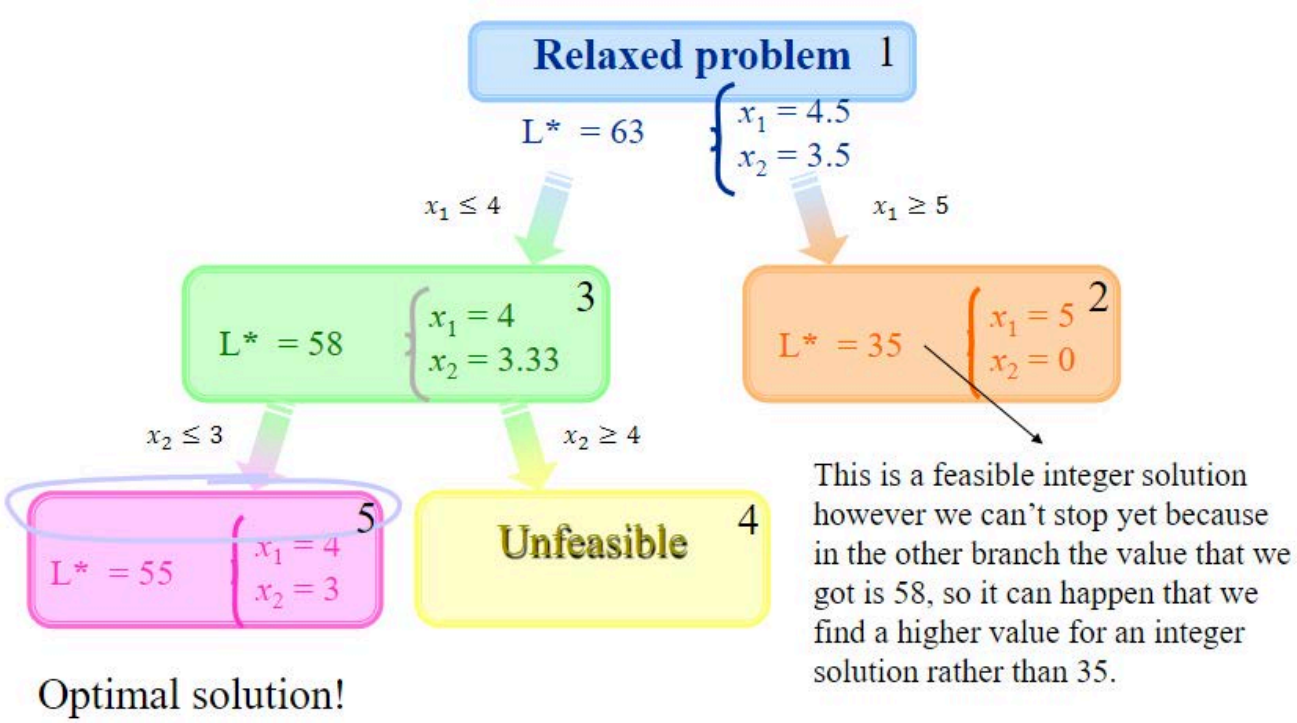
Let us take a look into another example. Consider the optimization problem given by:

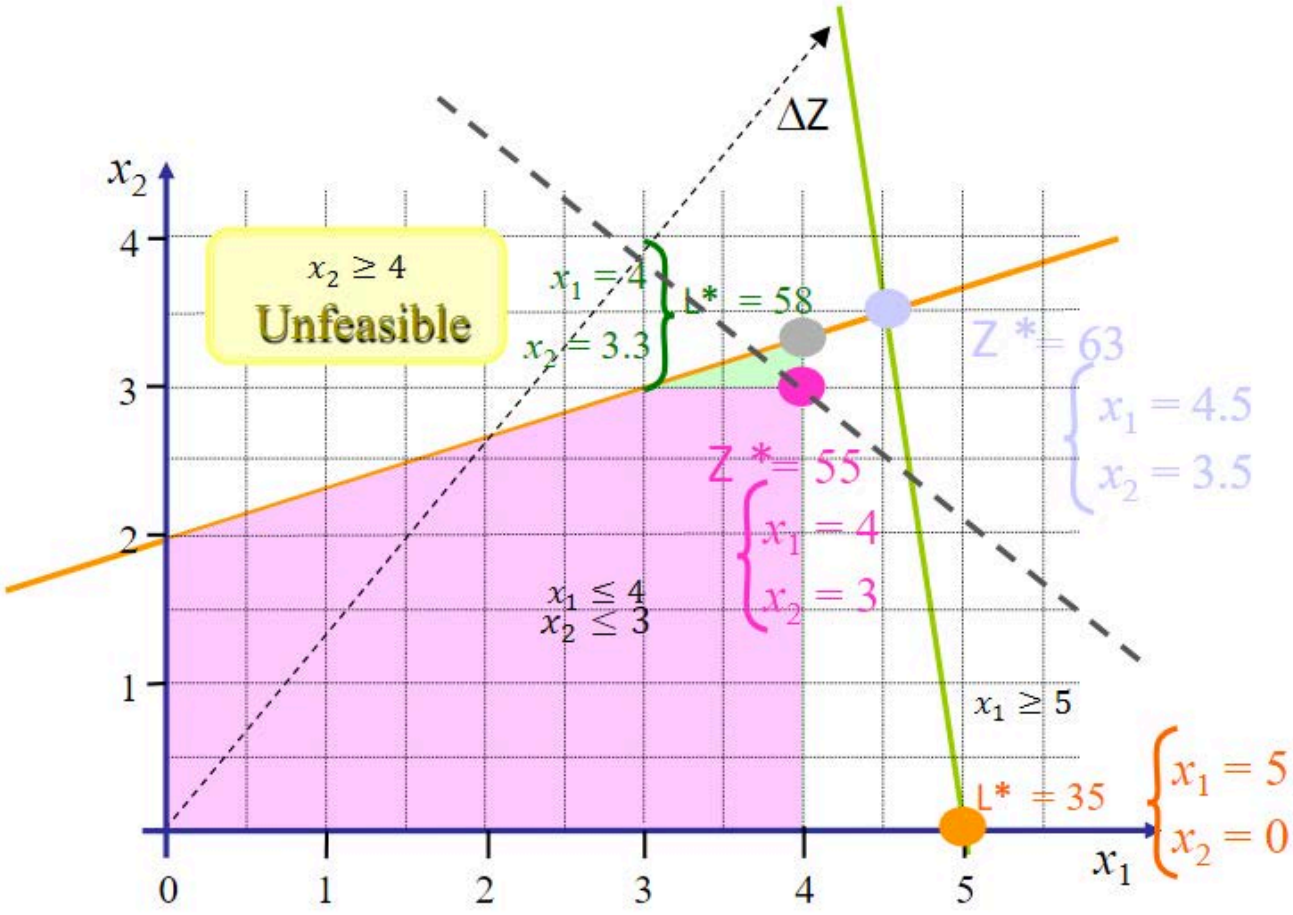
$$\max Z = 7x_1 + 9x_2$$

such that:

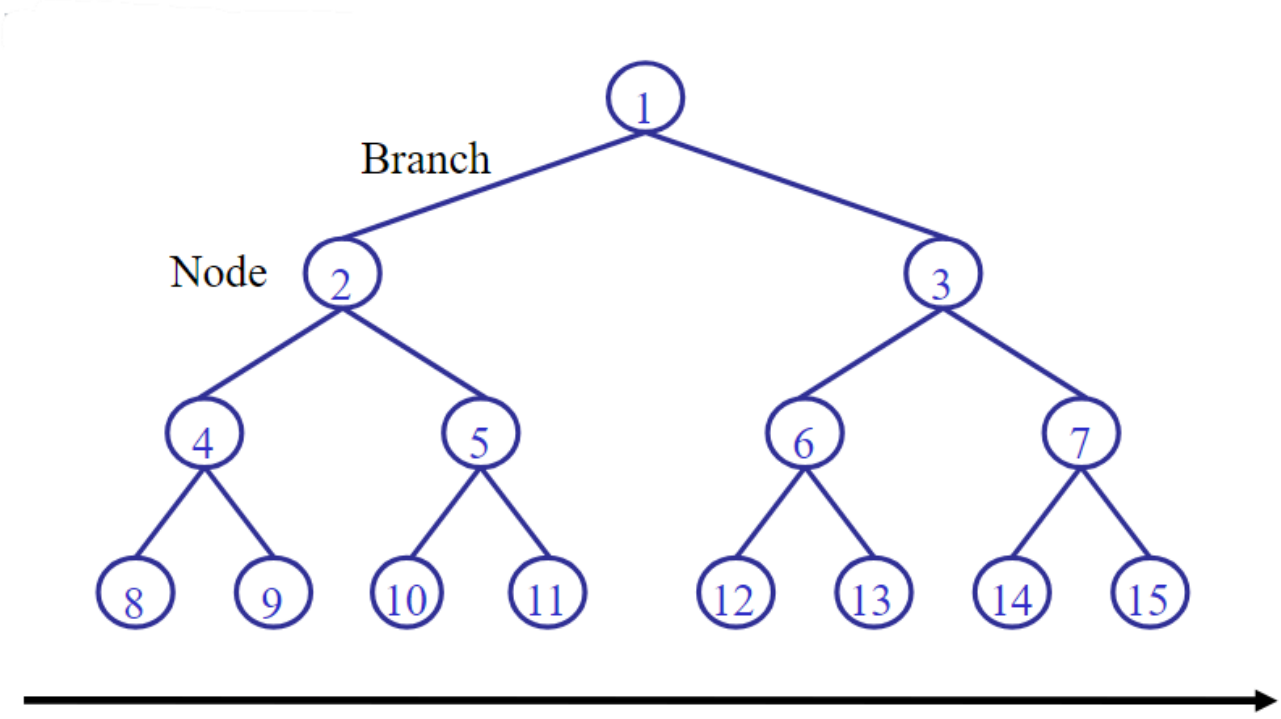
$$\begin{cases} -x_1 + 3x_2 \leq 6 \\ 7x_1 + x_2 \leq 35 \\ x_1, x_2 \in \mathbb{N}_0 \end{cases}$$

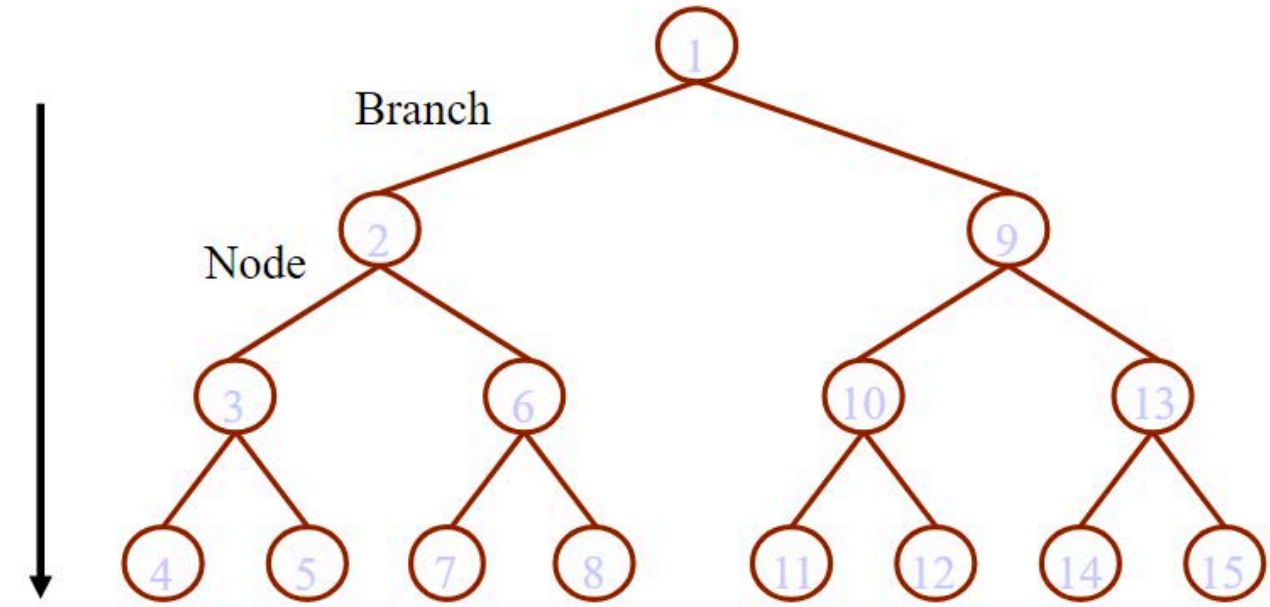
Let us take a look into the solving process using the branch-and-bound method:



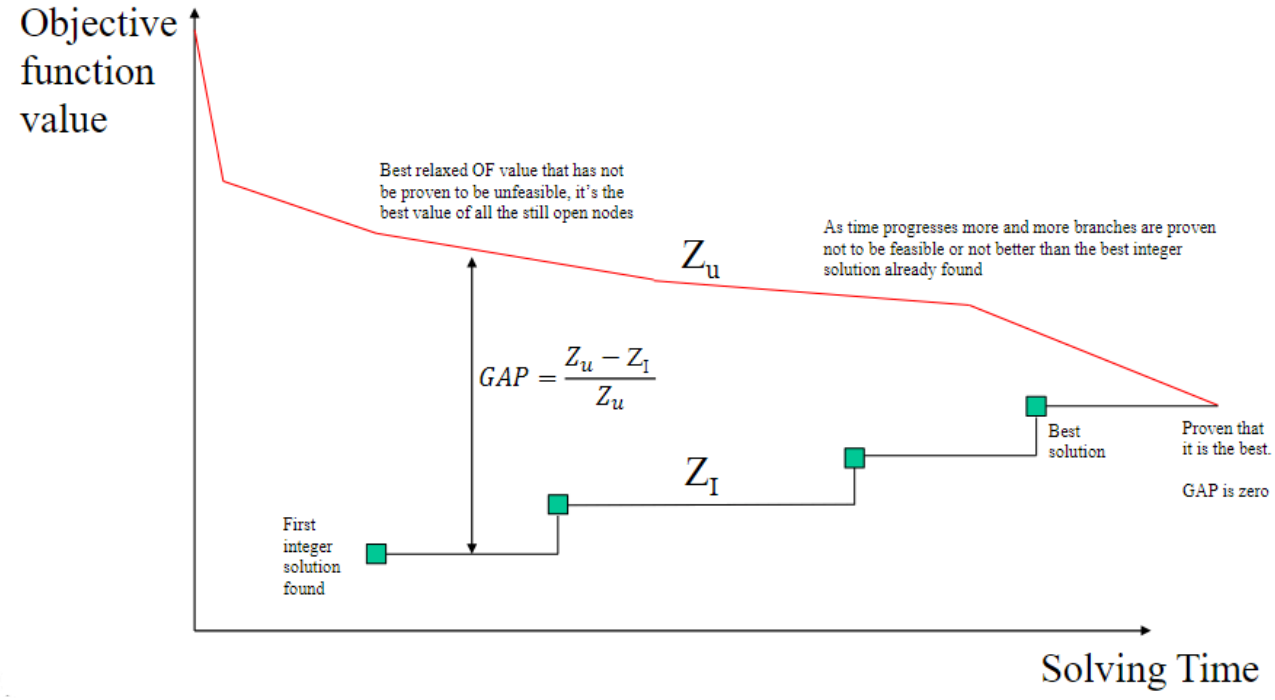


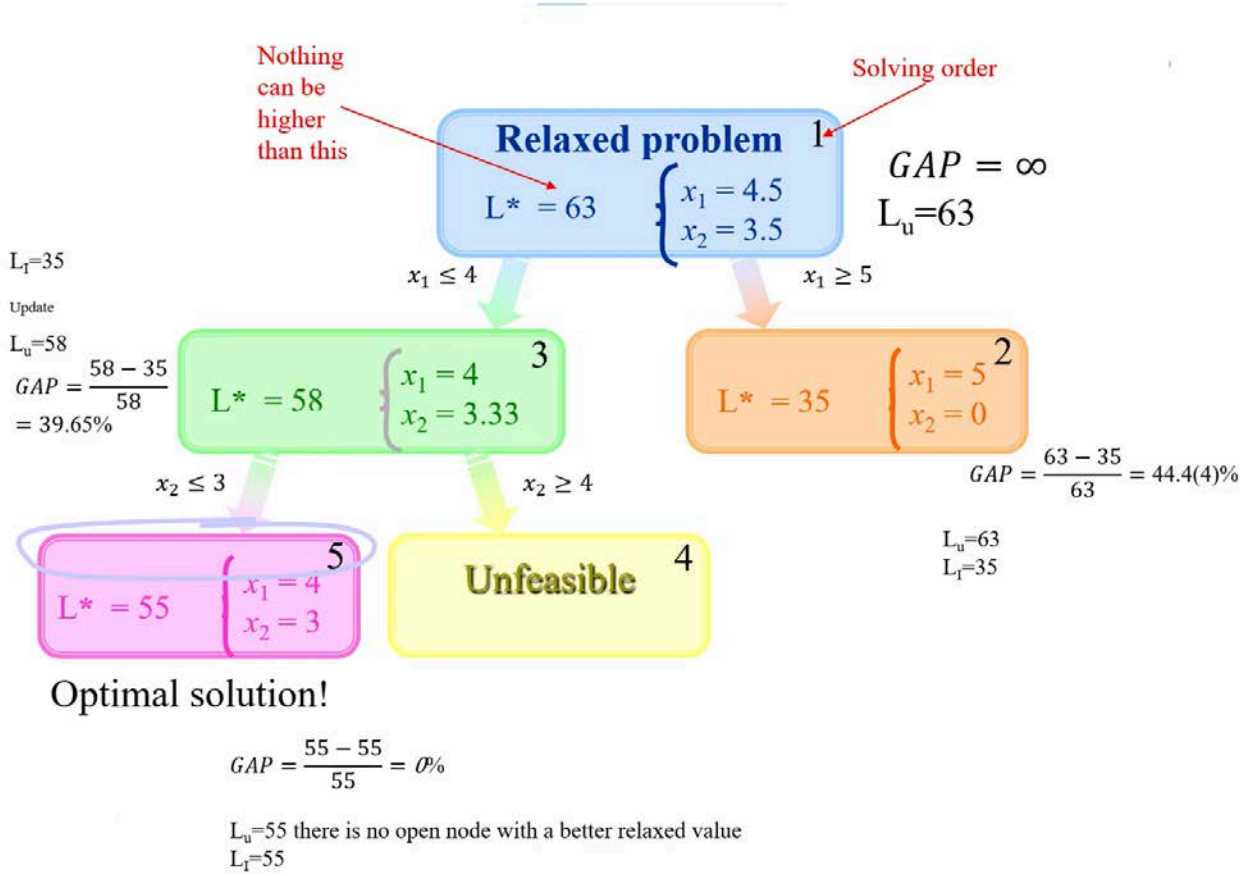

**Horizontal progression**






## Gap concept (example for maximization)






# Computational efficiency

- The efficiency of using the branch-and-bound for solving integer problems depends on obtaining a good $Z_I$ in the beginning (high value in the case of a maximization problem and low in the case of a minimization problem). Because this value will be used to close many nodes that have an OF worse than that integer solution
- This depends on the order by which the sub-problems are generated and solved, which in its turn depends on the variable to do the branching
- Unfortunately, there is no general procedure that allows determining the variable that allows this search to go faster

**Quiz questions**

What does the gap tell you?

- ○ That the optimal solution is better by that percentage (gap) than the current solution.
- ○ That the optimal solution will never be found. That's a pity.
- ○ That the optimal solution will be better than the current integer solution by at most that gap.

Check

**Attribution**

This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Some constraints that take advantage of integer/binary variables

## Contents


- Either-or constraints
- Discontinuous variables
- The product of two binary variables
- The product of a binary and a continuous variable


At most two of the projects 1, 2, 3, 4 and 5 can be done:

$$x_1 + x_2 + x_3 + x_4 + x_5 \leq 2$$

If project 1 is done then also project 2 is done:

$$x_2 \geq x_1$$

Restrict the variable $x_i$ to have just one of a given set of integers {2,3,5,7} $\rightarrow$ let $y_1$,$y_2$,$y_3$,$y_4$ be binary, and add the constraints:

$$\begin{cases} x_i = 2y_1 + 3y_2 + 5y_3 + 7y_4 \\ y_1 + y_2 + y_3 + y_4 = 1 \end{cases}$$

## Either-or constraints

Satisfy only one of these two constraints depending on something else in the model:

$$\begin{cases} 3x_1 + 5x_2 \leq 7 \\ 2x_1 + 2x_2 \leq 4 \end{cases}$$

Replace with the following two constraints:

$$3x_1 + 5x_2 \leq 7 + My$$
$$2x_1 + 2x_2 \leq 4 + M(1 - y)$$

where $y$ is a binary variable and M is sufficiently large. When $y$ is 1 the second constraint is active and the first is relaxed ($\leq$ big number), when $y$ is 0, the inverse happens.

If $x_1$ happens (it has the value 1) then $x_2$ cannot happen and vice-versa. Both can be zero as well meaning both will not happen, as $x_1 + x_2 \leq 1$.

# Discontinuous variables

If you want to impose that in some situations $x$ can only be defined as:

$$x = 0 \text{ or } l \leq x \leq u$$

Introduce the binary variable $y$ and add the following constraints:

$$x \leq uy$$
$$x \geq ly$$

In this way when $y$ is 0, $x$ is zero, and when it is 1 then $x$ is between $u$ and $l$.

# The product of two binary variables

As you know, in linear programming you cannot multiply two variables but this does not mean that you cannot use some constraints to produce that product. The product $x_1x_2$ of two binary variables $x_1$ and $x_2$ can be replaced by additional binary variable $y$ by adding the following constraints:

$$y \leq x_1$$
$$y \leq x_2$$
$$y \geq x_2 - M(1 - x_1)$$

with $M$ sufficiently large. The $y$ variable will be the result of multiplying $x_1$ for $x_2$.

# The product of a binary and a continuous variable

The product $x_1x_2$ of the binary variable $x_1$ and the continuous variable $x_2$ with $0 \leq x_2 \leq u$ can be replaced by an additional continuous variable $y$ by adding the following constraints:

$$\begin{cases} y \leq ux_1 \\ y \leq x_2 \\ y \geq x_2 - u(1 - x_1) \\ y \geq 0 \end{cases}$$

$y$ is obviously the continuous variable you want to get.

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Genetic Algorithm

## Contents

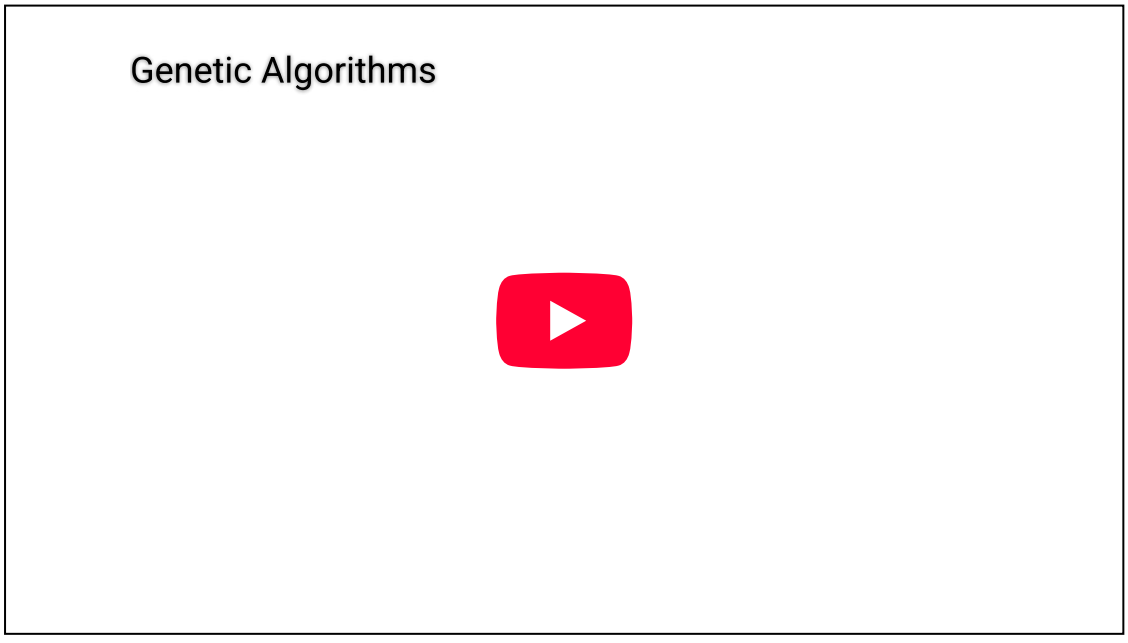

- Video
- Recap - Optimization Models & Exact optimization approaches
- (Meta-)Heuristic optimization approach
- Genetic algorithm
- Example


## Video

The story is told once in a video. The video below has a one-to-one correspondence with this book

**MUDE Exam Information**

For the exam, you are expected to have a clear understanding of the Genetic Algorithm described and its steps. You do not need to know how to implement it in code!



# Recap - Optimization Models & Exact optimization approaches

- Linear programming problems and integer programming problems
- Simplex and Branch and Bound

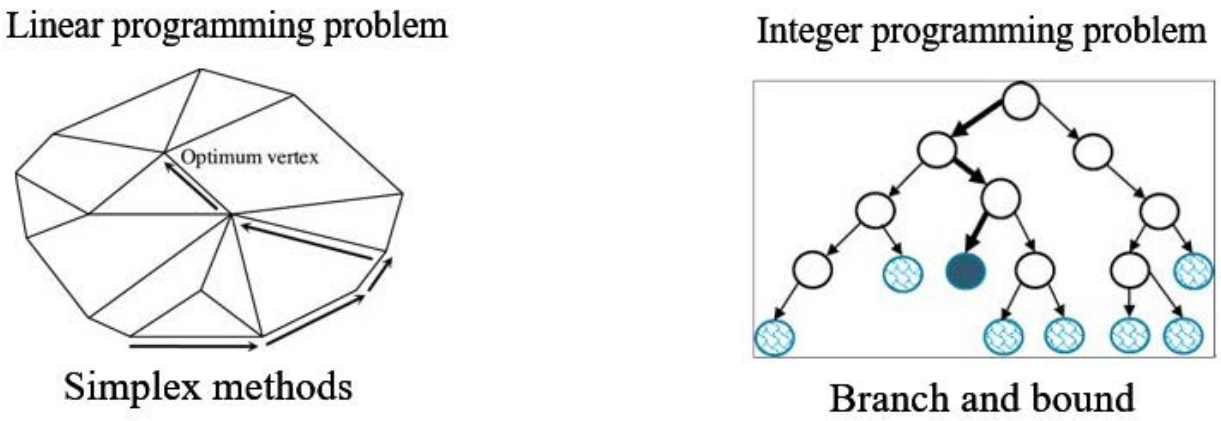



*Fig. 5.36* The figure on the left is from Daneshian (2025) and the figure on the right is from an unknown source. Find out more here.

Exact optimization approaches basically guarantees to find the optimal solution to a problem!

# Computation complexity

These methods can be time-consuming and computationally expensive, especially for large, complex problems. Many exact algorithms have exponential time complexity.

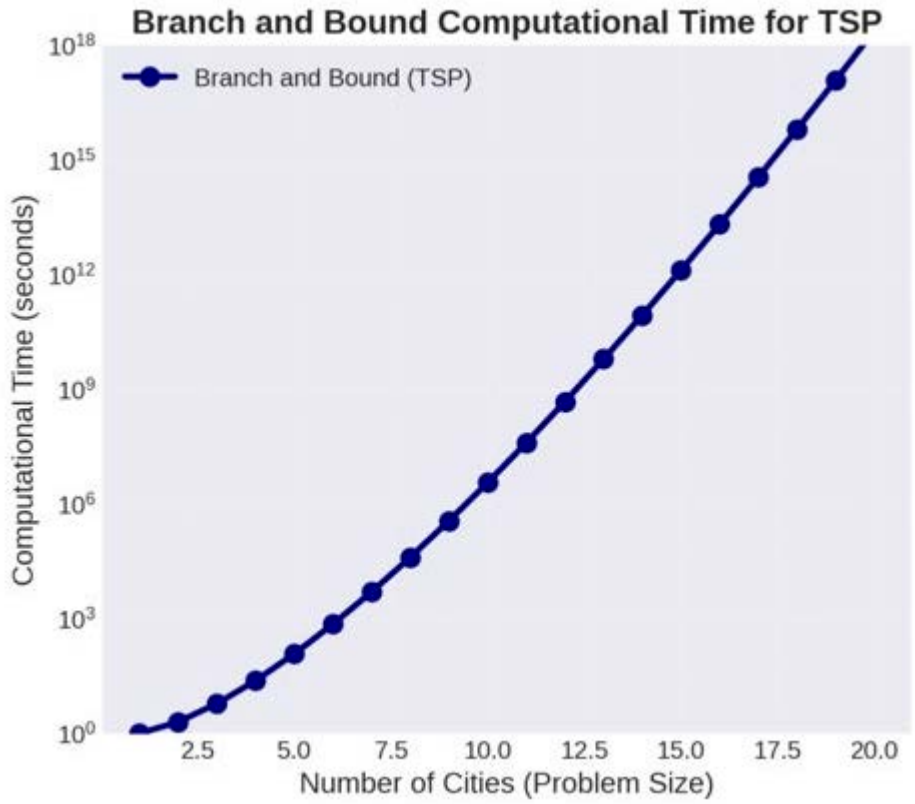



*Fig. 5.37* This figure is not included in the CC BY license of this book. Find out more here.

Numerous real-life optimization problems in science, engineering, economics, and business are complex and difficult to solve. They cannot be solved in an exact manner within a reasonable amount of time.

Consequently, the use of meta-heuristic algorithms, a form of approximate techniques, serves as the main alternative for solving such complex optimization problems.

# (Meta-)Heuristic optimization approach

A (meta-)heuristic approach is a procedure that determines near-optimal solutions to an optimization problem.

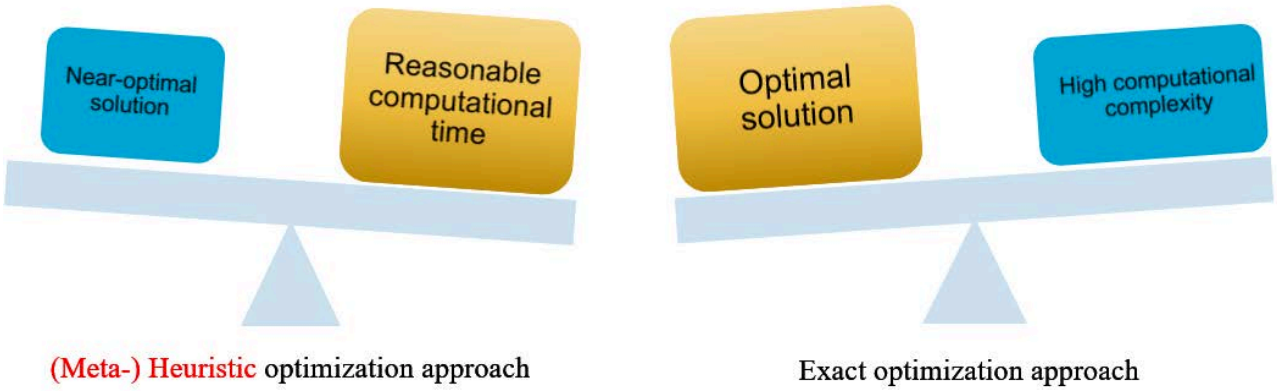



*Fig. 5.38* This figure is not included in the CC BY license of this book. Find out more here.

Metaheuristics represents a family of approximate optimization techniques. They provide “acceptable” solutions in a reasonable time for solving hard and complex problems in science and engineering. Generally, metaheuristics can be classified into Single-solution based meta-heuristics and Population based meta-heuristics. Evolutionary algorithm is a kind of population based meta-heuristics.

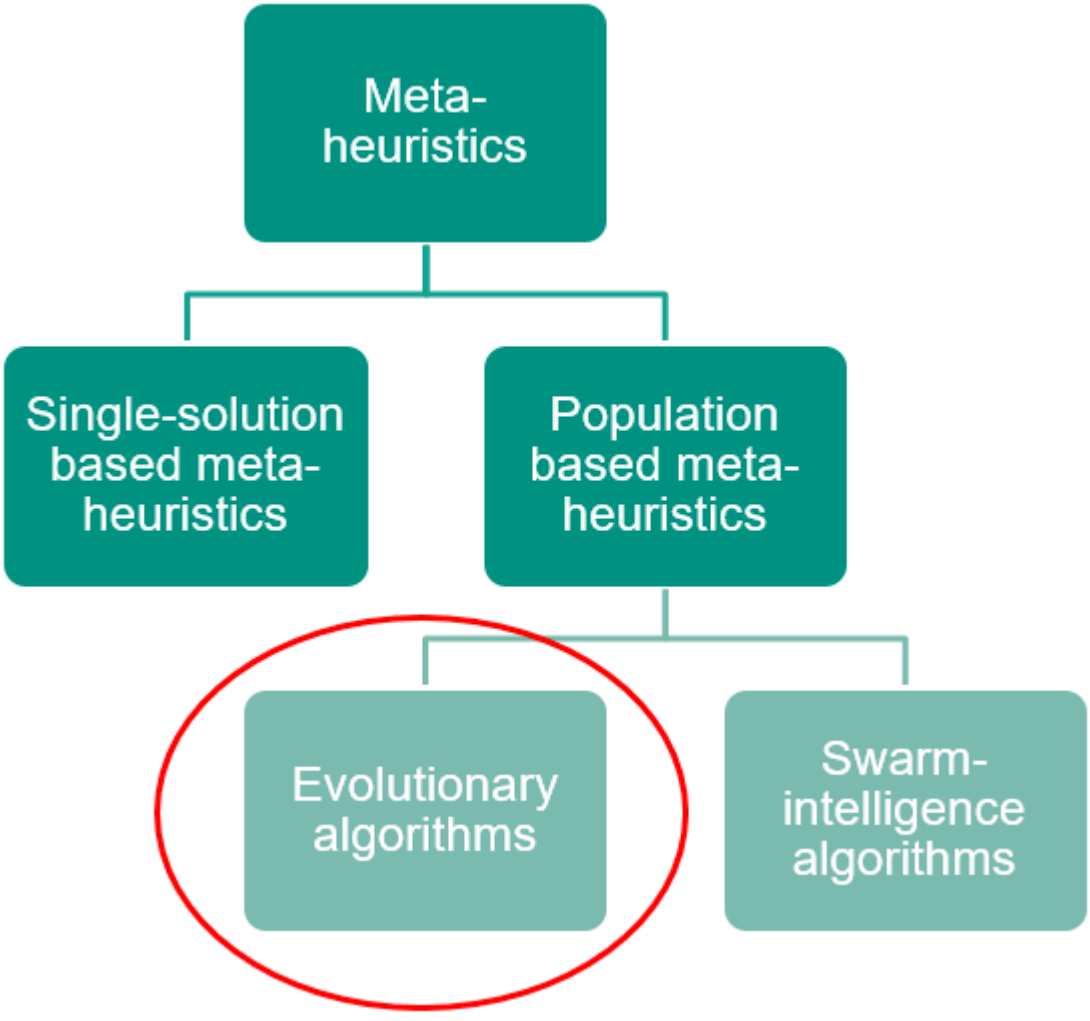


*Fig. 5.39* This figure is not included in the CC BY license of this book. Find out more here.

# Evolutionary algorithm - a high level structure

Consider an optimization problem to find

$$\min 10x_1 + 15x_2$$

such that:

$$\begin{cases} x_1 + 2x_2 \geq 2 \\ -2x_1 + 2x_2 \geq 1 \\ x_1, x_2 = 0 \text{ or } 1 \end{cases}$$

The evolutionary algorithm to solve an optimization problem like the one described above is described by the following scheme:

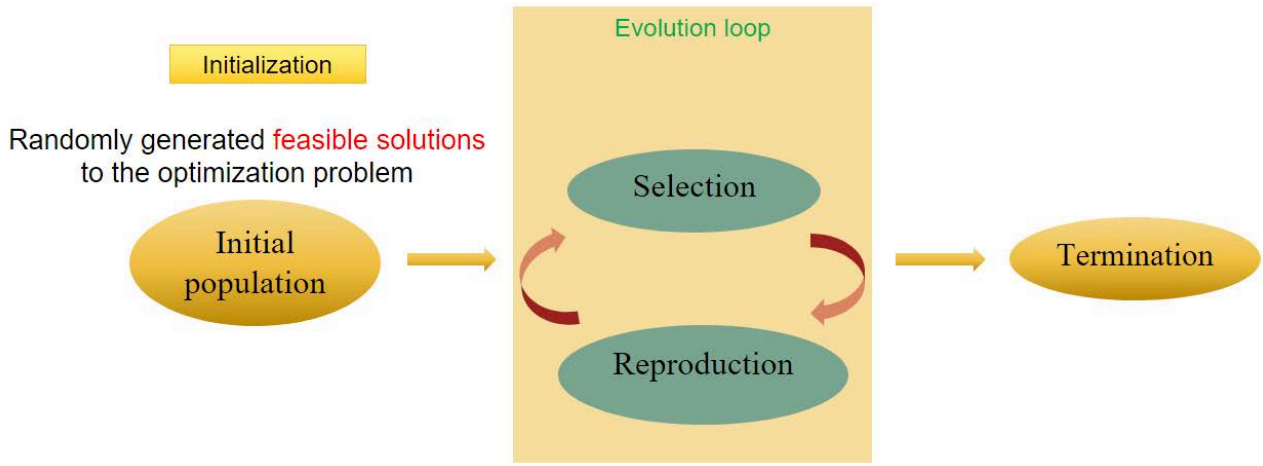



*Fig. 5.40* This figure is not included in the CC BY license of this book. Find out more here.

# General structure of evolutionary algorithms

Consider now a different problem, to find the value of $x$ that minimizes the function $y = x^2$, where $x \in [-10, 10]$.

**Initial population**

We would start by defining an initial population, randomly-generated feasible solutions to this optimization problem. We would get, for example:

$$\begin{cases} x_1 = -3.5 \\ x_2 = -1.5 \\ x_3 = 6 \\ x_4 = 9.3 \\ x_5 = 2.2 \end{cases}$$

**Selection**

The next step in the process is selection. We first need to calculate a fitness value for each individual in the population. In this example, we can define the **fitness function** as $f = x^2$. Following this:

$$\begin{cases} f(x_1) = 12.25 \\ f(x_2) = 2.25 \\ f(x_3) = 36 \\ f(x_4) = 86.49 \\ f(x_5) = 4.84 \end{cases}$$

**Reproduction - crossover**

Next we apply the crossover process, where we choose the two $x$ values with the best fitness function and we produce an offspring:

$$\begin{cases} x_2 = -1.5 \\ x_5 = 2.2 \end{cases} \implies x = \frac{x_2 + x_5}{2} = 0.35$$

**Reproduction - mutation**

We can change the value that we obtained by adding a small random number (which could be positive or negative). If the mutation step is a random number in the range $[-1, 1]$, then a possible result of the mutation could be:

$$x_{\text{mutated offspring}} = 0.35 - 0.05 = 0.3$$

Therefore, and replacing the value of the initial population with the worst fitness function, we get our new population given by:

$$\begin{cases} x_1 = -3.5 \\ x_2 = -1.5 \\ x_3 = 6 \\ x_5 = 2.2 \\ x_{\text{mo}} = 0.3 \end{cases}$$

**Termination**

The algorithm stops after running this cyclic process for a predetermined number of iterations, such as 500 or 5000 generations.

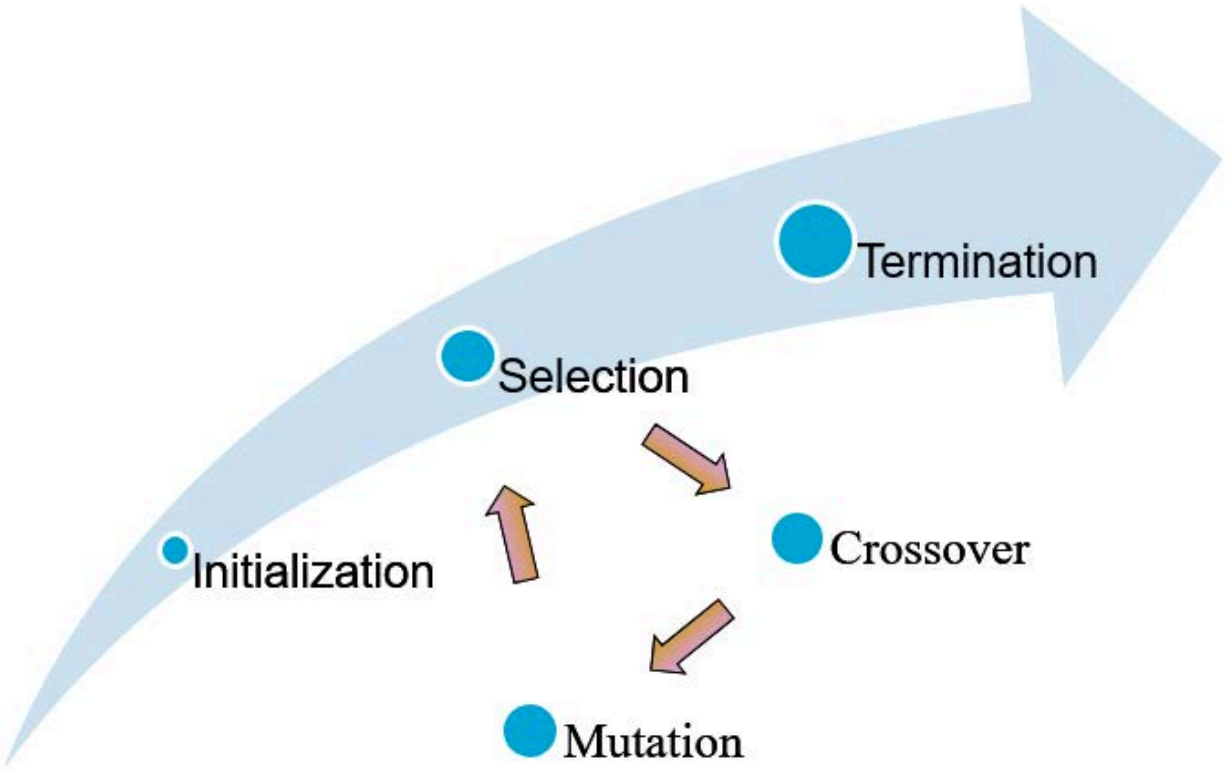



*Fig. 5.41* This figure is not included in the CC BY license of this book. Find out more here.

# Genetic algorithm

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms.

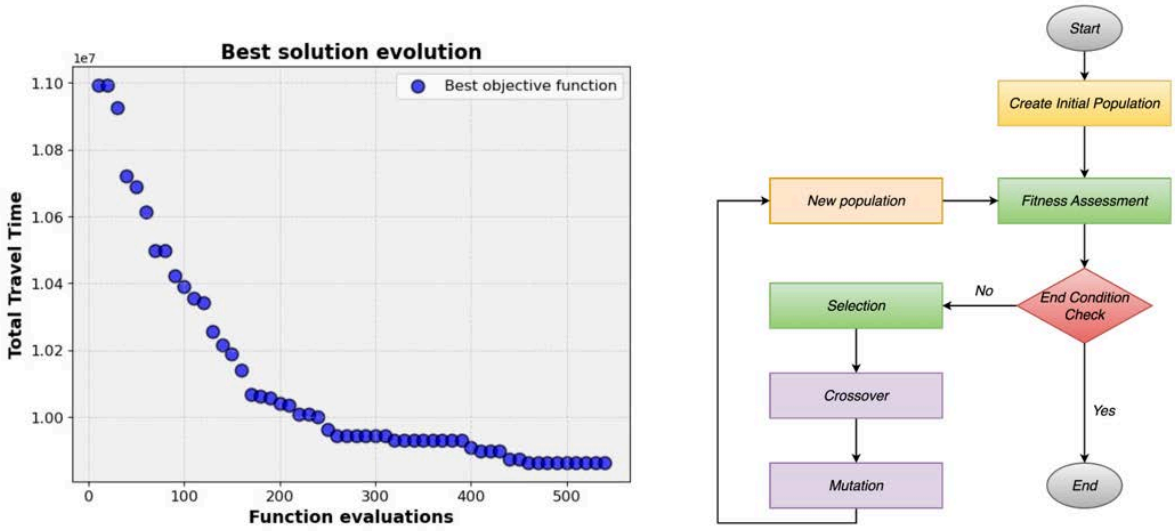



*Fig. 5.42* This figure is not included in the CC BY license of this book. Find out more here.

# Population Initialization

The process of generating an initial set of potential solutions to a problem - the “population”. Each member of this population is a potential solution, and should be feasible.

### Encoding

In genetic algorithms, potential solutions to problems are encoded as a series of symbols, usually a string of numbers or characters. This encoding method is similar to how genetic information is stored in DNA in nature

### Chromosome

In genetic algorithms, a chromosome represents a potential solution. It is a specific instance of encoding!

**Example**

Suppose we have an optimization problem given by

max $3x_1 + 4x_2 + 3x_3 + x_4 + 5x_5 + 2x_6$

such that

$$\begin{cases} x_1 + x_2 + x_3 \leq 2 \\ x_4 + x_5 + x_6 \leq 1 \\ x_1, x_2, x_3, x_4, x_5, x_6 = \{0, 1\} \end{cases}$$

Assuming a potential solution given by $\{x_1, x_2, x_3, x_4, x_5, x_6\} = \{1, 1, 0, 0, 0, 1\}$. Encoding here is: | 1 | 1 | 0 | 0 | 0 | 1 |. This also represents a specific chromosome.

#### Random generation

Random generation of four chromosomes for our example problem might give:

| 0 | 1 | 1 | 0 | 1 | 0 |

| 1 | 1 | 0 | 1 | 0 | 0 |

| 1 | 0 | 0 | 1 | 0 | 0 |

| 0 | 1 | 1 | 0 | 0 | 0 |

showing the values for $\{x_1, x_2, x_3, x_4, x_5, x_6\}$

# Fitness assessment

#### Fitness function

The fitness function should be designed to reflect the objectives of the problem. For example, in an optimization problem, it might measure how close the solution is to a desired outcome.

#### Role in genetic algorithms

- **Selection process:** fitness scores are used in the selection process, where individuals with better scores have a higher chance of being selected for reproduction

- **Guiding the evolution:** by favoring individuals with higher fitness scores, the algorithm “steers” the evolution of the population towards better solutions over successive generations

---

The fitness function is defined as the value of the objective function and, therefore, for the example shown above and using the first random solution generated, we would find a fitness value of 12!

# End condition check

**Maximum number of generations**

The algorithm terminates after a predefined number of generations have been produced

**Resource-based termination**

The algorithm stops after it has used a certain amount of computational resources like CPU time, memory, or a set number of function evaluations.

# Selection methods

**Roulette Wheel Selection (Fitness Proportionate Selection)**

Each individual’s probability of being selected is proportional to its fitness, such that

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

with

$$f = 3x_1 + 4x_2 + 3x_3 + x_4 + 5x_5 + 2x_6$$

In the example shown above, we would have:

| 0 | 1 | 1 | 0 | 1 | 0 |

Fitness value = 12 $\implies$ selected probability $\frac{12}{12+8+4+7} = 0.387$

| 1 | 1 | 0 | 1 | 0 | 0 |

Fitness value = 8 $\implies$ selected probability $\frac{8}{12+8+4+7} = 0.258$

| 1 | 0 | 0 | 1 | 0 | 0 |

Fitness value = 4 $\implies$ selected probability $\frac{4}{12+8+4+7} = 0.129$

| 0 | 1 | 1 | 0 | 0 | 0 |

Fitness value = 7 $\implies$ selected probability $\frac{7}{12+8+4+7} = 0.226$

# Crossover

First, let us better understand why is crossover done and its importance

- **Preserving good traits:** the goal is to combine the good characteristics of the parent solutions to potentially create even better offsprings
- **Diversity vs. Convergence:** crossover must be managed to maintain genetic diversity in the population to avoid premature convergence
- **Problem-specific adaptation:** sometimes, standard crossover methods are modified or completely new methods are developed to better suit specific problems, especially when the chromosome structure is complex or unique

And now, let us study different crossover methods:

- **Single-point crossover**
    1. Select two parents for mating

       | 0 | 1 | 1 | 0 | 1 | 0 |

       | 1 | 1 | 0 | 1 | 0 | 0 |

    2. Select a crossover point at random and swap the bits at the right site (let's say we do it after the first 2 bits in this example)
    3. Then the offsprings are:

       | 0 | 1 | 0 | 1 | 0 | 0 |

       | 1 | 1 | 1 | 0 | 1 | 0 |

- Two **-point crossover**
    1. Select two parents for mating

       | 0 | 1 | 1 | 0 | 1 | 0 |

| 1 | 1 | 0 | 1 | 0 | 0 |

2. Select two crossover points at random and swap the bits at the middle site (let's say we do it after the first 2 bits and before the last 2 bits in this example)
3. Then the offsprings are:

   | 0 | 1 | 0 | 1 | 1 | 0 |

   | 1 | 1 | 1 | 0 | 0 | 0 |

- Uniform crossover
  1. Select two parents for mating (we will choose the same as shown in the previous examples)
  2. At each bit position of the parents, toss a coin (T=1 will mean to swap the bits and T=0 to not swap them)
  3. Getting [1,0,0,1,1,0] as the final result for the coin tossing, then the offsprings for this method are:

     | 1 | 1 | 1 | 1 | 0 | 0 |

     | 0 | 1 | 0 | 0 | 1 | 0 |

- Half-uniform crossover
  1. Select two parents for mating (we will choose the same as shown in the previous examples)
  2. Toss a coin and swap the bits only if T=1
  3. Getting [1,x,0,1,1,x] for the final result of the coin tossing, the offsprings for this method are:

     | 1 | 1 | 1 | 1 | 0 | 0 |

     | 0 | 1 | 0 | 0 | 1 | 0 |

# Mutation: role and importance

**Exploration of solution space:**

Mutation helps to explore new areas of the solution space, potentially finding better solutions that are not in the vicinity of the current population.

**Maintaining diversity:**

By continuously introducing new genetic material, mutation prevents the population from becoming too similar, which can lead to getting stuck in sub-optimal solutions

**Balance with crossover:**

While crossover combines existing genetic material to explore the solution space, mutation introduces completely new genetic material.

---

Different mutation methods can be used, namely:

- **Bit Flip mutation:** for binary representations, this involves flipping bits (changing a 0 to a 1, or vice versa)
- **Swap mutation:** two genes in a chromosome are selected randomly and their values are swapped (this is often used in problems where the chromosome represents a sequence or ordering)

# New population

The newly generated offspring replace some individuals in the original population to form a new generation. The replacement strategy can be either complete replacement, where all old individuals are replaced by new ones, or partial replacement, where some well-performing old individuals are retained.

# Example

As an example, consider we adopt roulette wheel selection, single-point crossover, bit flip mutation, and complete replacement for our algorithm. Considering the optimization problem initially described and an initial population as given above, we start by applying **roulette wheel selection**:

- The chosen ones:
  | 0 | 1 | 1 | 0 | 1 | 0 |

  | 1 | 1 | 0 | 1 | 0 | 0 |
- The rest of the initial population:
  | 1 | 0 | 0 | 1 | 0 | 0 |

  | 0 | 1 | 1 | 0 | 0 | 0 |

Applying **single-point crossover:**

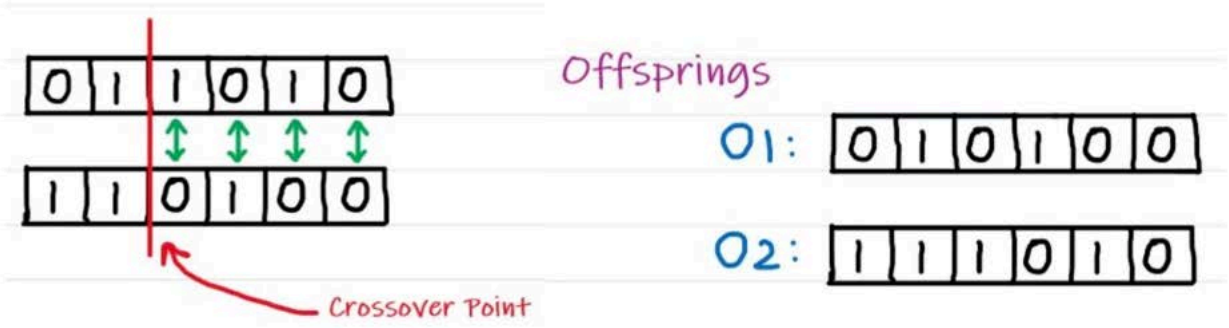



*Fig. 5.43* This figure is not included in the CC BY license of this book. Find out more here.

After this, we apply the **single-point crossover** to get:

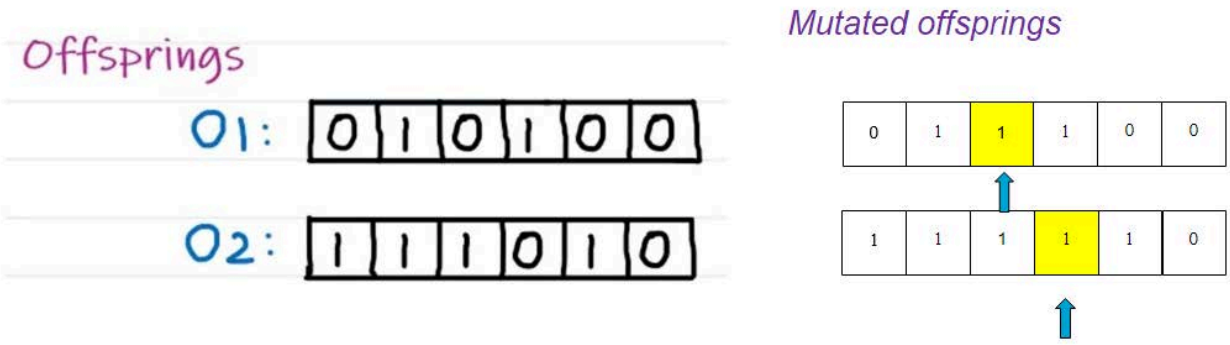



*Fig. 5.44* This figure is not included in the CC BY license of this book. Find out more here.

Finally, using complete replacement our new population is given by:

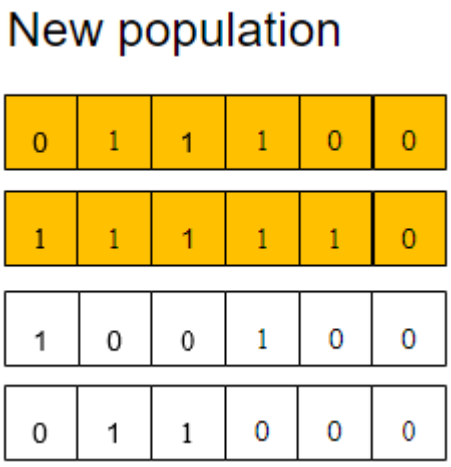



*Fig. 5.45* This figure is not included in the CC BY license of this book. Find out more here.

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Exercise: Airlines Problem (Optional)

## Contents


- Problem description
- Data preprocessing
- Model construction
- Solving the model
- Exercise
- Analysis
- Exercise


In this tutorial, we show you an example of a large-scale and (sort of) practical linear programming problem using real data.

**MUDE Exam Information**

The airlines problem serves as an example for solving more complex optimization problems and is included for practice. You're not expected to know the problem's details.

**Note**

Gurobi cannot be loaded in this online book, so download this notebook to work on it with Gurobi locally installed. Instruction on how to do that are in the `README.md` and `PA_2_4_gurobilicious.ipynb` of week 2.4.

# Problem description

## Data description

The dataset we will use for this example can be found here. Description of the data including column names is provided here. We also use monthly leasing rates for aircrafts, which can be found here.

The flight dataset includes monthly records of all flights in the US for each Origin-Destination city pair from the beginning of 1990 till the end of 2009. This data is used to model a realistic problem, with some simplifying assumptions, that can be formulated as a linear programming problem and solved using Gurobi.

## Problem definition

Our task is to specify the fleet size and the frequency of flights for each route and each aircraft type in each time period that minimizes the total operating and purchasing cost of the entire fleet while satisfying travel demand and operating constraints.

## Assumptions (to make the problem manageable)

- We want to model the problem as a linear program with continous decision variables
- There are no constraints on the type of aeroplane that can land at an airport; all airplanes can land at any airport, and we can operate in all cities.
- There is no minimum flight frequency.
- All aircrafts are leased, allowing us to change fleet size from one month to the next.

# Data preprocessing

You need to download the dataset and upload it on the folder where your notebook is.

```
import gurobipy as gp
import pandas as pd
import numpy as np

# read data
data = pd.read_csv('flight_edges.tsv', sep='\t', header=None)

# add column names (they are not included in the dataset but can be found in dataset desc
data.columns = ['Origin', 'Destination', 'Origin_City', 'Destination_City', 'Passengers',
                'Flights', 'Distance', 'Fly_Date', 'Origin_Population', 'Destination_Popu

# let's check out the first three rows of the data
data.head(3)
```

| | Origin | Destination | Origin_City | Destination_City | Passengers | Seats | Fligh |
|---|---|---|---|---|---|---|---|
| **0** | MHK | AMW | Manhattan, KS | Ames, IA | 21 | 30 | |
| **1** | EUG | RDM | Eugene, OR | Bend, OR | 41 | 396 | 2 |
| **2** | EUG | RDM | Eugene, OR | Bend, OR | 88 | 342 | 1 |

```
# check for missing values (if the resulting dataframe is empty, it means there is no mis
missing_rows = data[data.isna().any(axis=1)]
missing_rows
```

| | Origin | Destination | Origin_City | Destination_City | Passengers | Seats | Flights |
|---|---|---|---|---|---|---|---|

```
# eliminate columns we don't need (for this study)
data = data.drop(columns=['Origin_City', 'Destination_City', 'Origin_Population', 'Destin

# sort data based on OD and month
data = data.sort_values(['Origin', 'Destination', 'Fly_Date'])

# eliminate flights with the same OD (not sure why these records existed, but it makes se
data = data[data['Origin'] != data['Destination']]

# calculate sum of passengers for each record and add to data (they are spread in multipl
data['passenger_sum'] = data.groupby(['Origin', 'Destination', 'Fly_Date'])['Passengers']
data['Seats'] = data.groupby(['Origin', 'Destination', 'Fly_Date'])['Seats'].transform('s
data['Flights'] = data.groupby(['Origin', 'Destination', 'Fly_Date'])['Flights'].transfor
data['Passengers'] = data.groupby(['Origin', 'Destination', 'Fly_Date'])['Passengers'].tr

# eliminate duplicates (since aggregated group values, first row in each group is enough
data = data.drop_duplicates(subset=['Origin', 'Destination', 'Fly_Date'], keep='first')

# eliminate rows where the number of passengers is less than 5
data = data[data['passenger_sum'] > 5]

# some useful variables
demand_grouped_monthly = data.groupby(['Origin', 'Destination', 'Fly_Date'])['passenger_s

# let's check out how this grouped object looks like
demand_grouped_monthly.head(3)
```

```
Origin  Destination  Fly_Date
ABE     ACY          199007      73
        AGS          200402      47
                     200508      57
Name: passenger_sum, dtype: int64
```

```
# save the distance between each OD pair (since there will be duplicate OD pairs with the
distance = data.groupby(['Origin', 'Destination'])['Distance'].mean()

# let's examine the distance data
distance.head(3)
```

```
Origin  Destination
ABE     ACY            94.0
        AGS           618.0
        ALB           167.0
Name: Distance, dtype: float64
```

```
# save all unique OD pairs in a list; this gives us the full list of routes
# note: pandas unique function preserves order (which is the desired behavior here)
routes = pd.unique(distance.index.values.tolist())

# check the first 5 rows of the route list
routes[:5]
```

```
array([('ABE', 'ACY'), ('ABE', 'AGS'), ('ABE', 'ALB'), ('ABE', 'ATL'),
       ('ABE', 'AVL')], dtype=object)
```

```
# find all months that appear in the dataset and select the last (in this case) 6 months
# note: numpy unique function sorts the output (which is the desired behavior here)
months = np.unique(data['Fly_Date'])[-6:]
months
```

```
array([200907, 200908, 200909, 200910, 200911, 200912], dtype=int64)
```

```
# let's make a table with origin-destination pairs (routes) on rows and monthly demand on

# empty dataset with origin-destination pairs (routes) as rows and monthly demand as colu
demand_data = pd.DataFrame(index=routes, columns=[month for month in months])

# find monthly demand for each route and add to monthly demand dataset
# note that iterating through rows of a dataframe can be prohibitively slow, so we use a
for month in months:
    # initiate relevant column with zeros
    demand_data[month].values[:] = 0
    # temporary array to store iteration results
    demand_temp_all_routes = demand_data[month]
    # temporary array to store demand for routes that have none-zero demand values
    demand_temp_nonzero = demand_grouped_monthly[demand_grouped_monthly.index.get_level_v
    # eliminate month from array index (all values here have the same month)
    demand_temp_nonzero.index = demand_temp_nonzero.index.droplevel(2)
    # insert none-zero demand values into relevant rows
    demand_temp_all_routes[demand_temp_all_routes.index.intersection(demand_temp_nonzero.
    # assign iteration results (monthly demand) to monthly demand dataset
    demand_data[month] = demand_temp_all_routes

# convert demand data to a numpy array for efficiency
demand = demand_data.values

# let's see how it looks like
demand_data.head(4)
```

| | 200907 | 200908 | 200909 | 200910 | 200911 | 200912 |
|---|---|---|---|---|---|---|
| **(ABE, ACY)** | 0 | 0 | 0 | 0 | 0 | 0 |
| **(ABE, AGS)** | 0 | 0 | 0 | 0 | 0 | 0 |
| **(ABE, ALB)** | 0 | 0 | 0 | 0 | 0 | 0 |
| **(ABE, ATL)** | 3854 | 3307 | 2407 | 2855 | 2739 | 2344 |

Now we have the set of routes (OD pairs), set of months, distance for each route and demand for each route in each month. So we are ready to build the model in Gurobi.

# Model construction

Let's build a Gurobi model object first. Then we will add all required components to the model step by step.

```
# create a gurobi model object
model = gp.Model()

# just to avoid cluttering the notebook with unnecessary logging output
model.Params.LogToConsole = 0
```

```
Set parameter Username
Academic license - for non-commercial use only - expires 2023-05-06
```

# Parameters

We need average aircraft speed ($S$), number of working days per period ($N$), and daily operating hours ($H$) to use in the model. It is good Python practice to use meaningful names instead of notations, so we use full names for parameters. They are introduced below.

We also need the list of aircraft types ($j \in J$), their capacity ($C_j$), their hourly operating cost ($O_j$) and their monthly leasing cost ($L_j$). We store these in a multidictionary, which is a convenient way of storing related parameters and sets in Gurobi.

```
average_aircraft_speed = 500
working_day_per_period = 30
daily_operating_hours = 20

aircraft_type, aircraft_capacity, aircraft_hourly_operating_cost, aircraft_leasing_cost =
    'A320-200 CEO': [180, 4378, 196 * 1000],
    'A320 NEO':     [188, 4378, 315 * 1000],
    'A321-200 CEO': [230, 4432, 310 * 1000],
    'A321 NEO':     [236, 5149, 359 * 1000],
    'A330-200':     [436, 6564, 515 * 1000]
})
```

# Sets

We need the following sets to model this problem:

- set of routes (OD pairs) that we extracted from the data: $i \in I$
- set of months that we extracted from the data: $m \in M$

- set of aircraft types (extracted from the references mentioned in data description and listed above): $j \in J$

# Decision variables

The decision variables are:

- frequency of flights for each route and each aircraft type in each time period ($x_{ijm}$; **i**: route, **j**: aircraft type, **m**: month)
- fleet size for each route and each aircraft type in each time period ($y_{ijm}$; **i**: route, **j**: aircraft type, **m**: month)

```
## decision variables:

# frequency: Xijm; i: route, j: aircraft type, m: month
frequency = model.addVars(len(routes), len(aircraft_type), len(months), vtype=gp.GRB.CONT

# fleet size: Yijm; i: route, j: aircraft type, m: month
fleet_size = model.addVars(len(routes), len(aircraft_type), len(months), vtype=gp.GRB.CON
```

# Objective function

The objective function includes:

- aircraft leasing cost (sum for all routes, aircraft types and time periods): $\sum_{ijm} y_{ijm}. L_j$
- aircraft operating cost (frequency times hourly operating cost times flight duration, calculated as distance divided by average speed): $\sum_{ijm} x_{ijm}. O_j. (D_i/S_j)$

```
# objective 1 (aircraft leasing cost)
obj_leasing_cost = gp.quicksum(fleet_size[i, j, m] * aircraft_leasing_cost[aircraft_type[
                               for i in range(len(routes))
                               for j in range(len(aircraft_type))
                               for m in range(len(months)))
# objective 2 (aircraft operating cost)
obj_operating_cost = gp.quicksum(frequency[i, j, m] * aircraft_hourly_operating_cost[airc
                                 for i in range(len(routes))
                                 for j in range(len(aircraft_type))
                                 for m in range(len(months)))

# objective function
model.setObjective((obj_leasing_cost + obj_operating_cost), gp.GRB.MINIMIZE)
```

# Constraints

The constraints are:

- demand satisfaction constraints: the sum of frequencies multiplied by the capacity of all aircraft types should be greater than or equal to demand for each route in each time period

$$\sum_j x_{ijm}.C_j \geq D_{im}; \forall i \in I, \forall m \in M$$

- flight frequency constraint: the frequency of flights is restricted by the fleet size and available operating hours (per route, plane type and month)

$$x_{ijm}.(D_i/S_j) \leq (y_{ijm}.H.N); \forall i \in I, \forall j \in J, \forall m \in M$$

```
# demand satisfaction constraints
model.addConstrs(
    gp.quicksum(frequency[i, j, m] * aircraft_capacity[aircraft_type[j]] for j in range(l
    >= demand[i, m]
    for i in range(len(routes))
    for m in range(len(months)))

# flight frequency constraint
model.addConstrs(frequency[i, j, m] * distance[i] / average_aircraft_speed
                 <= fleet_size[i, j, m] * daily_operating_hours * working_day_per_period
                 for i in range(len(routes))
                 for j in range(len(aircraft_type))
                 for m in range(len(months)))

model.update()
```

# Complete model formulation

$$min_{x,y}$$

$$\sum_{ijm} y_{ijm}.\, L_j + \sum_{ijm} x_{ijm}.\, O_j.\, (D_i/S_j)$$

$$s.t.$$

$$\sum_{j} x_{ijm}.\, C_j \geq D_{im}; \qquad \forall i \in I, \forall m \in M$$

$$x_{ijm}.\, (D_i/S_j) \leq (y_{ijm}.\, H.\, N); \qquad \forall i \in I, \forall j \in J, \forall m \in M$$

$$x_{ijm}, y_{ijm} \geq 0$$

# Solving the model

For solving linear programming problems with continuous decision variables, possible solution algorithms (available with Gurobi) are:

- primal simplex
- dual simplex
- barrier algorithm (which is a specific type of interior point method).

When we ask Gurobi to solve a model without specifying the algorithm, it selects what it thinks is the best algorithm to solve each specific problem. It uses the concurrent optimizer, which is the default option for the solution algorithm.

Here we solve the model without specifying the solution method (letting Gurobi decide which algorithm to use). In the next tutorial, we discuss different algorithms.

```
# to see the output (remember we turned this off before)
model.Params.LogToConsole = 1

# solve the model
model.optimize()
```

```
Set parameter LogToConsole to value 1
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 959616 rows, 1599360 columns and 2399040 nonzeros
Model fingerprint: 0x97205460
Coefficient statistics:
  Matrix range     [4e-03, 6e+02]
  Objective range  [2e+01, 5e+05]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+00, 1e+05]

Concurrent LP optimizer: dual simplex and barrier
Showing barrier log only...

Presolve removed 959616 rows and 1599360 columns
Presolve time: 1.44s
Presolve: All rows and columns removed
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    7.9354517e+09   0.000000e+00   0.000000e+00      3s

Solved with dual simplex
Solved in 0 iterations and 2.73 seconds (0.88 work units)
Optimal objective  7.935451713e+09
```

# Exercise

Try to verify feasibility of the optimal solution as well as the objective value of this solution. You can access the optimal values of decision variables via their x attribute (e.g., fleet_size[i, j, m].x)

# Analysis

Let’s check out the optimal solution.

```
optimal_results = pd.DataFrame(columns=["Origin", "Destination", "Aircraft Type", "Month"
for i in range(len(routes)):
    for j in range(len(aircraft_type)):
        for m in range(len(months)):
            if frequency[i, j, m].x > 1e-6 or fleet_size[i, j, m].x > 1e-6:
                optimal_results = optimal_results.append({"Origin": routes[i][0],
                                                          "Destination": routes[i][1],
                                                          "Aircraft Type": aircraft_type[
                                                          "Month": months[m],
                                                          "Fleet size": fleet_size[i,j,m]
                                                          "Frequency": frequency[i,j,m].x
optimal_results.head(20)
```

| | Origin | Destination | Aircraft Type | Month | Fleet size | Frequency |
|---|---|---|---|---|---|---|
| **0** | ABE | ATL | A330-200 | 200907 | 0.020390 | 8.839450 |
| **1** | ABE | ATL | A330-200 | 200908 | 0.017496 | 7.584862 |
| **2** | ABE | ATL | A330-200 | 200909 | 0.012734 | 5.520642 |
| **3** | ABE | ATL | A330-200 | 200910 | 0.015104 | 6.548165 |
| **4** | ABE | ATL | A330-200 | 200911 | 0.014491 | 6.282110 |
| **5** | ABE | ATL | A330-200 | 200912 | 0.012401 | 5.376147 |
| **6** | ABE | AVL | A330-200 | 200907 | 0.000186 | 0.105505 |
| **7** | ABE | AVP | A330-200 | 200908 | 0.000018 | 0.107798 |
| **8** | ABE | CLE | A330-200 | 200907 | 0.007348 | 6.502294 |
| **9** | ABE | CLE | A330-200 | 200908 | 0.007749 | 6.857798 |
| **10** | ABE | CLE | A330-200 | 200909 | 0.006122 | 5.417431 |
| **11** | ABE | CLE | A330-200 | 200910 | 0.007177 | 6.350917 |
| **12** | ABE | CLE | A330-200 | 200911 | 0.006018 | 5.325688 |
| **13** | ABE | CLE | A330-200 | 200912 | 0.004999 | 4.424312 |
| **14** | ABE | CLT | A330-200 | 200907 | 0.019529 | 12.201835 |
| **15** | ABE | CLT | A330-200 | 200908 | 0.018623 | 11.635321 |
| **16** | ABE | CLT | A330-200 | 200909 | 0.018193 | 11.366972 |
| **17** | ABE | CLT | A330-200 | 200910 | 0.018681 | 11.672018 |
| **18** | ABE | CLT | A330-200 | 200911 | 0.017940 | 11.208716 |
| **19** | ABE | CLT | A330-200 | 200912 | 0.017474 | 10.917431 |

Did you notice something odd about the results? The frequencies and fleet sizes are represented as decimal numbers. This occurs because we specified the decision variable types as continuous during their declaration.

While integer decision variables are more realistic for representing quantities such as flight frequencies or fleet sizes, they significantly increase the complexity of the problem. Large-scale linear programs with integer decision variables can be much more challenging to solve and, in some cases, practically infeasible.

Using continuous variables in this example allows us to simplify the problem and focus on learning core optimization techniques. Such simplifications are also common in practice, especially when approximate solutions are sufficient for decision-making.

Now let's try to force Gurobi to only find solutions with integer decision variable values. In this case, we can do that by iterating through decision variables and changing their type from continuous to integer (it's not always that simple).

```
# discard existing solution information from the model
model.reset(0)

# iterate over decision variables and modify their type to integer (remember we declare t
for var in model.getVars():
    var.vtype = gp.GRB.INTEGER
```

```
Discarded solution information
```

```
# optimize the modified model
model.optimize()
```

```
Gurobi Optimizer version 9.5.1 build v9.5.1rc2 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 959616 rows, 1599360 columns and 2399040 nonzeros
Model fingerprint: 0xb003a3e3
Variable types: 0 continuous, 1599360 integer (0 binary)
Coefficient statistics:
  Matrix range     [4e-03, 6e+02]
  Objective range  [2e+01, 5e+05]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+00, 1e+05]
Found heuristic solution: objective 2.043082e+10
Presolve removed 809084 rows and 1599350 columns (presolve time = 35s) ...
Presolve removed 959610 rows and 1599350 columns
Presolve time: 35.19s
Presolved: 6 rows, 10 columns, 15 nonzeros
Found heuristic solution: objective 1.588337e+10
Variable types: 0 continuous, 10 integer (5 binary)

Explored 0 nodes (0 simplex iterations) in 35.82 seconds (2.52 work units)
Thread count was 8 (of 8 available processors)

Solution count 2: 1.58834e+10 2.04308e+10

Optimal solution found (tolerance 1.00e-04)
Best objective 1.588336894802e+10, best bound 1.588282454130e+10, gap 0.0034%
```

The optimal objective function value is higher (less desirable) compared to the continuous problem, which was to be expected. Can you guess why?

Also note that the computation time is significantly higher compared to the continuous problem.

Now let's make a table and look at the results.

```
optimal_results_integer = pd.DataFrame(columns=["Origin", "Destination", "Aircraft Type",
for i in range(len(routes)):
    for j in range(len(aircraft_type)):
        for m in range(len(months)):
            if frequency[i, j, m].x > 1e-6 or fleet_size[i, j, m].x > 1e-6:
                optimal_results_integer = optimal_results_integer.append({"Origin": route
                                                                          "Destination":
                                                                          "Aircraft Type"
                                                                          "Month": months
                                                                          "Fleet size": f
                                                                          "Frequency": fr

optimal_results_integer.head(20)
```

| | Origin | Destination | Aircraft Type | Month | Fleet size | Frequency |
|---|---|---|---|---|---|---|
| **0** | ABE | ATL | A320-200 CEO | 200907 | 1.0 | 22.0 |
| **1** | ABE | ATL | A320-200 CEO | 200908 | 1.0 | 19.0 |
| **2** | ABE | ATL | A320-200 CEO | 200909 | 1.0 | 14.0 |
| **3** | ABE | ATL | A320-200 CEO | 200910 | 1.0 | 16.0 |
| **4** | ABE | ATL | A320-200 CEO | 200911 | 1.0 | 16.0 |
| **5** | ABE | ATL | A320-200 CEO | 200912 | 1.0 | 14.0 |
| **6** | ABE | AVL | A320-200 CEO | 200907 | 1.0 | 1.0 |
| **7** | ABE | AVP | A320-200 CEO | 200908 | 1.0 | 1.0 |
| **8** | ABE | CLE | A320-200 CEO | 200907 | 1.0 | 16.0 |
| **9** | ABE | CLE | A320-200 CEO | 200908 | 1.0 | 17.0 |
| **10** | ABE | CLE | A320-200 CEO | 200909 | 1.0 | 14.0 |
| **11** | ABE | CLE | A320-200 CEO | 200910 | 1.0 | 16.0 |
| **12** | ABE | CLE | A320-200 CEO | 200911 | 1.0 | 13.0 |
| **13** | ABE | CLE | A320-200 CEO | 200912 | 1.0 | 11.0 |
| **14** | ABE | CLT | A320-200 CEO | 200907 | 1.0 | 30.0 |
| **15** | ABE | CLT | A320-200 CEO | 200908 | 1.0 | 29.0 |
| **16** | ABE | CLT | A320-200 CEO | 200909 | 1.0 | 28.0 |
| **17** | ABE | CLT | A320-200 CEO | 200910 | 1.0 | 29.0 |
| **18** | ABE | CLT | A320-200 CEO | 200911 | 1.0 | 28.0 |
| **19** | ABE | CLT | A320-200 CEO | 200912 | 1.0 | 27.0 |

Now fleet size and frequencies make more sense. If you still haven't figured out why the objective function value of the integer problem is a bit greater, comparing fleet size and

frequency values of integer and continuous variations should help you understand that (hint: aircraft leasing cost depends on the fleet size and aircraft operating cost depends on the frequency).

# Exercise

As you saw above, computation time of solving the model changed when we changed decision variable types from continuous to integer. On the other hand, computation time usually increases with problem size. Try to change the size of this problem (simply by changing the number of months to include in input cell 5 from 6 to a different number and rerunning the code), solve both continuous and integer variations, and record computation times. You can go all the way up to 240 months (but note that this might take quite some time). Make a figure to see the trends properly (e.g., number of months on the x-axis, computation time on the y-axis, and a separate line for each problem variation). What can you conclude?

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Road Network Design Problem

## Contents


- Problem description
- Mathematical description
- Adaption for GA
- PyMOO
- Implementation in Python


> **Attribution**
>
> This chapter reuses content Bahman Ahmadi et al. (2024). Find out more here.

During Friday's session, you will work on the road Network Design Problem (NDP) and explore its implementation using two optimization approaches: Mixed-Integer Linear Programming (MILP) and a Genetic Algorithm (GA). The problem description, mathematical formulation, and Python implementation are provided in this book, should you wish to prepare in advance for the session.

> **MUDE Exam Information**
>
> The road Network Design Problem serves as an example for solving a complex optimization problem. You're not expected to understand the problem details for the exam.

## Problem description

The road Network Design Problem (NDP) involves determining which road links to build, refurbish, or upgrade to improve the performance of a road network. A specific variation of the problem is the road NDP with capacity expansions, where the objective is to decide

which links should have their capacity increased. This is a complex problem that must take into account a variety of factors, including:

- The current state of the road network
- The projected future traffic demand
- The budget available for improvements
- The impacts of the adjustments (could be environmental, social, etc).

There are a variety of approaches to dealing with the road NDP with capacity expansions. One common approach is to use mathematical optimization models. In this assignment, we use a simplified example to show how optimization can be used to tackle road NDPs. Note that the classical approaches to dealing with road NDPs can be more complicated and will be covered in other courses in the TTE track of the Civil Engineering Master's program.

In this assignment, the goal is to minimize the total travel time on the network by selecting a predefined number of links for capacity expansion (subject to the available budget). The following (main) assumptions and simplifications are made to make the problem solvable using methods and algorithms that you have learned so far.

# 1. Link travel time function

The travel time on a stretch of road (i.e., a link) depends on the flow of vehicles (vehicles/hour) on that link and the capacity of the link (maximum of vehicles/hour). The most common function to calculate travel time on a link is the so-called Bureau of Public Roads (BPR) function, which is a polynomial (degree 4) function. That function, if used in the assignment, would make the problem non-linear and therefore very hard to solve. So we use a simplified linear function where travel time increases linearly with the flow of vehicles on a road link. More details are provided within the formulation section.

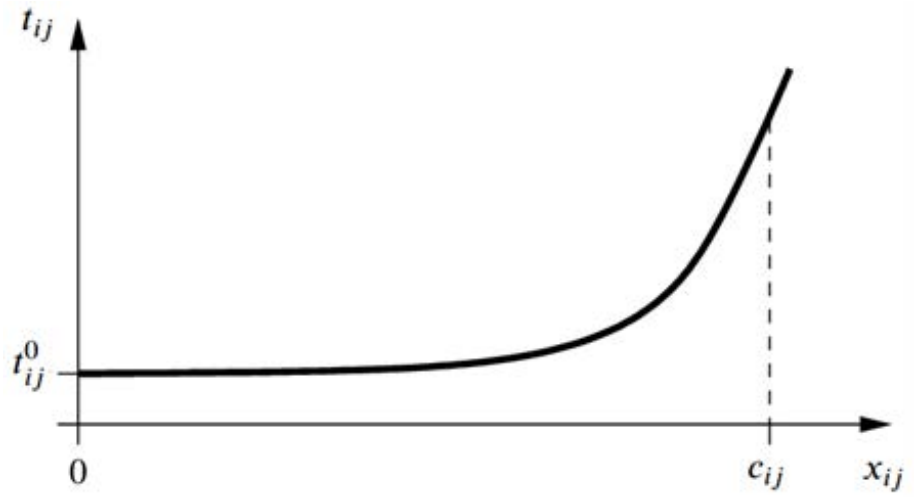


*Fig. 5.46* Link travel time function

$$t_{ij} = t_{ij}^0 \left(1 + \alpha \left(\frac{x_{ij}}{c_{ij}}\right)^{\beta}\right) \quad (i,j) \in A$$

Where $t_{ij}$ is the current travel time on the link, $t_{ij}^0$ is the travel time without congestion (free flow), $x_{ij}$ is the flow of cars, and $c_{ij}$ the capacity in maximum flow of cars. $\alpha$ and $\beta$ are calibration parameters that should be adapted to the reality of the region in which the model is being run. However, typically $\alpha$ is $1$, and $\beta$ is $4$.

# 2. Route choice behavior

To assess the quality of solutions for the road capacity expansion problem, it is essential to understand the effect of added capacity on travel time. This requires not only modeling the travel time-flow relationship but also knowing how vehicles distribute themselves across the network.

In congested networks, the route choice behavior of drivers often follows the so-called User Equilibrium (UE) principle. Under UE, each traveler seeks to minimize their own individual generalized travel time. The principle states that, for each origin-destination pair, all routes used have equal and minimal travel times. In this state of equilibrium, no driver can improve their travel time by switching to a different route.

However, calculating UE requires advanced methods, which are beyond the scope of the MUDE course. Instead, we assume that drivers follow the so-called System Optimal (SO) principle. Under SO, route choices are made to minimize the total travel time across the entire network, summed over all drivers. This means some drivers may take longer routes to allow others to save time, a concept also referred to as social equilibrium.

The SO equilibrium is easier to compute than the UE. However, it is important to note that achieving a system-optimal traffic distribution in real-world road networks is nearly impossible, as drivers independently make route choices based on personal preferences and information. In practice, it is not feasible to dictate where drivers should go.

Example of a perfect user equilibrium in a very simple road:

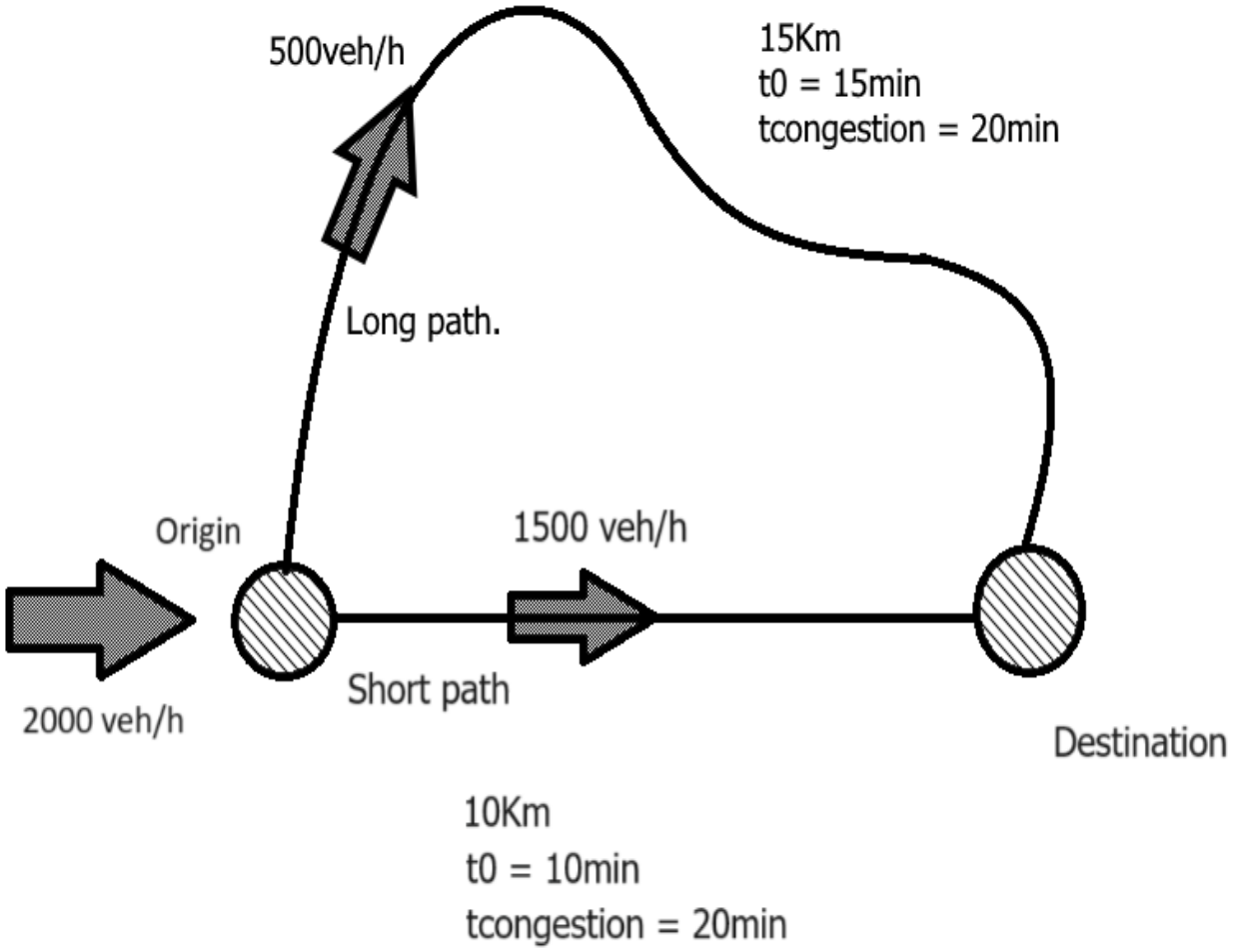



*Fig. 5.47* Route choice behavior according to the user equilibrium (UE) principle

# 3. Quadratic terms

In the formulation of the Road NDP, even after applying the simplifications mentioned earlier, a quadratic term arises. Specifically, the flow (a variable) must be multiplied by the travel time (also a variable), making the problem non-linear.

Fortunately there are different methods to transform quadratic terms to linear variables and constraints with mathematical programming. You do not need to learn these techniques. Most solvers (including Gurobi) can handle this well given some adjustments to the formulation. Gurobi uses the McCormick Envelopes (MCE) to transform each quadratic term into one new variable and four constraints. For more information on MCE and how Gurobi implements them, check these links (but this is not part of your learning goals):

- The theory behind MCE
- Gurobi webinar presentation on quadratic optimization
- Full video of the webinar

In this assignment, we will proceed with a quadratic term in the objective function. With the earlier simplifications and assumptions, we can formulate and solve the road NDP using the branch-and-bound method, which you have studied before.

We are using the Sioux Falls network which is one of the [most used networks in transportation research]

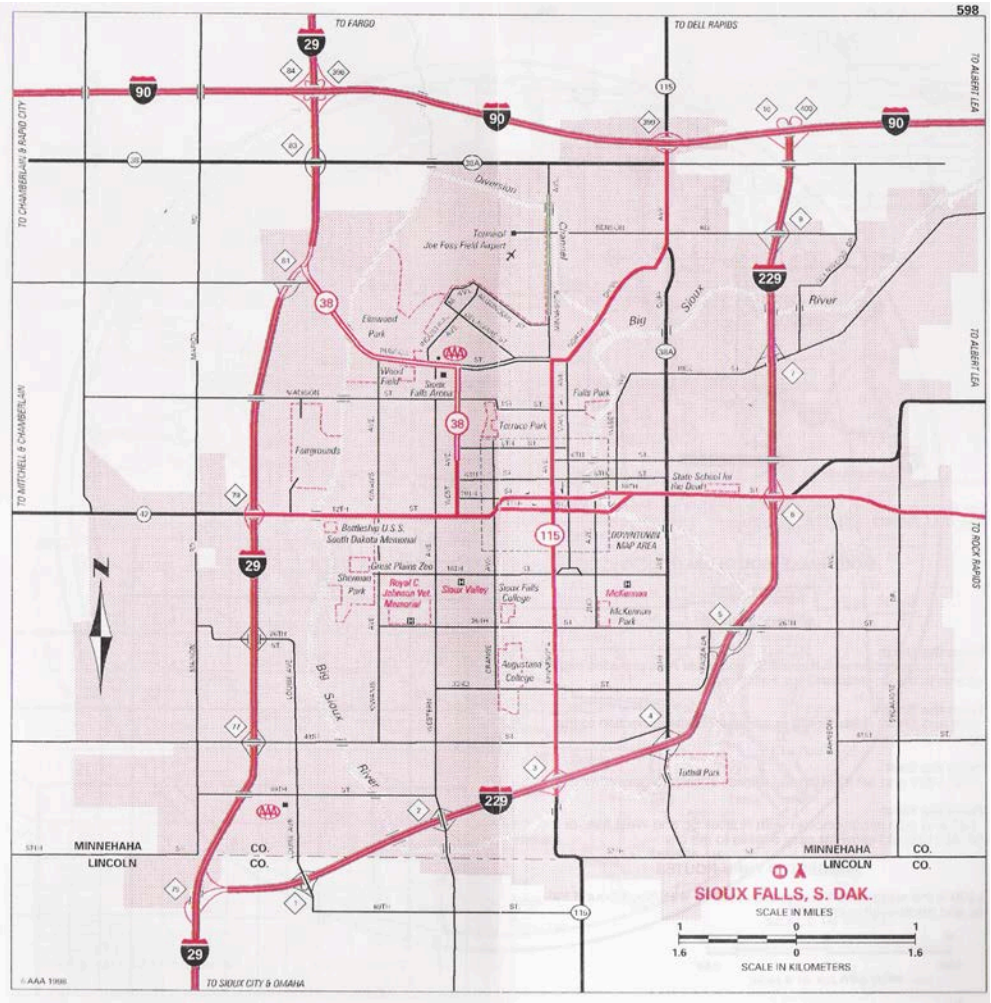



*Fig. 5.48* SiouxFalls map

# Mathematical description

## Decision variables

We have a set of binary variables $y_{ij}$, these variables take the value 1 if link $(i, j)$ connecting node $i$ to node $j$ is selected for expansion, and 0 otherwise.

We also have two sets of decision variables representing link flows; $x_{ij}$, representing flow on link $(i, j)$ in cars per hour, and $x_{ijs}$, representing flow on link $(i, j)$ going to destination $s$

.The first represents the total number of cars passing on that road, while the second represents the number of cars specifically going to destination $s$. Summing the latter over all $s$ results in the former for a link $(i, j)$.

Therefore, mathematically we define the domain of the variables as follows:

$$
\begin{aligned}
&y_{ij} \in \{0, 1\} \quad \forall (i, j) \in A \\
&x_{ij} \geq 0 \quad \forall (i, j) \in A \\
&x_{ijs} \geq 0 \quad \forall (i, j) \in A, \forall s \in D
\end{aligned}
$$

As you will see in the code implementation, we have one extra set of variables called `x2` (representing $x^2$). This is to help Gurobi isolate quadratic terms and perform required transformations based on MCE to keep the problem linear. This is not part of your learning goals.

**Test yourself**

Can the $x_{ij}$ variables be integer?

- ○ Yes
- ○ No

# Objective function

The objective function of the problem (in its simplest form), is the minimization of the total travel time on the network. This is calculated by multiplying the flow of vehicles on each link by the corresponding travel time and summing over all links ($A$ represents the collection of all links to simplify the notation):

$$
Z = \sum_{(i,j) \in A} x_{ij} \cdot t_{ij}
$$

The travel time $t_{ij}$ is a function of the flow on a link and can be expressed as follows (where $\beta$ is a parameter):

$$t_{ij} = t_{ij}^0 \cdot \left(1 + \beta \frac{x_{ij}}{c_{ij}}\right) \quad \forall (i,j) \in A$$

> Note that the commonly used travel time function based on the Bureau of Public Roads (BPR) is as follows:
>
> $$t_{ij} = t_{ij}^0 \cdot \left(1 + \alpha \left(\frac{x_{ij}}{c_{ij}}\right)^{\beta}\right) \quad \forall (i,j) \in A$$
>
> Where $\beta$ usually assumes the value of $4$ making this function (and the problem) non-linear. Therefore, we use the linear function mentioned before to make the problem manageable using the methods we have learned so far.

The following constraint yields the capacity of each link based on which ones are selected for expansion, when $y_{ij}$ is 1 there is added capacity as you can see:

$$c_{ij} = (1 - y_{ij}) \cdot c_{ij}^0 + y_{ij} \cdot c_{ij}^1 \quad \forall (i,j) \in A$$

This allows us to represent $t_{ij}$ as:

$$t_{ij} = t_{ij}^0 \cdot \left(1 + \beta \cdot x_{ij} \cdot \left(\frac{1 - y_{ij}}{c_{ij}^0} + \frac{y_{ij}}{c_{ij}^1}\right)\right) \quad \forall (i,j) \in A$$

Which leads to the following extended objective function:

$$Z = \sum_{(i,j) \in A} x_{ij} \cdot \left(t_{ij}^0 \cdot \left(1 + \beta \cdot x_{ij} \cdot \left(\frac{1 - y_{ij}}{c_{ij}^0} + \frac{y_{ij}}{c_{ij}^1}\right)\right)\right)$$

Now, for Gurobi (and other solvers as well), it is necessary to keep binary variables and quadratic terms clean and separate for the solver to perform the required transformations to linearize the problem. Thus, the following, despite its complexity, is the most solver-friendly formulation of our objective function:

$$Z = \sum_{(i,j)\in A} t_{ij}^0 \cdot x_{ij} + \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \right) - \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \cdot y_{ij} \right) + \sum_{(i,j)\in A} \left( t_{i}^{0} \right.$$

Therefore, we use this equation to model our objective function in Gurobi.

# Constraints

We have four sets of constraints for this problem. Let's go through them one by one and add them to the model.

## 1. Budget constraint

We can only extend the capacity of certain number of links based on the available budget. So first, we have to make sure to limit the number of extended links to the max number that can be expanded:

$$\sum_{(i,j)\in A} y_{ij} = B$$

**Test yourself**

Can the sign $\leq$ be used instead, do you expect the solution to be different?

- ○ Yes
- ○ No

## 2. Link flow conservation constraints

We have two sets of decision variables representing link flows; $x_{ij}$, which represents the total flow on link $(i, j)$, and $x_{ijs}$, which represents flow on link $(i, j)$ going to destination $s$. So we have to make sure that the sum of the flows over all destinations equals the flow on each link.

$$\sum_{s \in D} x_{ijs} = x_{ij} \quad \forall (i,j) \in A$$

## 3. Node flow conservation constraints

This set of constraints ensures that the total incoming and outgoing flow at each node balances, except at origin and destination nodes where flow is either generated or absorbed. For example, at an intersection, the number of vehicles entering equals the number leaving, except for origins (extra outgoing flow) and destinations (extra incoming flow).

$d_{is}$ here is the number of travelers from node $i$ to node $s$ with the exception of $d_{ss}$, which is all the demand that arrives at node $s$.

$$\sum_{j \in N;(i,j) \in A} x_{ijs} - \sum_{j \in N;(j,i) \in A} x_{jis} = d_{is} \quad \forall i \in N, \forall s \in D$$

The figure gives an example:

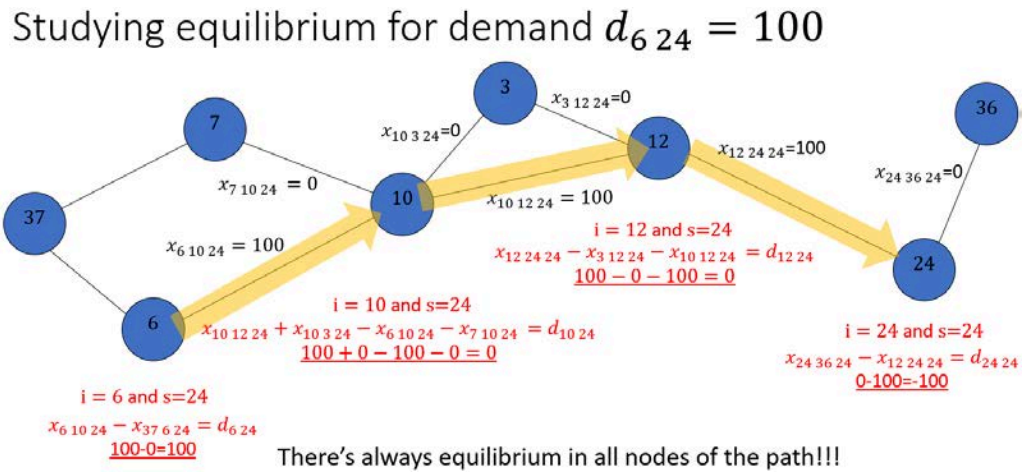



*Fig. 5.49* Constraint example

$$\sum_{j \in N;(i,j) \in A} x_{ijs} - \sum_{j \in N;(j,i) \in A} x_{jis} = d_{is} \quad \forall i \in N, \forall s \in D$$

**Test yourself**

What does the constraint 3 in the above formulation do?

- ○ To ensure that vehicles don't magically appear and disappear and do partial paths on the road
- ○ To ensure that all cars that leave a node always return at the end of the day to that node

## 4. Quadratic variable constraints (you do not need to fully understand this)

These constraints are essentially dummy equations to help Gurobi handle the quadratic terms (introduced earlier as dummy variables). Instead of directly using $x_{ij}^2$ in the model, we define a new set of decision variables and constraints to represent their values. This informs Gurobi that these are quadratic terms and allows it to replace them with the necessary variables and constraints to maintain linearity. This is not part of your learning goals!

# Conclusions

Here is the complete problem formulation.

$$min \quad \sum_{(i,j)\in A} t_{ij}^0 \cdot x_{ij} + \sum_{(i,j)\in A} \left( \frac{t_{ij}^0 \cdot \beta}{c_{ij}^0} \cdot x_{ij}^2 \right) - \sum_{(i,j)\in A} \left( \frac{t_{ij}^0 \cdot \beta}{c_{ij}^0} \cdot x_{ij}^2 \cdot y_{ij} \right) + \sum_{(i,j)\in A} \left( t$$

s.t.

$$\sum_{(i,j)\in A} y_{ij} = B$$

$$\sum_{s\in D} x_{ijs} = x_{ij}, \quad \forall (i,j) \in A$$

$$\sum_{j\in N;(i,j)\in A} x_{ijs} - \sum_{j\in N;(j,i)\in A} x_{jis} = d_{is}, \quad \forall i \in N, \forall s \in D$$

$$y_{ij} \in \{0,1\}, \quad \forall (i,j) \in A$$

$$x_{ij} \geq 0, \quad \forall (i,j) \in A$$

$$x_{ijs} \geq 0, \quad \forall (i,j) \in A, \forall s \in D$$

# Adaption for GA

To be able to use a Genetic Algorithm (GA), we need to define the problem slightly differently. Essentially, we break down the problem into two sub-problems:

1. The Traffic Assignment (TA) problem: the route choices of the drivers.
2. The road Network Design Problem (NDP): where we select which links should be upgraded.

We solve the problem by iteratively going between the Traffic assignment and the Design Problem. The idea is for the GA to move to better networks as generations pass which are evaluated by the traffic assignment process that you have learned. We use Gurobi to solve the Traffic Assignment sub-problems, which provide us with the objective function (or fitness function within the context of GA) value of the decision problem (which will be dealt with using GA). This is usually referred to as the iterative-optimization-assignment method since we iteratively improve the objective function value of the NDP using the assignment problem.

Let's see how this works.

# The Network Design sub-problem

The network design is where we use the genetic algorithm. As explained before, GA uses a population of solutions and iteratively improves this population to evolve to new generations of populations with a better objective function value (being that minimization or maximization). In this problem, the decision variables are links for capacity expansion and the objective function value is the total system travel time that we want to minimize.

$$min \quad \sum_{(i,j)\in A} t_{ij}^0 \cdot x_{ij} + \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \right) - \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \cdot y_{ij} \right) + \sum_{(i,j)\in A} \left($$

s.t.

$$\sum_{(i,j)\in A} y_{ij} = B$$

$$y_{ij} \in \{0,1\}, \quad \forall (i,j) \in A$$

Where the values of $x_{ij}$ are not decision variables anymore, they will be obtained from solving the Traffic Assignment problem with Gurobi which evaluates the travel times on the network.

# The Traffic Assignment sub-problem

This is just part of the original NDP that assigns traffic to the network based on a set of given capacity values, which are defined based on the values of the DP decision variables (links selected for capacity expansion). The main difference (and the advantage) here is that by separating the binary decision variables, instead of a mixed integer programming problem, which are hard to solve, here we have a quadratic programming problem with continuous decision variables, which will be transformed to a linear problem that Gurobi can solve very fast.

$$min \quad \sum_{(i,j)\in A} t_{ij}^0 \cdot x_{ij} + \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \right) - \sum_{(i,j)\in A} \left( t_{ij}^0 \cdot \frac{\beta}{c_{ij}^0} \cdot x_{ij}^2 \cdot y_{ij} \right) + \sum_{(i,j)\in A}$$

s.t.

$$\sum_{s\in D} x_{ijs} = x_{ij}, \quad \forall (i,j) \in A$$

$$\sum_{j\in N;(i,j)\in A} x_{ijs} - \sum_{j\in N;(j,i)\in A} x_{jis} = d_{is}, \quad \forall i \in N, \forall s \in D$$

$$x_{ij} \geq 0, \quad \forall (i,j) \in A$$

$$x_{ijs} \geq 0, \quad \forall (i,j) \in A, \forall s \in D$$

Where the values of $y_{ij}$ are constant and are defined by GA.

**Test yourself**

Select the right answer(s):

- [ ] We divided the NDP into two problems because solving the traffic assignment problem by the branch and bound algorithm is faster than solving it by GA
- [ ] The traffic assignment problem works as the fitness function of the GA to solve the network design problem
- [ ] The NDP works as a fitness function of the GA to solve the traffic assignment problem:
- [ ] The objective function is not quadratic anymore after disentangling the traffic assignment from the network design problem

Check

# Summarizing

The following is a diagram that shows what you are finally doing to solve the same problem but with a meta-heuristic approach:

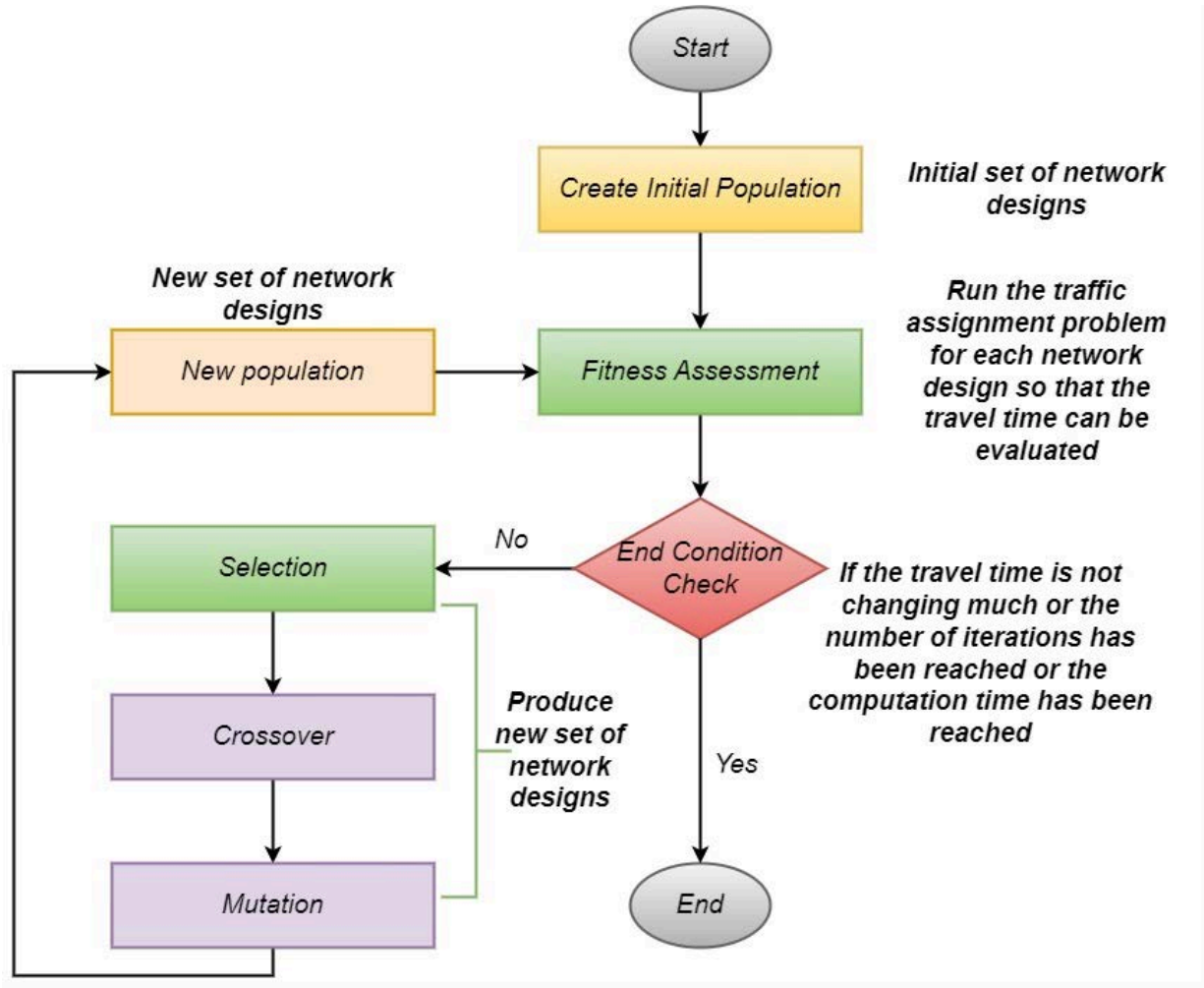



*Fig. 5.50* Summary meta-heurstic approach

# PyMOO

PyMOO is a Python library that provides a comprehensive and easy-to-use framework for multi-objective optimization (MOO). For this case, we are going to deal with only one objective; nevertheless, this is a useful tool if you have more objectives. In addition, PyMOO easily allows us to define our optimization problem by specifying the objectives, constraints, and decision variables.

# Implementation in Python

The implementation in Python is shown in the notebooks for MILP and GA. The full notebooks including source data are provided here

We can now start solving our problem!

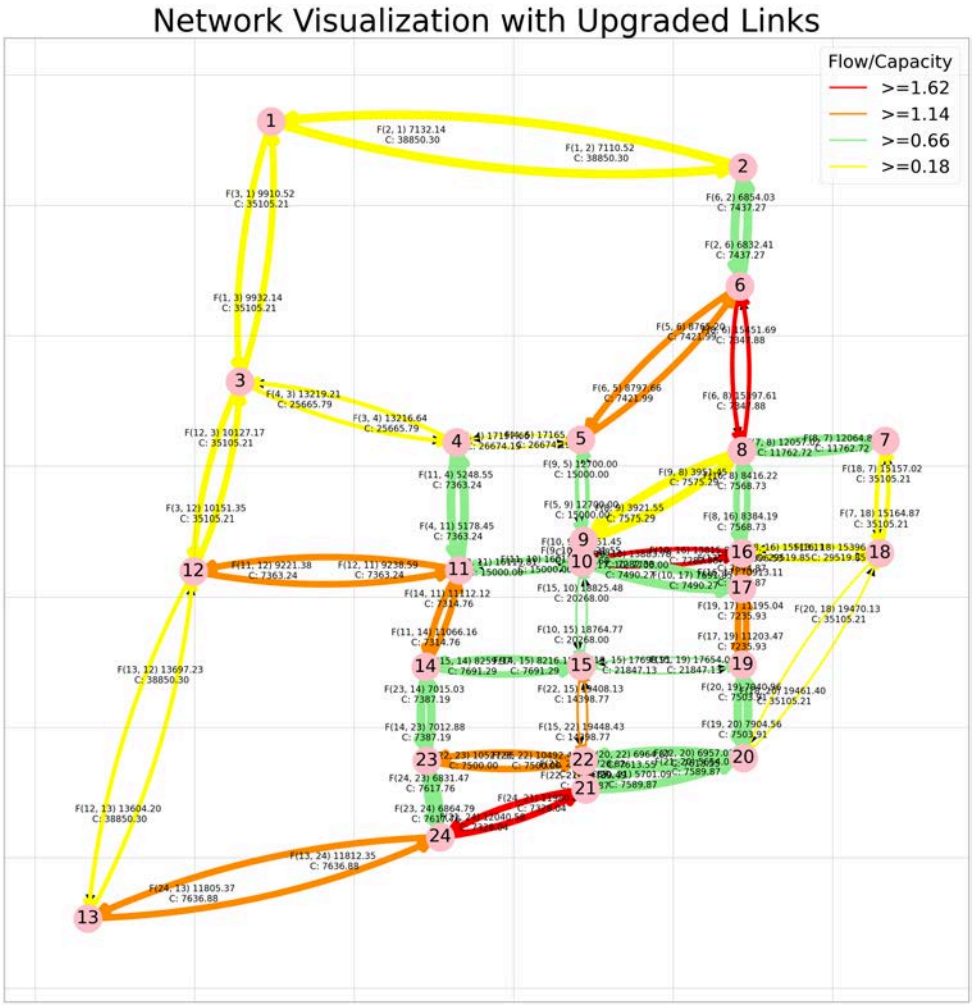



*Fig. 5.51* A solution of the problem

> **Attribution**
>
> This chapter is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi. Find out more here.

# Python implementation with mixed integer linear program

## Contents


- Data preprocessing
- Modeling in Gurobi
- Solving the model
- Analysis


**Note**

Gurobi cannot be loaded in this online book, so download this notebook to work on it with Gurobi locally installed.Instruction on how to do that are in the `README.md` and `PA_2_4_A_gurobilicious.ipynb` of week 2.4.

**MUDE Exam Information**

The road network design problem serves as an example for solving a complex optimization problem. You're not expected to understand the problem details for the exam.

## Data preprocessing

We use some networks from the well-known transportation networks for benchmarking repository as well as a small toy network for case studies of NDPs. The following functions read data from this repository and perform data preprocessing to have the input and the parameters required for our case studies.

```
# import required packages
import os
import time
import gurobipy as gp
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

```python
# read network file, a function to import the road networks
def read_net(net_file):
    """
       read network file
    """

    net_data = pd.read_csv(net_file, skiprows=8, sep='\t')
    # make sure all headers are lower case and without trailing spaces
    trimmed = [s.strip().lower() for s in net_data.columns]
    net_data.columns = trimmed
    # And drop the silly first and last columns
    net_data.drop(['~', ';'], axis=1, inplace=True)

    # make sure everything makes sense (otherwise some solvers throw errors)
    net_data.loc[net_data['free_flow_time'] <= 0, 'free_flow_time'] = 1e-6
    net_data.loc[net_data['capacity'] <= 0, 'capacity'] = 1e-6
    net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
    net_data.loc[net_data['power'] <= 1, 'power'] = int(4)
    net_data['init_node'] = net_data['init_node'].astype(int)
    net_data['term_node'] = net_data['term_node'].astype(int)
    net_data['b'] = net_data['b'].astype(float)

    # extract features in dict format
    links = list(zip(net_data['init_node'], net_data['term_node']))
    caps = dict(zip(links, net_data['capacity']))
    fftt = dict(zip(links, net_data['free_flow_time']))
    lent = dict(zip(links, net_data['length']))
    alpha = dict(zip(links, net_data['b']))
    beta = dict(zip(links, net_data['power']))

    net = {'capacity': caps, 'free_flow': fftt, 'length': lent, 'alpha': alpha, 'beta': b

    return net


# read OD matrix (demand), a function to import Origin and Destination Matrices,
# that is a table that says how many vehicles go from i to j
def read_od(od_file):
    """
       read OD matrix
    """

    f = open(od_file, 'r')
    all_rows = f.read()
    blocks = all_rows.split('Origin')[1:]
    matrix = {}
    for k in range(len(blocks)):
        orig = blocks[k].split('\n')
        dests = orig[1:]
        origs = int(orig[0])

        d = [eval('{' + a.replace(';', ',').replace(' ', '') + '}') for a in dests]
        destinations = {}
        for i in d:
            destinations = {**destinations, **i}
        matrix[origs] = destinations
    zones = max(matrix.keys())
    od_dict = {}
    for i in range(zones):
        for j in range(zones):
            demand = matrix.get(i + 1, {}).get(j + 1, 0)
            if demand:
                od_dict[(i + 1, j + 1)] = demand
```

```
            else:
                od_dict[(i + 1, j + 1)] = 0

    return od_dict


# read case study data, we will have different case studies that have different demand an
def read_cases(networks, input_dir):
    """
       read case study data
    """

    # dictionaries for network and OD files
    net_dict = {}
    ods_dict = {}

    # selected case studies
    if networks:
        cases = [case for case in networks]
    else:
        # all folders available (each one for one specific case)
        cases = [x for x in os.listdir(input_dir) if os.path.isdir(os.path.join(input_dir

    # iterate through cases and read network and OD
    for case in cases:
        mod = os.path.join(input_dir, case)
        mod_files = os.listdir(mod)
        for i in mod_files:
            # read network
            if i.lower()[-8:] == 'net.tntp':
                net_file = os.path.join(mod, i)
                net_dict[case] = read_net(net_file)
            # read OD matrix
            if 'TRIPS' in i.upper() and i.lower()[-5:] == '.tntp':
                ods_file = os.path.join(mod, i)
                ods_dict[case] = read_od(ods_file)

    return net_dict, ods_dict


# create node-destination demand matrix
def create_nd_matrix(ods_data, origins, destinations, nodes):
    # create node-destination demand matrix (not a regular OD!)
    demand = {(n, d): 0 for n in nodes for d in destinations}
    for r in origins:
        for s in destinations:
            if (r, s) in ods_data:
                demand[r, s] = ods_data[r, s]
    for s in destinations:
        demand[s, s] = - sum(demand[j, s] for j in origins)

    return demand
```

Now that we have the required functions for reading and processing the data, let's define some problem parameters and prepare the input.

```
```

```
# define parameters, case study (network) list and the directory where their files are
extension_factor = 2  # capacity after extension (1.5 means add 50% of existing capacity)
extension_max_no = 40  # max number of links to add capacity to (simplified budget limit)
timelimit = 300  # seconds
beta = 2  # parameter to use in link travel time function (explained later)

networks_dir = 'input/TransportationNetworks'


# prep data
net_dict, ods_dict = read_cases(networks, networks_dir)
# Let's load the network and demand (OD matrix) data of the first network (SiouxFalls) to
# Reminding that we are using the SiouxFalls network which is one of the most used networ
net_data, ods_data = net_dict[networks[0]], ods_dict[networks[0]]

## now let's prepare the data in a format readable by gurobi

# prep links, nodes, and free flow travel times
links = list(net_data['capacity'].keys())
nodes = np.unique([list(edge) for edge in links])
fftts = net_data['free_flow']

# auxiliary parameters (dict format) to keep the problem linear (capacities as parameters
# this is the capacity of a road link in vehicles per hour without the expansion
cap_normal = {(i, j): cap for (i, j), cap in net_data['capacity'].items()}
#with the expansion
cap_extend = {(i, j): cap * extension_factor for (i, j), cap in net_data['capacity'].item

# origins and destinations
origs = np.unique([orig for (orig, dest) in list(ods_data.keys())])
dests = np.unique([dest for (orig, dest) in list(ods_data.keys())])

# demand in node-destination form
# an OD-matrix is built
demand = create_nd_matrix(ods_data, origs, dests, nodes)
```

```
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7368\2029668135.py:15: FutureWarning
  net_data.loc[net_data['free_flow_time'] <= 0, 'free_flow_time'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7368\2029668135.py:17: FutureWarning
  net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7368\2029668135.py:16: FutureWarning
  net_data.loc[net_data['capacity'] <= 0, 'capacity'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7368\2029668135.py:17: FutureWarning
  net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
```

# Visualize network

In this section, we'll initially visualize the network using the coordinates of the sample road network, 'SiouxFalls'. Later, we'll employ the same network topology to visualize the upgraded links.

Thankfully, Python offers a highly useful package for visualizing networks called **'networkx'**. We'll leverage some of its features, so:

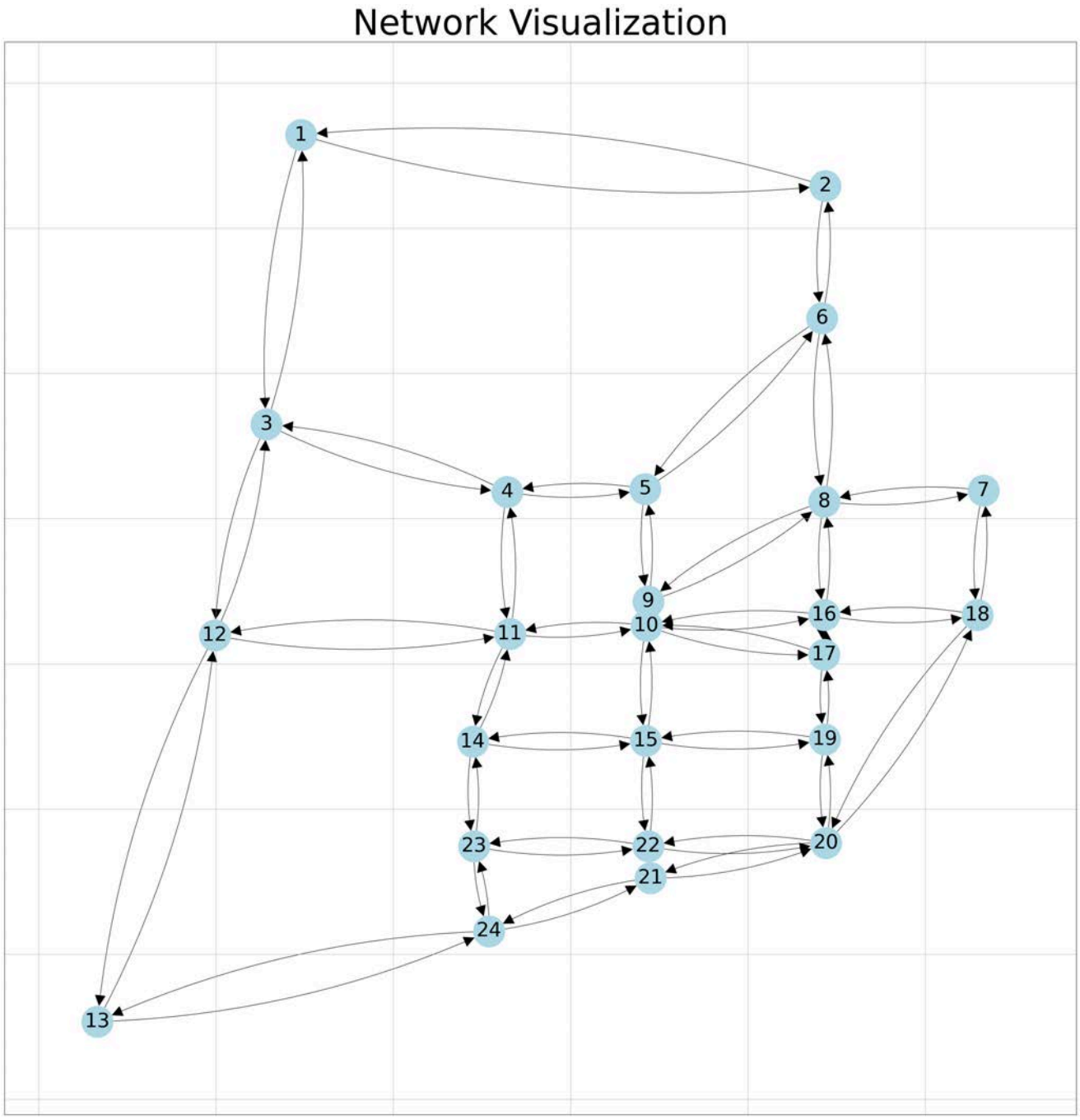
1. Feel free to explore further functionalities of the **networkx** package in its documentation.
2. Our coordinates in this specific example are in **JSON** format. Therefore, don't forget to import the package accordingly.
3. Interested in visualizing other networks? Fantastic! However, you'll need to check the format of your coordinates first. TAs will assist you if you wish to explore further.

The road network that we are using in the assessment is the Sioux Falls which is shown below:

```
# For visualization
import networkx as nx
import json
from matplotlib.lines import Line2D # this will later be used for highlighting edge color

from utils.network_visualization import network_visualization
from utils.network_visualization_highlight_link import network_visualization_highlight_li
from utils.network_visualization_upgraded import network_visualization_upgraded
```

```
coordinates_path = 'input/TransportationNetworks/SiouxFalls/SiouxFallsCoordinates.geojson
```

```
G, pos = network_visualization(link_flow = fftts,coordinates_path= coordinates_path) # th
```




# OD Matrix

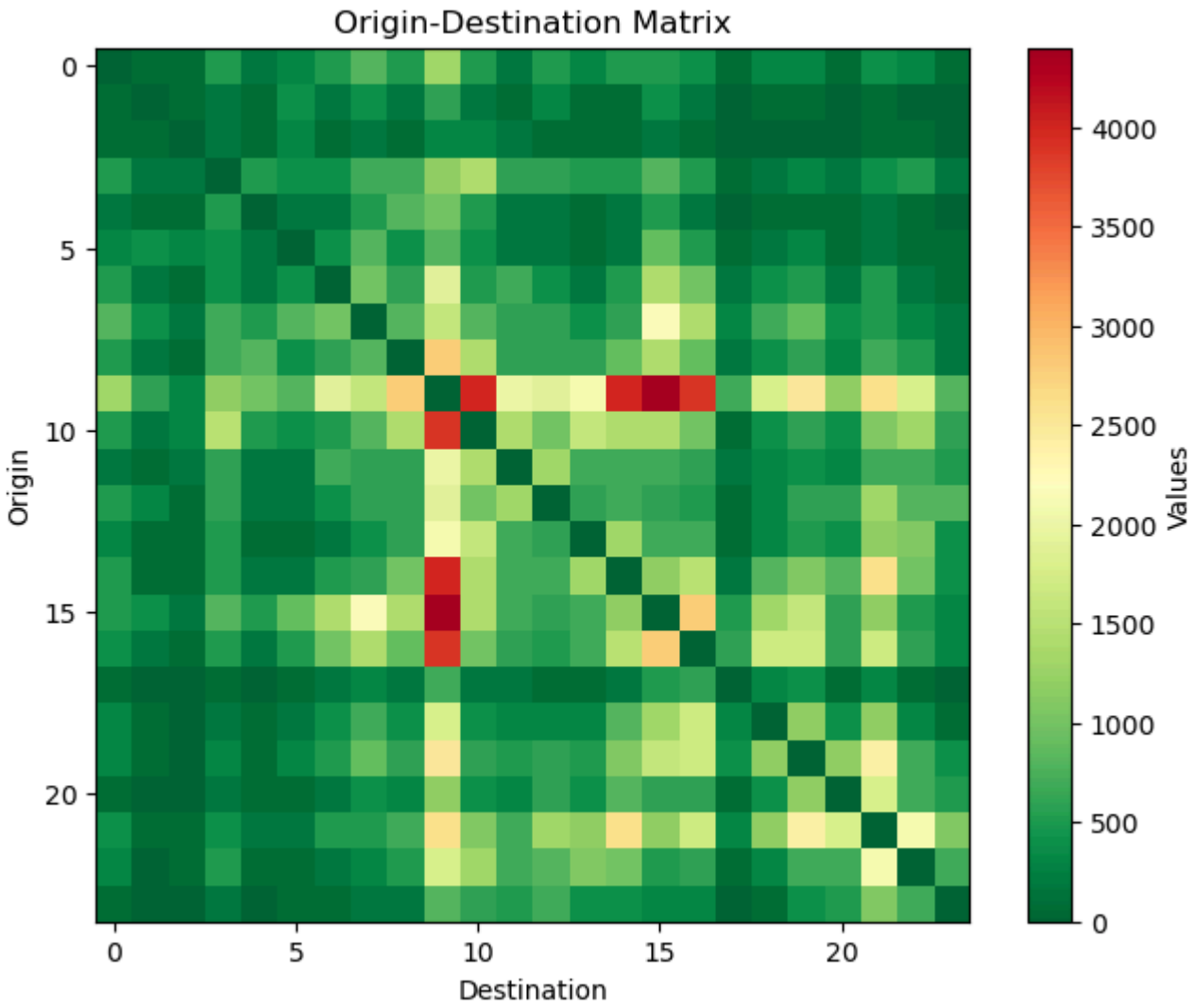
The trips per hour matrix for this network is:

This table is what is called an OD (Origin-Destination) matrix. It tells you how many cars go from node i to node j. The paths are chosen by solving the optimization model.

```
# Extracting the maximum values for dimensions
data = ods_data
max_origin = max(key[0] for key in data.keys())
max_destination = max(key[1] for key in data.keys())

# Creating a DataFrame to represent the matrix
od_matrix = pd.DataFrame(index=range(1, max_origin + 1), columns=range(1, max_destination

# Populating the DataFrame with the given data
for key, value in data.items():
    od_matrix.loc[key[0], key[1]] = value

# Displaying the OD matrix in table format
print("Origin-Destination Matrix:")
print(od_matrix.head(5)) # OD matric for the first 5 nodes.
```

```
Origin-Destination Matrix:
      1      2      3      4      5      6      7      8      9       10  ...  \
1      0  100.0  100.0  500.0  200.0  300.0  500.0  800.0  500.0  1300.0  ...
2  100.0      0  100.0  200.0  100.0  400.0  200.0  400.0  200.0   600.0  ...
3  100.0  100.0      0  200.0  100.0  300.0  100.0  200.0  100.0   300.0  ...
4  500.0  200.0  200.0      0  500.0  400.0  400.0  700.0  700.0  1200.0  ...
5  200.0  100.0  100.0  500.0      0  200.0  200.0  500.0  800.0  1000.0  ...

      15     16     17     18     19     20     21     22     23     24
1  500.0  500.0  400.0  100.0  300.0  300.0  100.0  400.0  300.0  100.0
2  100.0  400.0  200.0      0  100.0  100.0      0  100.0      0      0
3  100.0  200.0  100.0      0      0      0      0  100.0  100.0      0
4  500.0  800.0  500.0  100.0  200.0  300.0  200.0  400.0  500.0  200.0
5  200.0  500.0  200.0      0  100.0  100.0  100.0  200.0  100.0      0

[5 rows x 24 columns]
```

```
# To better understand the OD matrix we can also visualize the values.
# Creating a subset matrix for visualization
subset_matrix = np.zeros((24, 24))

# Filling subset matrix with data
for i in range(1, 25):
    for j in range(1, 25):
        subset_matrix[i-1, j-1] = data[(i, j)]

# Plotting the heatmap
plt.figure(figsize=(8, 6))
plt.title('Origin-Destination Matrix')
plt.xlabel('Destination')
plt.ylabel('Origin')
plt.imshow(subset_matrix, cmap='RdYlGn_r', interpolation='nearest')
plt.colorbar(label='Values')
plt.show()
```




# Modeling in Gurobi

## Initiate the Gurobi model

First, let's build a Gurobi model object and define some parameters based on the model type. We have a mixed integer quadratic program (MIQP), that's because the objective function has a quadratic term, which we want to transform to a mixed integer linear program (MILP) and solve using the branch and bound method. We discuss the transformations from quadratic to linear when we introduce quadratic terms.

```
## create a gurobi model object
model = gp.Model()

# define some important parameters for solving the model with gurobi
model.params.TimeLimit = timelimit  # 300 seconds timelimit since it can take long to red
model.params.NonConvex = 2  # our problem is not convex as it is now, so we let gurobi kn
#about convexity in optimization: a convex function will be a continuous functin that wil
#(or maximum depending on the problem), therefore in any point that you starting searchin
#to search for that extreme point. Non-convex problems can have local minimum (or maximum
#of searching for the solution
model.params.PreQLinearize = 1 # useful parameter to ask gurobi to try to linearize non-l
```

```
Set parameter TimeLimit to value 300
Set parameter NonConvex to value 2
Set parameter PreQLinearize to value 1
```

# Decision variables

As you will see below in the code block, we have one extra set of variables called x2 (x square). This is to help Gurobi isolate quadratic terms and perform required transformations based on MCE to keep the problem linear. This is not part of your learning goals.

```
## decision variables:

# link selected (y_ij); i: a_node, j: b_node (selected links for capacity expansion)
link_selected = model.addVars(links, vtype=gp.GRB.BINARY, name='y')

# link flows (x_ij); i: a_node, j: b_node
link_flow = model.addVars(links, vtype=gp.GRB.CONTINUOUS, name='x')

# link flows per destination s (xs_ijs); i: a_node, j: b_node, s: destination
dest_flow = model.addVars(links, dests, vtype=gp.GRB.CONTINUOUS, name='xs')

# link flow square (x2_ij); i: a_node, j: b_node (dummy variable for handling quadratic t
link_flow_sqr = model.addVars(links, vtype=gp.GRB.CONTINUOUS, name='x2')
```

# Objective function

```
```

```
## objective function (total travel time)

# total travel time = sum (link flow * link travel time)
# link travel time = free flow travel time * (1 + (flow / capacity))
# capacity = selected links *  base capacity + other links *  extended capacity
# other links: 1 - selected links

#note that this equation allows the number of vehicles to be greater than the capacity, t

model.setObjective(
    gp.quicksum(fftts[i, j] * link_flow[i, j] +
                fftts[i, j] * (beta/cap_normal[i, j]) * link_flow_sqr[i, j] -
                fftts[i, j] * (beta/cap_normal[i, j]) * link_flow_sqr[i, j] * link_select
                fftts[i, j] * (beta/cap_extend[i, j]) * link_flow_sqr[i, j] * link_select
                for (i, j) in links))
```

# Constraints

We have four sets of constraints for this problem. Let's go through them one by one and add them to the model.

## 1. Budget constraint

```
## constraints

# budget constraints, c_bgt is the name of the constraint
c_bgt = model.addConstr(gp.quicksum(link_selected[i, j] for (i, j) in links) <= extension
```

## 2. Link flow conservation constraints

```
# link flow conservation (destination flows and link flows), c_lfc is the name of the con
c_lfc = model.addConstrs(gp.quicksum(dest_flow[i, j, s] for s in dests) == link_flow[i, j
```

## 3. Node flow conservation constraints

```
# node flow conservation (demand), c_nfc is the name of the constraint
c_nfc = model.addConstrs(
    gp.quicksum(dest_flow[i, j, s] for j in nodes if (i, j) in links) -
    gp.quicksum(dest_flow[j, i, s] for j in nodes if (j, i) in links) == demand[i, s]
    for i in nodes for s in dests
)
```

## 4. Quadratic variable constraints (you do not need to fully understand this)

```
# dummy constraints for handling quadratic terms
c_qrt = model.addConstrs(link_flow_sqr[i, j] == link_flow[i, j] * link_flow[i, j] for (i,
```

# Solving the model

```
3#Next we are ready to solve the model
model.optimize()
```

▶ **Show code cell output**

Note that if you didn't find a solution, you can rerun the previous cell to continue the optimization for another 300 seconds (defined by `timelimit`).

# Analysis

```
# fetch optimal decision variables and Objective Function values
link_flows = {(i, j): link_flow[i, j].X for (i, j) in links}
links_selected = {(i, j): link_selected[i, j].X for (i, j) in links}
total_travel_time = model.ObjVal

# Let's print right now the objective function
print("Optimal Objective function Value", model.objVal)

# Let's print right now the decision variables
for var in model.getVars():
    print(f"{var.varName}: {round(var.X, 3)}")  # print the optimal decision variable val
```

▶ **Show code cell output**

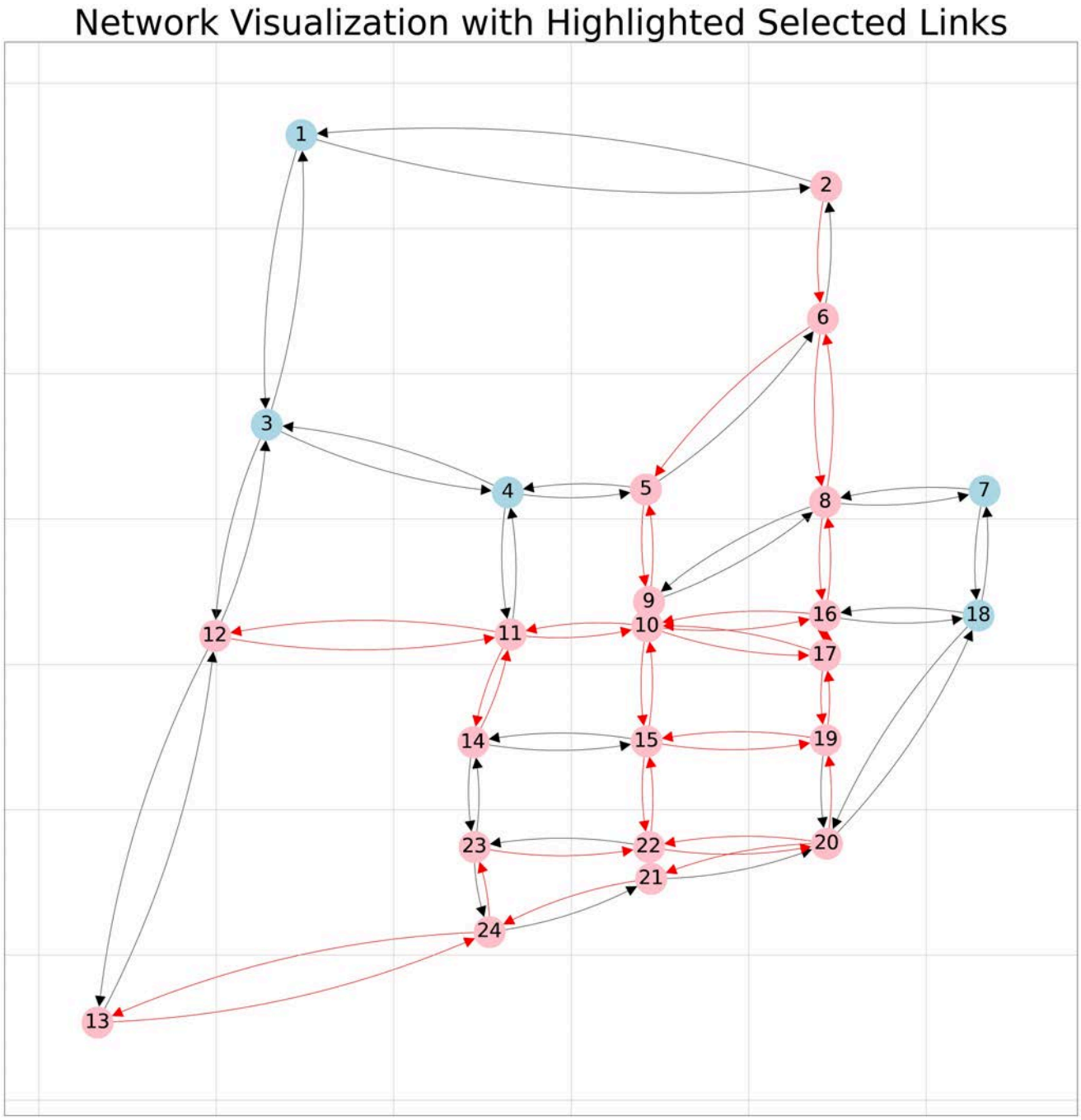
# Network Visualization

## 1. Network visualization with highlighted links

Now, let's visualize the results showcasing congested traffic flows and the selected links for expansion. In this graph, we'll observe the network's topology using node coordinates and links. Our graph will be **directional** to represent the road network.

Nodes are depicted in blue, while selected nodes and links are highlighted in pink and red.

```
network_visualization_highlight_links (G, pos, link_select=links_selected)
```




## 2. Network visualization with upgraded links

In this section, we'll visualize our upgraded network, incorporating the new capacities. Following that, we'll represent the network along with its results, displaying the flow (F) and capacity (C) alongside each link.

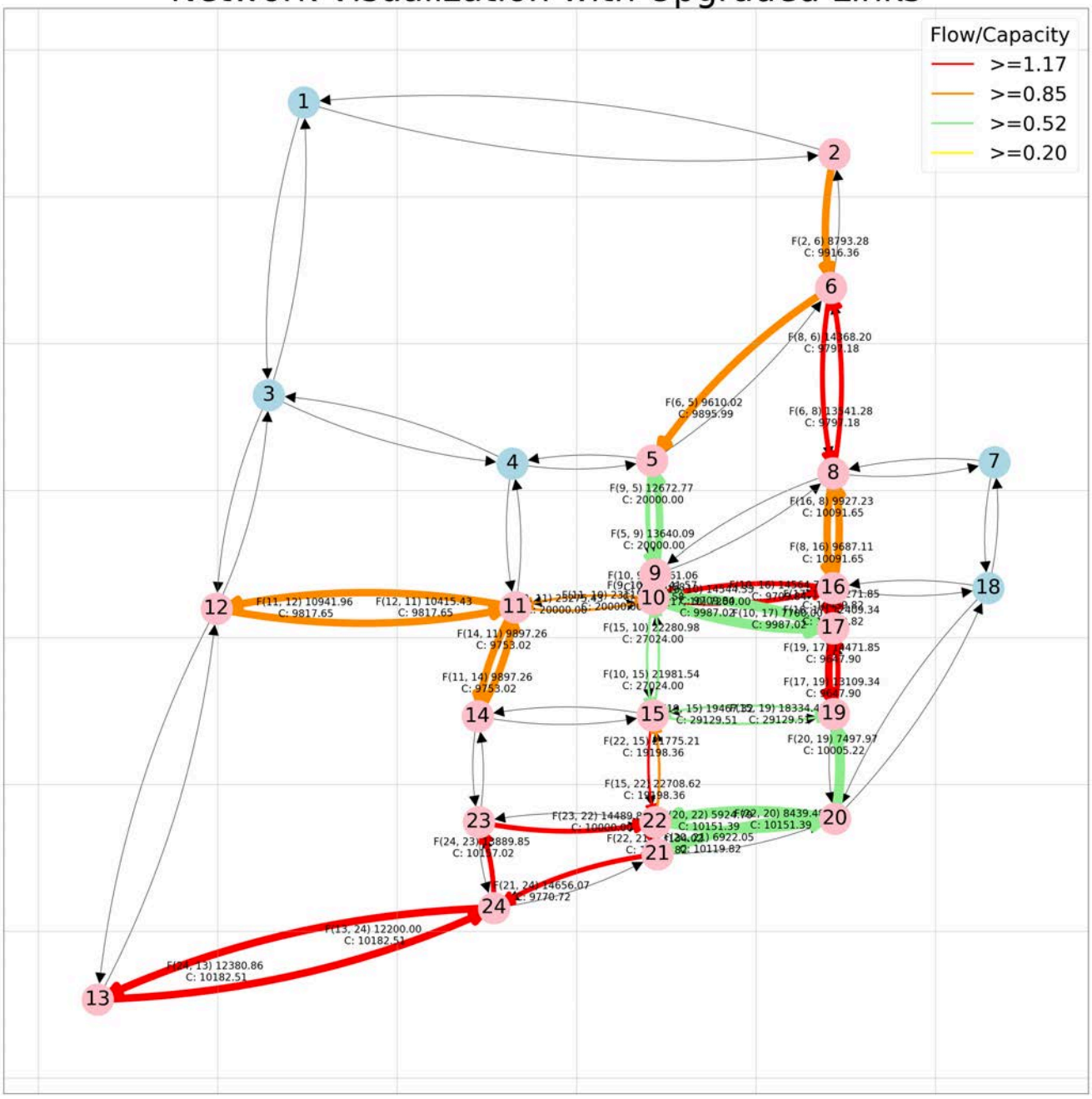
**Notes:**

1. **Pink nodes** highlight the selected nodes.

2. **Colored edges** denote the upgraded edges selected through the optimization process.
3. Various **edge colors** indicate different ranges for edge attributes (Flow/Capacity), as demonstrated in the legend.
4. Diverse **edge widths** represent varying flow ranges on the edges.
5. Your plot is interactive; to clearly view the numbers, simply click on them!

```
# Define new capacity after expansion
cap_normal = {(i, j): cap for (i, j), cap in net_data['capacity'].items()}
cap_extend = {(i, j): cap * extension_factor for (i, j), cap in net_data['capacity'].item
capacity = {(i, j): cap_normal[i, j] * (1 - links_selected[i, j]) + cap_extend[i, j] * li
            for (i, j) in links}
```

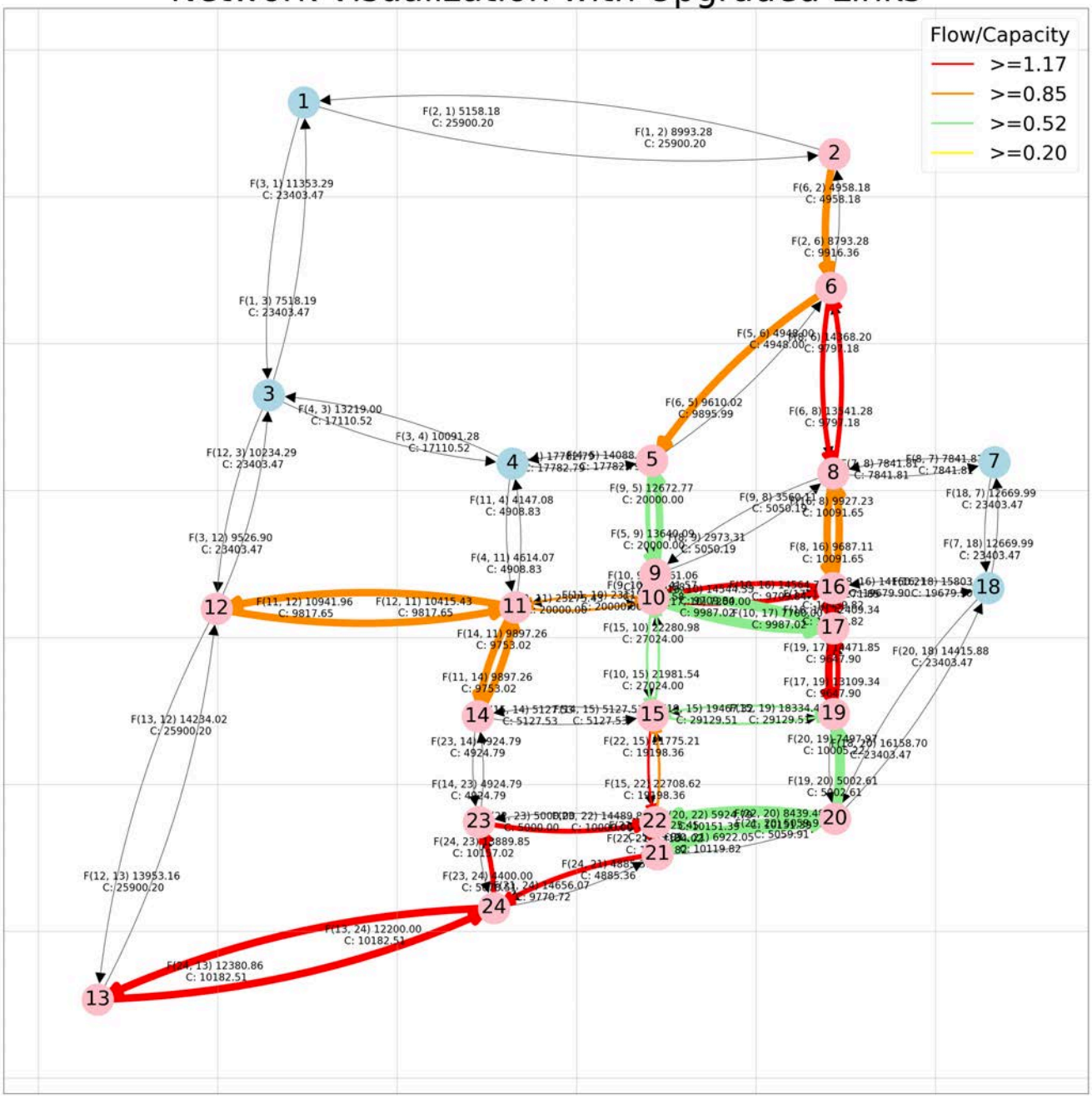
```
# Plot results
network_visualization_upgraded (G = G, pos=pos, link_flow=link_flows, capacity_new=capaci
```




If you wish to see the velues for the entire network you just need to turn on the labels and run the function again.

```
# To see flow and capacity for the entire network
network_visualization_upgraded (G = G, pos=pos, link_flow=link_flows, capacity_new=capaci
```

```
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_7368\4155739783.py:69: DeprecationWa
  nx.draw_networkx_edges(subgraph, pos,edgelist=edge_to_color.keys(), width=edge_widths,
```




> **Attribution**
>
> This chapter reuses content Bahman Ahmadi et al. (2024). Find out more here.

# Python implementation with genetic algoritm

## Contents


- Data preprocessing
- Modeling and solving the traffic assignment sub-problem with Gurobi
- Modeling with PyMOO
- Solve the problem
- Convergence curve
- Network Visualization


**Note**

Gurobi cannot be loaded in this online book, so download this notebook to work on it with Gurobi locally installed. You can make use of environment which is provided with the code and dataset, make sure you activate your Gurobi license. Instruction on how to do that are in the `README.md` and `PA_2_4_A_gurobilicious.ipynb` of week 2.4.

**MUDE Exam Information**

The road network design problem serves as an example for solving a complex optimization problem. You're not expected to understand the problem details for the exam.

## Data preprocessing

Our data preprocessing steps are similar to the previous notebook. We use some networks from the well-known transportation networks for benchmarking repository as well as a small toy network for case studies of NDPs. The following functions read data from this repository

and perform data preprocessing to have the input and the parameters required for our case studies.

```
import pandas as pd
import numpy as np
import gurobipy as gp
import matplotlib.pyplot as plt

# Genetic algorithm dependencies. We are importing the pymoo functions that are imporant
from pymoo.algorithms.soo.nonconvex.ga import GA
from pymoo.core.problem import ElementwiseProblem
from pymoo.optimize import minimize
from pymoo.operators.sampling.rnd import BinaryRandomSampling
from pymoo.operators.crossover.hux import HalfUniformCrossover
from pymoo.operators.mutation.bitflip import BitflipMutation

# not used here but generally useful dependencies
#from pymoo.core.problem import Problem
#from pymoo.operators.mutation.pm import PolynomialMutation
#from pymoo.operators.crossover.pntx import PointCrossover
#from pymoo.operators.crossover.sbx import SBX
#from pymoo.operators.crossover.sbx import SimulatedBinaryCrossover
#from pymoo.operators.crossover.pntx import Crossover
#from pymoo.operators.repair.bounds_repair import BoundsRepair
#from pymoo.core.repair import Repair
```

```
# For visualization
from utils.network_visualization import network_visualization
from utils.network_visualization_highlight_link import network_visualization_highlight_li
from utils.network_visualization_upgraded import network_visualization_upgraded
```

```
# import required packages
import os
import time

# read network file
def read_net(net_file):
    """
       read network file
    """

    net_data = pd.read_csv(net_file, skiprows=8, sep='\t')
    # make sure all headers are lower case and without trailing spaces
    trimmed = [s.strip().lower() for s in net_data.columns]
    net_data.columns = trimmed
    # And drop the silly first and last columns
    net_data.drop(['~', ';'], axis=1, inplace=True)

    # make sure everything makes sense (otherwise some solvers throw errors)
    net_data.loc[net_data['free_flow_time'] <= 0, 'free_flow_time'] = 1e-6
    net_data.loc[net_data['capacity'] <= 0, 'capacity'] = 1e-6
    net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
    net_data.loc[net_data['power'] <= 1, 'power'] = int(4)
    net_data['init_node'] = net_data['init_node'].astype(int)
    net_data['term_node'] = net_data['term_node'].astype(int)
    net_data['b'] = net_data['b'].astype(float)

    # extract features in dict format
    links = list(zip(net_data['init_node'], net_data['term_node']))
    caps = dict(zip(links, net_data['capacity']))
    fftt = dict(zip(links, net_data['free_flow_time']))
    lent = dict(zip(links, net_data['length']))
    alpha = dict(zip(links, net_data['b']))
    beta = dict(zip(links, net_data['power']))

    net = {'capacity': caps, 'free_flow': fftt, 'length': lent, 'alpha': alpha, 'beta': b

    return net


# read OD matrix (demand)
def read_od(od_file):
    """
       read OD matrix
    """

    f = open(od_file, 'r')
    all_rows = f.read()
    blocks = all_rows.split('Origin')[1:]
    matrix = {}
    for k in range(len(blocks)):
        orig = blocks[k].split('\n')
        dests = orig[1:]
        origs = int(orig[0])

        d = [eval('{' + a.replace(';', ',').replace(' ', '') + '}') for a in dests]
        destinations = {}
        for i in d:
            destinations = {**destinations, **i}
        matrix[origs] = destinations
    zones = max(matrix.keys())
    od_dict = {}
    for i in range(zones):
        for j in range(zones):
```

```
            demand = matrix.get(i + 1, {}).get(j + 1, 0)
            if demand:
                od_dict[(i + 1, j + 1)] = demand
            else:
                od_dict[(i + 1, j + 1)] = 0

    return od_dict


# read case study data
def read_cases(networks, input_dir):
    """
       read case study data
    """

    # dictionaries for network and OD files
    net_dict = {}
    ods_dict = {}

    # selected case studies
    if networks:
        cases = [case for case in networks]
    else:
        # all folders available (each one for one specific case)
        cases = [x for x in os.listdir(input_dir) if os.path.isdir(os.path.join(input_dir

    # iterate through cases and read network and OD
    for case in cases:
        mod = os.path.join(input_dir, case)
        mod_files = os.listdir(mod)
        for i in mod_files:
            # read network
            if i.lower()[-8:] == 'net.tntp':
                net_file = os.path.join(mod, i)
                net_dict[case] = read_net(net_file)
            # read OD matrix
            if 'TRIPS' in i.upper() and i.lower()[-5:] == '.tntp':
                ods_file = os.path.join(mod, i)
                ods_dict[case] = read_od(ods_file)

    return net_dict, ods_dict


# create node-destination demand matrix
def create_nd_matrix(ods_data, origins, destinations, nodes):
    # create node-destination demand matrix (not a regular OD!)
    demand = {(n, d): 0 for n in nodes for d in destinations}
    for r in origins:
        for s in destinations:
            if (r, s) in ods_data:
                demand[r, s] = ods_data[r, s]
    for s in destinations:
        demand[s, s] = - sum(demand[j, s] for j in origins)

    return demand
```

Now that we have the required functions for reading and processing the data, let’s define some problem parameters and prepare the input.

```
```

```
# define parameters, case study (network) list and the directory where their files are
extension_factor = 1.5  # capacity after extension (1.5 means add 50%)
extension_max_no = 20  # the number of links to add capacity to (simplified way of reprse
#it's the same to say that it's exactly this number of that this number is the max, that'
#even if just one car circulates.
timelimit = 300 # seconds
beta = 2  # parameter to use in link travel time function (explained later)

networks = ['SiouxFalls']
networks_dir = os.getcwd() +'/input/TransportationNetworks'


# prep data
net_dict, ods_dict = read_cases(networks, networks_dir)

# Let's load the network and demand (OD matrix) data of the first network (Sioux Falls) t
# WE USE THE SAME NETWORK FROM THE FIRST NOTEBOOK: SIOUX FALLS
# The network has 76 arcs in total
net_data, ods_data = net_dict[networks[0]], ods_dict[networks[0]]

## now let's prepare the data in a format readable by gurobi

# prep links, nodes, and free flow travel times
links = list(net_data['capacity'].keys())
nodes = np.unique([list(edge) for edge in links])
fftts = net_data['free_flow']

# auxiliary parameters (dict format) to keep the problem linear (capacities as parameters
cap_normal = {(i, j): cap for (i, j), cap in net_data['capacity'].items()}
cap_extend = {(i, j): cap * extension_factor for (i, j), cap in net_data['capacity'].item

# origins and destinations
dests = np.unique([dest for (orig, dest) in list(ods_data.keys())])
origs = np.unique([orig for (orig, dest) in list(ods_data.keys())])

# demand in node-destination form
demand = create_nd_matrix(ods_data, origs, dests, nodes)
```

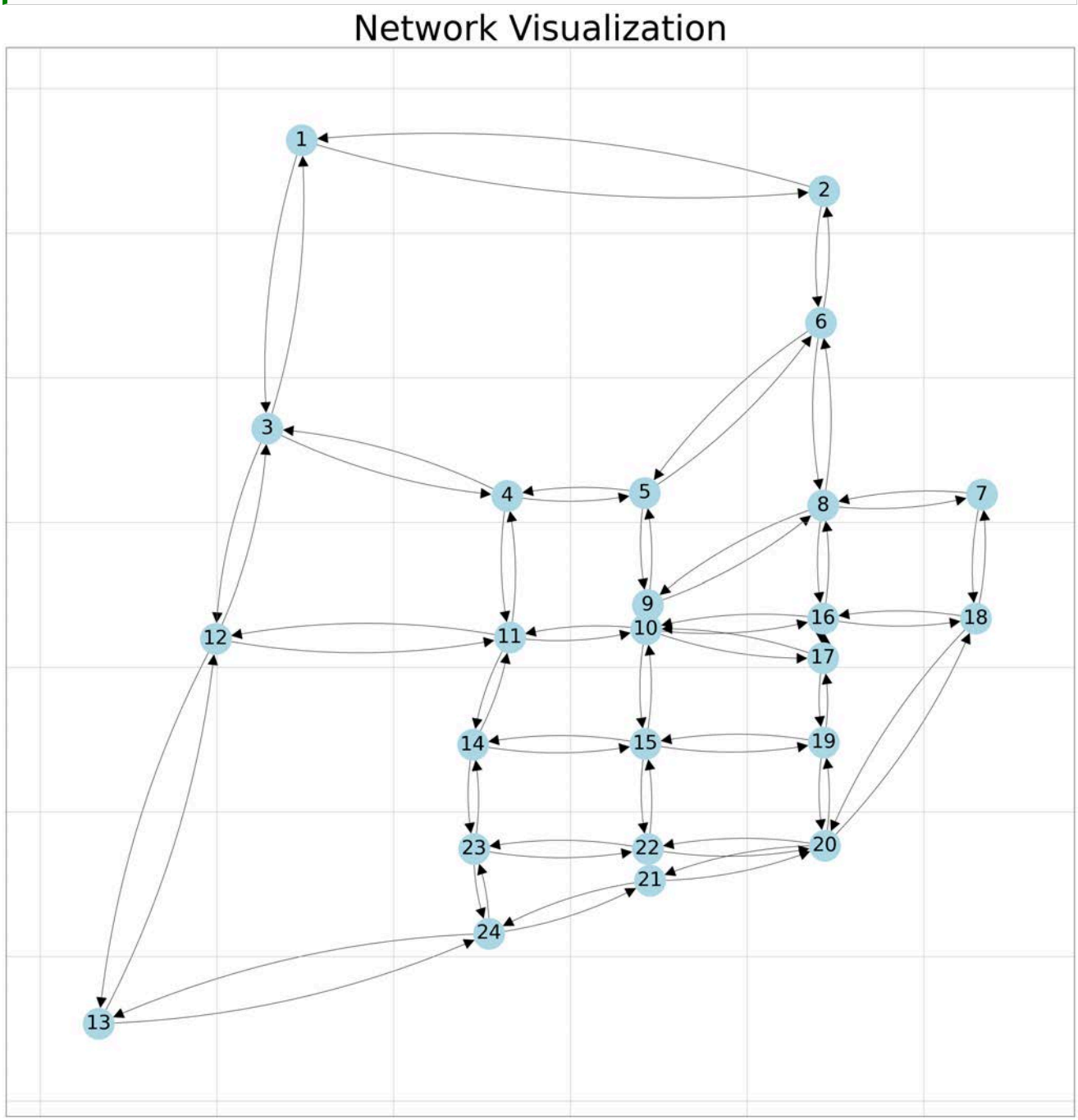
```
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_26868\123191696.py:19: FutureWarning
  net_data.loc[net_data['free_flow_time'] <= 0, 'free_flow_time'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_26868\123191696.py:21: FutureWarning
  net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_26868\123191696.py:20: FutureWarning
  net_data.loc[net_data['capacity'] <= 0, 'capacity'] = 1e-6
C:\Users\tomvanwoudenbe\AppData\Local\Temp\ipykernel_26868\123191696.py:21: FutureWarning
  net_data.loc[net_data['length'] <= 0, 'length'] = 1e-6
```

# Network Display

We will use the same function we used in the previous notebook to visualize the network.

```
coordinates_path = 'input/TransportationNetworks/SiouxFalls/SiouxFallsCoordinates.geojson
```

```
G, pos = network_visualization(link_flow = fftts,coordinates_path= coordinates_path) # th
```




Now we are ready to build our models!

# Modeling and solving the traffic assignment sub-problem with Gurobi

In this section we build a Gurobi model to solve the Traffic Assignment sub-problems. The decision variables, objective function, and the constraints of this problem were described before. Here we wrap the code in a function so that we can use it later within the GA.

```python
def ta_qp(dvs, net_data=net_data, ods_data=ods_data, extension_factor=1.5):

    # prep variables
    beta = 2
    links = list(net_data['capacity'].keys())
    nodes = np.unique([list(edge) for edge in links])
    fftts = net_data['free_flow']
    links_selected = dict(zip(links, dvs))

    # define capacity
    cap_normal = {(i, j): cap for (i, j), cap in net_data['capacity'].items()}
    cap_extend = {(i, j): cap * extension_factor for (i, j), cap in net_data['capacity'].
    capacity = {(i, j): cap_normal[i, j] * (1 - links_selected[i, j]) + cap_extend[i, j]
                for (i, j) in links}

    dests = np.unique([dest for (orig, dest) in list(ods_data.keys())])
    origs = np.unique([orig for (orig, dest) in list(ods_data.keys())])

    # demand in node-destination form
    demand = create_nd_matrix(ods_data, origs, dests, nodes)

    ## create a gurobi model object
    model = gp.Model()
    # just to avoid cluttering the notebook with unnecessary logging output
    model.Params.LogToConsole = 0

    ## decision variables:

    # link flows (x_ij); i: a_node, j: b_node
    link_flow = model.addVars(links, vtype=gp.GRB.CONTINUOUS, name='x')

    # link flows per destination (xs_ijs); i: a_node, j: b_node, s: destination
    dest_flow = model.addVars(links, dests, vtype=gp.GRB.CONTINUOUS, name='xs')

    ## constraints

    # node flow conservation (demand)
    model.addConstrs(
        gp.quicksum(dest_flow[i, j, s] for j in nodes if (i, j) in links) -
        gp.quicksum(dest_flow[j, i, s] for j in nodes if (j, i) in links) == demand[i, s]
        for i in nodes for s in dests
    )

    # link flow conservation (destination flows and link flows)
    model.addConstrs(gp.quicksum(dest_flow[i, j, s] for s in dests) == link_flow[i, j] fo

    ## objective function (total travel time)
    # total travel time = sum (link flow * link travel time)
    # link travel time = free flow travel time * (1 + (flow / capacity))

    model.setObjective(
        gp.quicksum(link_flow[i, j] * (fftts[i, j] * (1 + (beta * link_flow[i, j]/capacit


    ## solve
    model.update()
    start_solve = time.time()
    model.optimize()
    solve_time = (time.time() - start_solve)

    # fetch optimal DV and OF values
    link_flows = {(i, j): link_flow[i, j].X for (i, j) in links}
    total_travel_time = model.ObjVal
```

```
    return total_travel_time, capacity, link_flows, links_selected
```

# Modeling with PyMOO

Let's define a model in MyMOO and deal with the links selection problem with the GA.

First, we need to define a problem class.

```
#If you want to know more about the library that is being used: https://pymoo.org/algorit

class NDP(ElementwiseProblem):

    def __init__(self, budget):

        super().__init__(n_var=len(links),       # number of decision variables (i.e., nu
                         n_obj=1,                # for now we use only one objective (tot
                         n_constr=1,             # one constraint for budget, that's beca
                         vtype=bool,             # binary decision variables
                        )
        self.budget = budget

    def _evaluate(self, decision_vars, out, *args, **kwargs):

        # call TA to calculate the objective fucntion, meaning to do the evaluation of th
        total_travel_time,capacity, link_flows, links_selected = ta_qp(decision_vars)

        # the budget constraint
        # In the GA part the only variables are the binary decision variables, don't forg
        # produces the travel time on the network is done in the evaluation of the soluti
        g = sum(decision_vars) - self.budget

        out["F"] = total_travel_time
        out["G"] = g
```

Now, let's initiate an instance of the problem based on the problem class we defined, and initiate the GA with its parameters. Note that depending on the problem size and the number of feasible links, you might need larger values for population and generation size to achieve good results or even feasible results. Of course this increases the computation times.

**Test yourself**

Select the right answer(s):

- [ ] This evolutionary method is great because if I wait enough time all solutions will have been tested and I make sure that I found the optimal solution
- [ ] The size of the population is the number of variables in my problem
- [ ] Sometimes some solutions tested in the algorithm have fractional variables and that's a problem because we can't cut roads in pieces.
- [ ] The individuals in the population represent different plans of road upgrade which in time should be better and better according to the objective function
- [ ] Later generations can be worst then the earlier generations

Check

```
Budget = 76
pop_size = 10

# initiate an instance of the problem with max number of selected links as budget constra
problem = NDP(budget=Budget)

# initiate the GA with parameters appropriate for binary variables
method = GA(pop_size=pop_size,
            sampling=BinaryRandomSampling(),
            mutation=BitflipMutation(),
            crossover=HalfUniformCrossover()
            )
```

# Solve the problem

Now we are ready to minimize the NDP problem using the GA method we defined.

```
opt_results = minimize(problem,
                method,
                termination=("time", "00:05:00"), #5 minute maximum computation time
                seed=7,
                save_history=True,
                verbose=True,
                )

print("Best Objective Function value: %s" % opt_results.F)
print("Constraint violation: %s" % opt_results.CV)
print("Best solution found: %s" % opt_results.X)

#To better interpret the results, this is the legend:
#n_gen:  Number of generations
#n_eval: Number of function evaluations
#cv_min: Minimum constraint violation
#cv_avg: Average constraint violation
#f_avg:  Average objective function value
#f_min:  Minimum objective function value
```

▶ **Show code cell output**

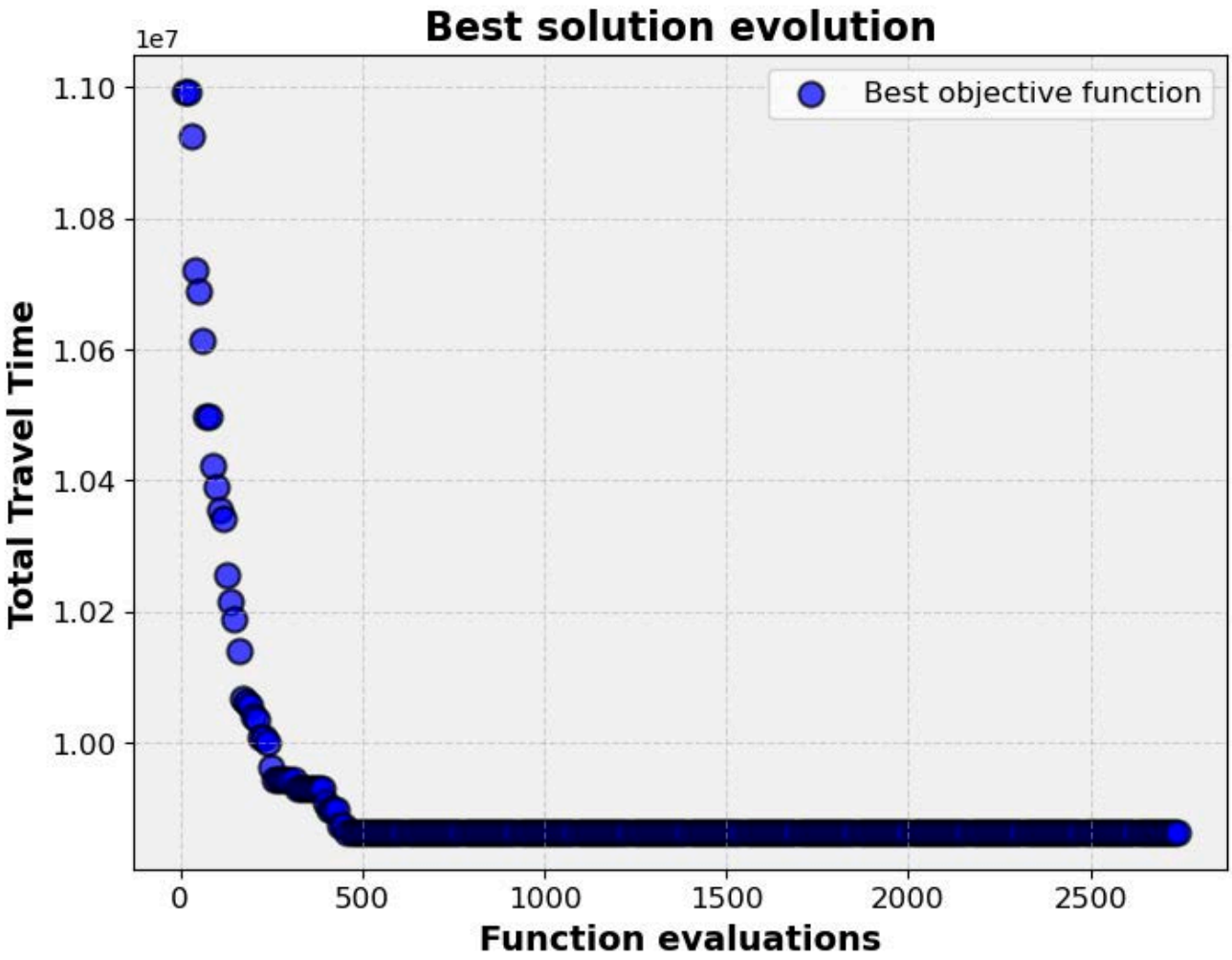
# Convergence curve

Let’s first define some functions (to use later) to get the results and plot them.

```
def get_results(opt_results):

    number_of_individuals = [] # The number of individuals in each generation
    optimal_values_along_generations = []  # The optimal value found in each generation

    for generation_status in opt_results.history:

        # retrieve the optimum from the algorithm
        optimum = generation_status.opt

        # filter out only the feasible solutions and append and objective space values
        try:
            feas = np.where(optimum.get("feasible"))[0]
            optimal_values_along_generations.append(optimum.get("F")[feas][0][0])
            # store the number of function evaluations
            number_of_individuals.append(generation_status.evaluator.n_eval)
        except:
            #In case a generation does not have any feasible solutions, it will be ignore
            pass

    return number_of_individuals, optimal_values_along_generations


def plot_results(number_of_individuals, optimal_values_along_generations):

    # Create a scatter plot with enhanced styling
    plt.figure(figsize=(8, 6))  # Set the figure size

    # Create a scatter plot
    plt.scatter(number_of_individuals, optimal_values_along_generations, label='Best obje

    # Add labels and a legend with improved formatting
    plt.xlabel('Function evaluations', fontsize=14, fontweight='bold')
    plt.ylabel('Total Travel Time', fontsize=14, fontweight='bold')
    plt.title('Best solution evolution', fontsize=16, fontweight='bold')
    plt.legend(loc='upper right', fontsize=12)

    # Customize the grid appearance
    plt.grid(True, linestyle='--', alpha=0.5)

    # Customize the tick labels
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # Add a background color to the plot
    plt.gca().set_facecolor('#f2f2f2')

    # Show the plot
    plt.show()
```

Now let's use these functions to plot the results.

```
number_of_individuals, optimal_values_along_generations = get_results(opt_results)

plot_results(number_of_individuals, optimal_values_along_generations)
```

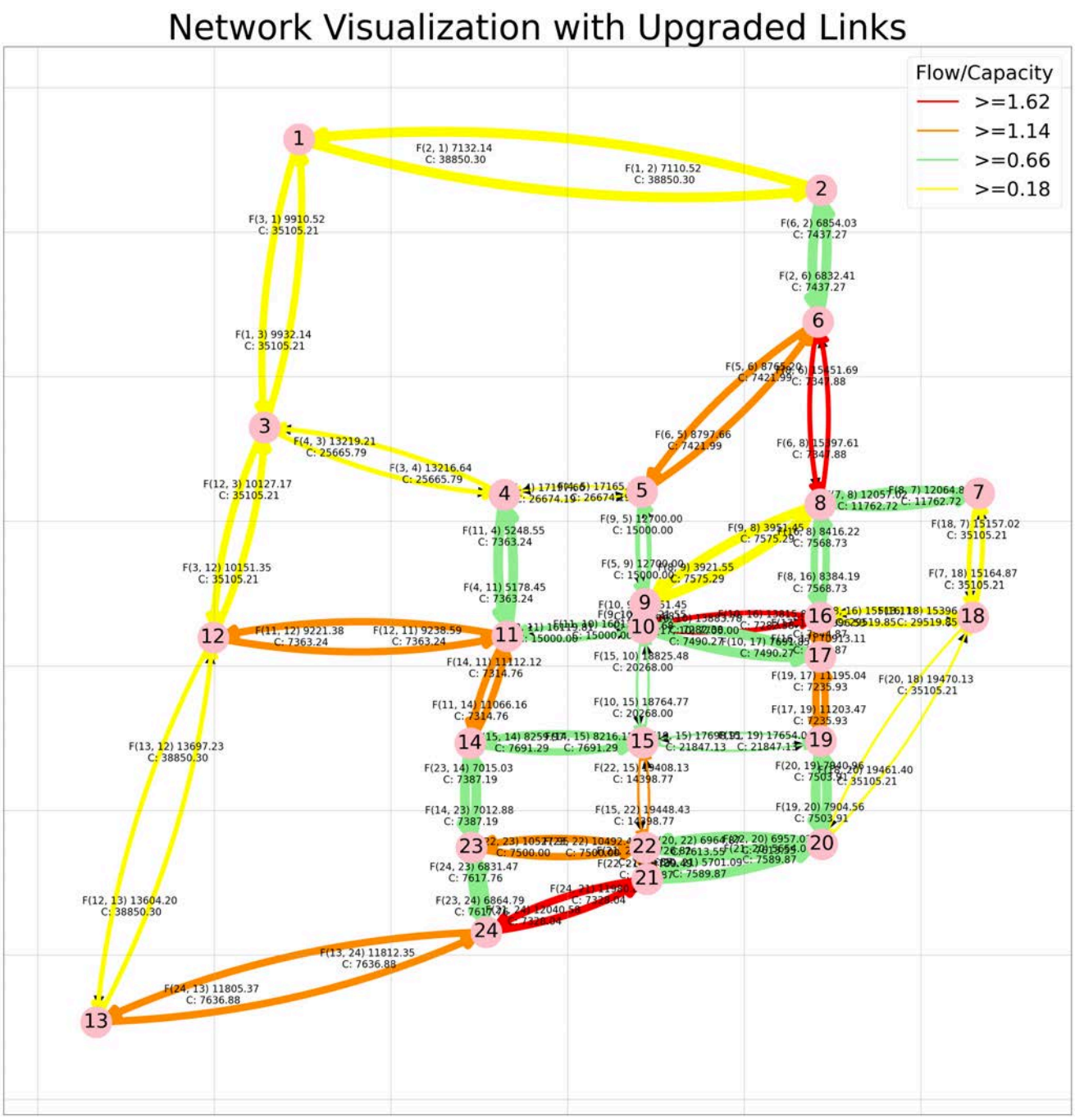



# Network Visualization

Same as the previous notebook we use link_flows, links_selected to visualize our results on the network.

```
travel_time, capacity, link_flows, links_selected= ta_qp(dvs=opt_results.X, net_data=net_
```

```
# Plot results
# To see the values for all the links just turn on the labels in the function below.
network_visualization_upgraded (G = G, pos=pos, link_flow=link_flows, capacity_new=capaci
```




> **Attribution**
>
> This chapter reuses content Bahman Ahmadi et al. (2024). Find out more here.

# Introduction to Machine Learning

> **Attribution**
>
> This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

*Machine learning* is a set of techniques that uses computer algorithms to fit the parameters of a model. In the context of MUDE, it is a set of data-driven modelling techniques, closely related to the methods used in observation theory. However, whereas in observation theory we constructed the design matrix ourselves, based on some physics-based knowledge of the system of interest, in machine learning applications, we allow the algorithm more freedom to build (and refine) the model. Although this comes at the cost of less control over model parameters (the "unknowns" from observation theory) and increased complexity, the benefit is a more flexible model that is capable of capturing key characteristics of the problem at hand. For example, a neural network can be defined using thousands of weights (model parameters); far more than we can create with more "hands-on" techniques!

The problem we would like to solve:

- Given: Some complex process $p(x, t)$, usually highly nonlinear
- Goal: Construct a model $y(x)$ that explains it
- In practice: We do not know $p(x, t)$, but only have $N$ observations of it

Typically, these techniques are well-suited to applications that have large data sets, which is required to fit many parameters. As such, the resulting model is generally better suited to *interpolation*, rather than *extrapolation.* In other words: better to use the model to evaluate situations "between" the data, rather than "outside" them.

The field of *machine learning* is multidisciplinary, which means it comes with a new set of terms and conceptual perspectives. Some examples are provided here:

- The observations of the process (i.e., the "answer") is often called the *target*

- The process of fitting a model to data (i.e., computing the model parameters) is called *training* the model
- Rather than simply fit a model to data and validate it (e.g., by computing and evaluating the residuals), it is standard practice to withhold a portion of the data set and use it to validate the model *after* the parameters have been computed. The concept of model *validation* itself is unchanged.
- When the data is partitioned into two sets for model validation, the set used to find the model parameters is referred to as the *training* set, whereas the remainder is the *validation* set.
- As the optimization of model parameters can computationally expensive for large data sets, incorporating iterative cycles during the *training* process can significantly improve the model and modelling process. This is referred to as *tuning* the model.
- It is common practice to create a *third* partition of the original data that can be used during the *tuning* process; this is referred to as the *test* set.
- Parameters that are used to control the model *training* process, and related characteristics of the model, are called *hyperparameters.* For example, the $k$ in the k-Nearest neighbors approach. These parameters are often what is found during the *validation* process, and can be improved during the *tuning* process.
- *Loss* is a term used for quantitative measures of model error. Although the mathematical definition will change for various model approaches and, also, when it is used in different phases of the modelling process (train, validate, test), the general use of *loss* as a measure of error remains unchanged.

**Contents**

The story in this set of chapters is told through videos, along with interactive pages to work with the methods directly.

Contents:

- Decision theory for regression, k-Nearest Neighbors estimator
- Linear regression with nonlinear basis functions
- Introduction to neural networks for regression

By the end of this series, you will be able to:

- Understand and compare different approaches for regression modeling
- Construct parsimonious regression models for general applications
- Critically assess model performance from a probabilistic standpoint

# Introduction and k-Nearest Neighbors

## Contents


- Introduction
- k-nearest neighbors
- Varying our model parameters
- Playing around with the plots
- Final remarks


This page contains interactive python element: click 🚀 –> **Live Code** in the top right corner to activate it.

## Introduction

A very general problem often encountered is predicting some target variable $t$ as a function of a predictor variable $x$, but we only have inaccurate measurements of our target variable.

Some examples of possible problems where we can apply this kind of framework are given below:

| **Problem** | **Target variable $t$** | **Predictor variable $x$** |
|---|---|---|
| Structural health monitoring of a bridge | Displacement | Location along the bridge |
| Fatigue testing of a steel specimen | Stress | Number of loading cycles |
| Flood control of the river Maas | Volumetric flow rate | Precipitation in the Ardennes |
| Radioactive decay of a radon-222 sample | Number of $\alpha$-particles emitted | Time |
| Cooling rate of my coffee cup | Temperature | Time |

To strip the problem down to its bare essentials, we will use a toy problem, where we actually know the ground truth $f(x)$. In our case, this will be a simple sine wave, of which we make 100 noisy measurements from $x = 0$ to $x = 2\pi$. Note that, in a practical setting, we would only have access to these noisy measurements and not to the true function that generated the data. Finding a good estimate of $f(x)$ based on this contaminated data is one of our main objectives.

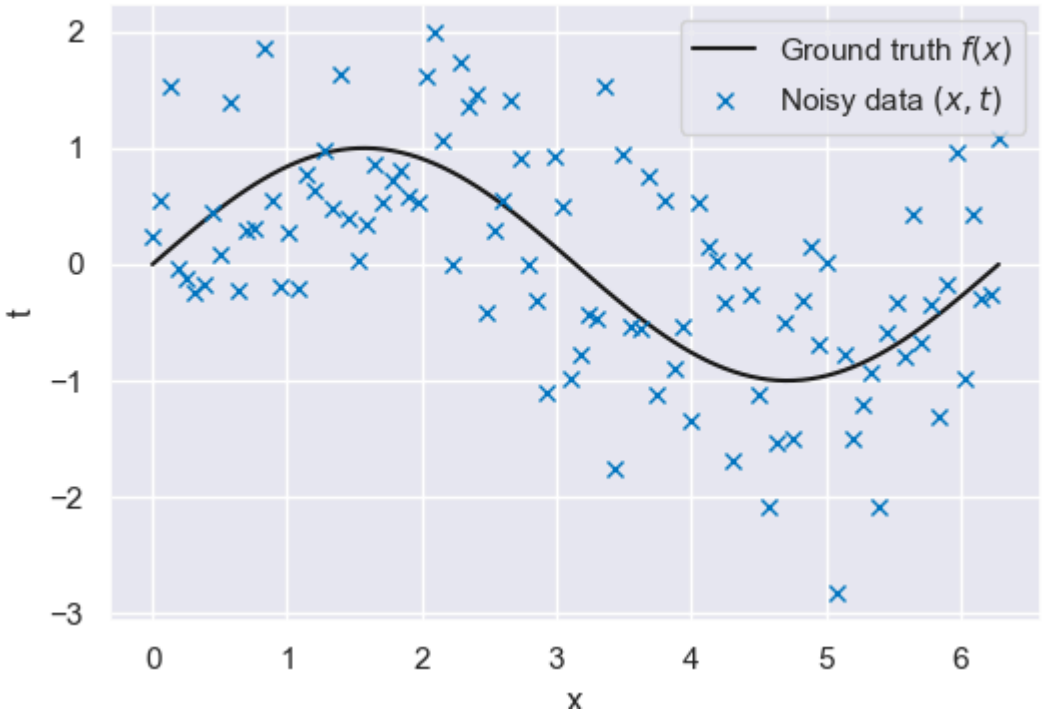



*Fig. 6.24* 100 noisy measurements from sine wave ground truth

# k-nearest neighbors

A (perhaps naive) approach to find $y(x)$ (i.e. the approximation of $f(x)$) would be to simply look at the surrounding data points and take their average to get an estimate of $f(x)$. This approach is called **k-nearest neighbors**, where $k$ refers to the number of surrounding points we are looking at.

Implementing this is not trivial, but thankfully we can leverage existing implementations. The `KNeighborsRegressor` function from the `sklearn.neighbors` library can be used to fit our data and get $y(x)$ to make predictions.

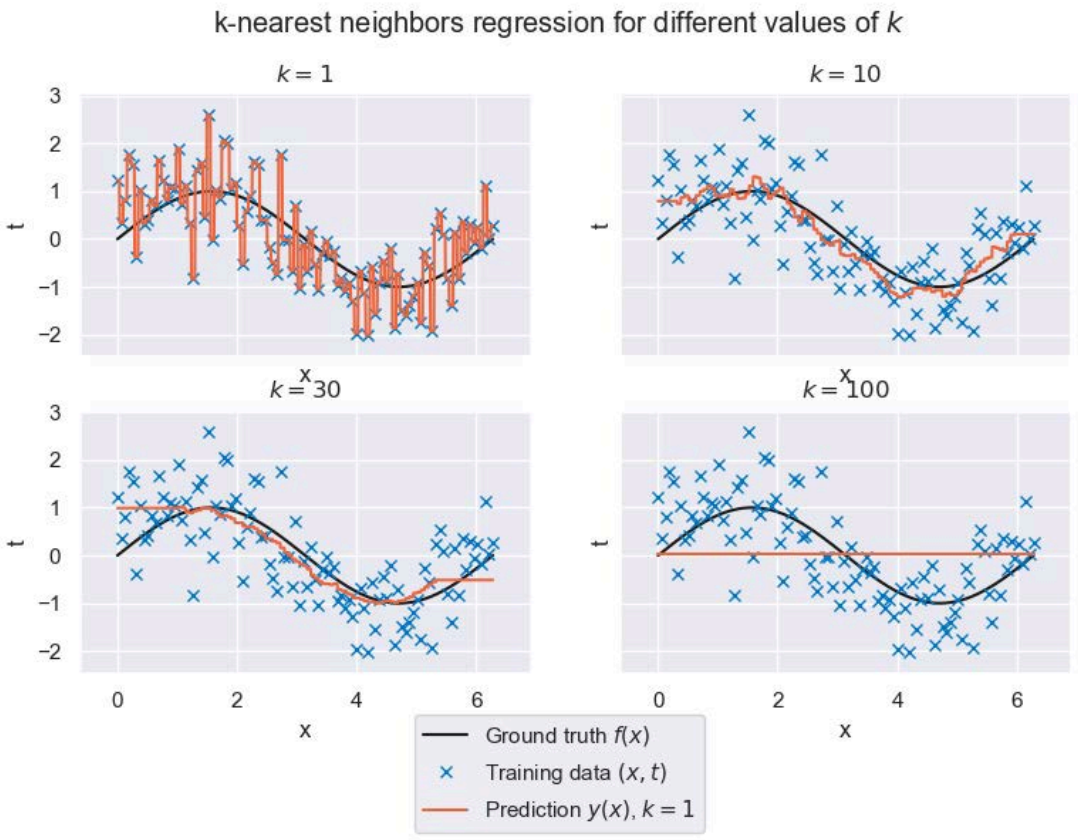



*Fig. 6.25* k-nearest neighbors regression for different values of $k$

Looking at the previous plots, a few questions might pop up:

- For $k = 1$, we see that our prediction matches the observed data exactly in all data points. Is this desirable?
- For $k = 30$, what is going on around $x = 0$ and $x = 2\pi$?
- For $k = 100$, why is our prediction constant with respect to $x$?

# Varying our model parameters

Clearly, some value of $k$ between 1 and 100 would give us the best predictions. In the next interactive plot the following variables can be adjusted:

- $N$, the size of the training data set
- $\varepsilon$, the level of noise associated with the data
- $k$, the number of neighbors over which the average is taken
- The seed can be updated to generate new random data sets
- The oscillation frequency of the underlying ground truth model, which controls how nonlinear the approximation should be

- A probing location $x_0$, allowing you to see which neighbors are used to get the average response $y(x_0)$ of the kNN estimator

# Playing around with the plots

By visual inspection, use the slider of $k$ to find its optimal value. The following questions might be interesting to ask yourself:

- If the training size $N$ increases/decreases, how does this affect my optimal value of $k$?

**Answer**

In general, the more data we have, the more complex we can allow our model to be. But kNN estimators can be somewhat exceptional in this regard, here we tend to see higher values of $k$ produce a visually better result for larger datasets.

- If the frequency increases/decreases, how does this affect my optimal value of $k$?

**Answer**

The frequency is related to how nonlinear our ground truth is. Excessive averaging (higher $k$) might be detrimental in this scenario. And of course, the more complex our process is, the more data we will need

- If my measurements are less/more noisy, how does this affect my optimal value of $k$?

**Answer**

Fitting noise is one of the main causes of overfitting. The less noise we have the lesser the risk of overfitting. Keep in mind however that even models trained on zero-noise datasets can overfit (i.e. give senseless predictions between data points). Reducing the noise here makes the resulting model less sensitive to our choice of $k$.

- If I generate new data by changing the seed, how is my prediction affected for small values of $k$? What about large values of $k$?

**Answer**

This will become quite clear after watching all the videos for this week. Models with low $k$ have high variance and low bias. High-variance models are highly sensitive to specific realizations of small datasets (this is why we say 'high variance'). If these terms sound mysterious to you just keep going for now and come back after you have been through the whole content.

- So far, all observations were distributed uniformly over $x$. How would our predictions change if our observed data was more clustered?

**Answer**

kNN regards on neighborhood data. You can have an idea of what would happen by looking at the edges of our sampling space (around $x = 0$ and $x = 6$). Since there is nothing on the left (or right) of these points, kNN has to rely on a larger range of $x$ values in order to be able to find the same number of neighbors. This degrades prediction performance quite a bit.

- If I do not know the truth, how do I figure out what my value of $k$ should be?

**Answer**

In the next chapters we will go through concepts of model selection, including the definition of a validation dataset and how we can use it to make informed choices about model complexity.

# Final remarks

So far, we have looked at our k-nearest neighbors regressor mostly qualitatively. However, it is possible to apply a more quantitative framework to our model and find the optimal value for $k$ in a more structured way. The following video and accompanying chapter will discuss how this is done. See you there!

**Attribution**

This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

# Decision Theory

## Contents


- Introduction
- Loss function
- Theory
- Model selection
- Bias-variance tradeoff
- Playing around with the plot
- Final remarks


This page contains interactive python element: click 🚀 –> Live Code in the top right corner to activate it.

## Introduction

In the previous chapter, we built a k-nearest neighbors model and observed the influence of choosing various values for k. We found that choosing k depends, among other things, on

the specific dataset used and the noise in the data. By tweaking k, we could get a model which **qualitatively** fits our data well.

In this chapter, we will try to **quantify** how well a specific model performs for any dataset, which can help us choose the best one.

# Loss function

To quantify the "closeness" between predictions of some model $y(\mathbf{x})$ and the observed data $t$, we introduce the squared loss function:

$$L(t, y(\mathbf{x})) = (y(\mathbf{x}) - t)^2,$$

where $t$ are the observed values corresponding to $x$. Squaring the difference gives a number of nice properties:

- The loss is always positive, regardless of whether we underestimate or overestimate a prediction.
- It is differentiable at $y(\mathbf{x}) = t$, which is not true for all loss functions (e.g. the absolute loss)

Below we plot this loss for some data points. Notice that in this case we are comparing the observations $t$ to the ground truth $f(\mathbf{x})$ (as opposed to some fitted model). The error therefore comes solely from the noise in our observations.

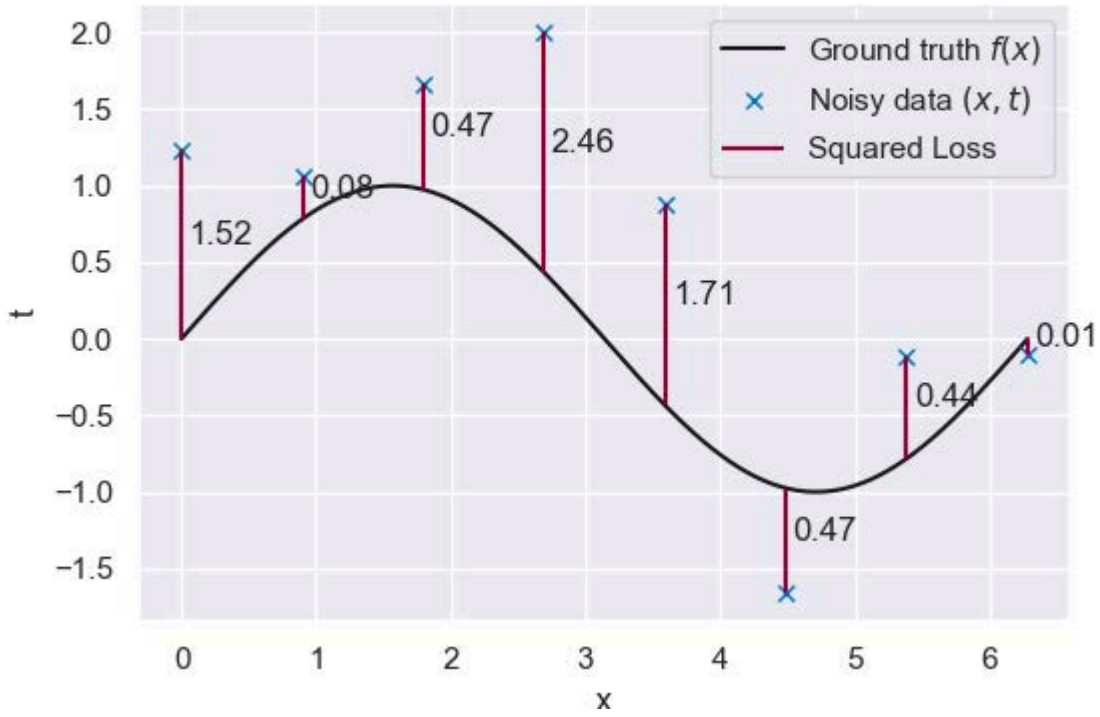



*Fig. 6.26* Ground truth vs noisy data

# Theory

To obtain our model, we want a single value that tells us how well it explains all the data. Therefore it is natural to compute the expected loss:

$$\mathbb{E}[L] = \int \int (y(\mathbf{x}) - t)^2 p(\mathbf{x}, t) d\mathbf{x} dt$$

where $(y(\mathbf{x}) - t)^2$ is the error term we have seen before, and we multiply it with $p(\mathbf{x}, t)$, the probability of this particular point occuring. Integrating this over the complete probability density $p(\mathbf{x}, t)$ results in a single scalar that summarizes our expected prediction error.

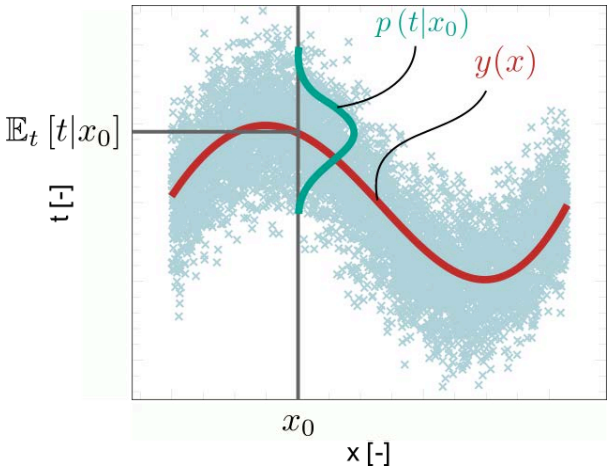
Naturally, our goal is to choose a model $y(\mathbf{x})$ so as to minimize $\mathbb{E}[L]$. We can do this using calculus of variations:

$$\frac{\delta \mathbb{E}[L]}{\delta y(\mathbf{x})} = 2 \int (y(\mathbf{x}) - t) p(\mathbf{x}, t) dt = 0,$$

where we keep the detailed steps of the derivation out of the scope of this discussion. Solving for $y(\mathbf{x})$ we get:

$$\begin{aligned}\int y(\mathbf{x})p(\mathbf{x},t)dt &= \int tp(\mathbf{x},t)dt \\ y(\mathbf{x})p(\mathbf{x}) &= \int tp(\mathbf{x},t)dt \\ y(\mathbf{x}) &= \frac{\int tp(\mathbf{x},t)dt}{p(\mathbf{x})} = \int tp(t|\mathbf{x})dt = \mathbb{E}_t[t|\mathbf{x}]\end{aligned},$$

where the Sum Rule of probability $p(\mathbf{x}) = \int p(\mathbf{x},t)dt$ has been employed and we factorize the joint probability distribution as $p(\mathbf{x},t) = p(\mathbf{x})p(t|\mathbf{x})$ (Product Rule). From the result above, the model that minimizes the squared loss is therefore given by the mean of the conditional distribution $p(t|\mathbf{x})$. This important result can also be seen graphically for a regression model with a single input:




In practice we generally do not know $p(\mathbf{x},t)$ or $p(t|\mathbf{x})$ exactly. Instead, we can estimate the loss for any model by taking the average of the squared losses for each point in our limited dataset. We can then tweak $y(\mathbf{x})$ to minimize this loss.

$$\int\int (y(\mathbf{x})-t)^2 p(\mathbf{x},t)d\mathbf{x}dt \approx \frac{1}{N}\sum_{i}^{N}(y(\mathbf{x}_i)-t_i)^2$$

This is known as the Mean Squared Error (MSE) function. We now revisit our k-nearest neighbors model with the value of the training loss shown in the plot title.

# Model selection

For $k = 1$ our loss function goes to exactly zero, yet the resulting model will not yield good predictions for unseen data. Our model has fitted the noise in the dataset, and therefore does not generalize well. We say that we have **overfitted** our model to the training data.

Clearly, evaluating the squared loss solely on our training points does not suffice if our goal is to make an informed choice for $k$.

To solve this problem a validation set is introduced. This set consists of different observations from the process which are not included during training. We also define a test set with samples that can be used **at the very end** of our model building process in order to assess in an unbiased way if our choice of $k$ has been well informed. We therefore have three separate loss values:

| | |
|---|---|
| **Training Loss:** | The value of the loss function based on all observations used to train a model |
| **Validation Loss:** | The value of the loss function based on observations not used during training, used for finding hyperparameters such as $k$ |
| **Test Loss:** | The value of the loss function based on observations not used during either training or validation, used for evaluating the fit of the final model (for example to compare to alternative models) |

We repeat the experiment above with an additional validation set for which we compute the validation loss. This validation error represents the error we obtain when making new predictions that were hidden from the model during training and is therefore a much better indication for assessing our model compared to the training error. Minimizing the error on this validation set allows us to find a good value for k.

**Try it out above!** How does this compare with your findings from the previous chapter? Note that we now leave 20% of data points out for validation, and that the trained function does not fit them exactly. For a given dataset size, move the $k$ slider until the validation loss is as low as possible.

Having selected a value for $k$, we could then compute the loss on our test dataset as one final check of model validity. We leave that step to you when working on our real-world problems later on.

# Bias-variance tradeoff

As we determined earlier, the optimal prediction (in the sense that it minimizes the squared loss function), which we will denote as $h(\mathbf{x})$, is given by:

$$h(\mathbf{x}) = \mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x}) dt$$

Starting from the expression for the loss function you know, we add and subtract the equation above with no loss of generality. This allows us to split the loss expression in two important parts:

$$\begin{aligned}
\mathbb{E}[L] &= \int\int (y(\mathbf{x}) - t)^2 p(\mathbf{x}, t) d\mathbf{x} dt \\
&= \int\int (y(\mathbf{x}) - h(\mathbf{x}) + h(\mathbf{x}) - t)^2 p(\mathbf{x}, t) d\mathbf{x} dt \\
&= \int\int (y(\mathbf{x}) - h(\mathbf{x}))^2 p(\mathbf{x}, t) d\mathbf{x} dt + \int\int (h(\mathbf{x}) - t)^2 p(\mathbf{x}, t) d\mathbf{x} dt + 2 \int\int (y(
\end{aligned}$$

Now note that the first term does not depend on $t$ anymore (remember $h(x)$ is already an integral over $t$). This means we can simplify it to an expectation only over $\mathbf{x}$. Furthermore, the third term vanishes once we integrate over $t$. We therefore end up with the following very important decomposition:

$$\mathbb{E}[L] = \underbrace{\int (y(\mathbf{x}) - h(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x}}_{\text{reducible loss}} + \underbrace{\int\int (h(\mathbf{x}) - t)^2 p(\mathbf{x}, t) d\mathbf{x} dt}_{\text{irreducible noise}}$$

The irreducible noise term represents the variance that will always be present in our model because we make noisy observations. Even if we had the perfect model $h(\mathbf{x})$, we would still have an expected loss greater than zero due to the noise inherent to our data. For the kNN model above, this is a bit hidden because we have **exactly one observation** for each value of $x$. This is the reason why we can still obtain zero loss with our kNN estimator, but in practice we would ideally have several observations for the same value of our inputs.

Regardless, focusing now on the first term, we might note that actually, our model $y(\mathbf{x})$ not only depends on $\mathbf{x}$ but also on which dataset $\mathcal{D}$ we used for training. So, the first integral should be replaced as follows:

$$\int (y(\mathbf{x}) - h(\mathbf{x}))^2 p(\mathbf{x}) d\mathbf{x} \quad \Longrightarrow \quad \int \mathbb{E}_{\mathcal{D}} \left[ (y(\mathbf{x}, \mathcal{D}) - h(\mathbf{x}))^2 \right] p(\mathbf{x}) d\mathbf{x}$$

Now, we do some rearrangement in the inner part of this expression:

$$
\begin{aligned}
(y(\mathbf{x}, \mathcal{D}) - h(\mathbf{x}))^2 &= (y(\mathbf{x}, \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})] + \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})] - h(\mathbf{x}))^2 \\
&= (y(\mathbf{x}, \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})])^2 + (\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})] - h(\mathbf{x}))^2 + 2\,(y(\mathbf{x}, \mathcal{I}
\end{aligned}
$$

and then taking the expectation over all possible datasets we get:

$$
\begin{aligned}
\mathbb{E}_{\mathcal{D}}\left[(y(\mathbf{x}, \mathcal{D}) - h(\mathbf{x}))^2\right] &= \mathbb{E}_{\mathcal{D}}\left[(y(\mathbf{x}, \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})])^2\right] + \mathbb{E}_{\mathcal{D}}[(\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})] - h(\mathbf{x}))^2] \\
&= \underbrace{\mathbb{E}_{\mathcal{D}}\left[(y(\mathbf{x}, \mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})])^2\right]}_{\text{variance}} + \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})] - h(\mathbf{x})\}^2}_{\text{bias}}
\end{aligned}
$$

Overall, the expected loss can be decomposed as follows:

$$
\text{expected loss} = \text{bias}^2 + \text{variance} + \text{noise}
$$

**MUDE Exam Information**

You will not be asked to reproduce this derivation on the exam, but you should be familiar with the meaning of this final result and with the interpretation of each of these important terms.

These terms can be understood as follows:

- **Bias**: Represents the mismatch between the expectaction of our model $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})]$ and the optimal prediction $h(\mathbf{x})$. An **unbiased** model will on average give us the optimal prediction, whereas a **biased** model will on average deviate from the optimal prediction.
- **Variance**: Represents the degree to which a given model $y(\mathbf{x}, \mathcal{D})$ (trained with one specific realization of $\mathcal{D}$) deviates from the average model $\mathbb{E}_{\mathcal{D}}[y(\mathbf{x}, \mathcal{D})]$. If the variance is large, it means that a new dataset would produce a very different model.
- **Noise**: Represents the inherent variation in our dataset. This term cannot be reduced because it stems from our data and not from our choice of model.

Generally speaking, there is a certain balance to strike between bias and variance. A more flexible model tends to target the optimal prediction $h(\mathbf{x})$ on average but can be significantly influenced by the dataset to which it is fitted. In other words, it has a lower bias but a higher variance. On the other hand, a less flexible model tends to be less affected by the dataset to

which it is fitted, but this causes it to deviate from $h(\mathbf{x})$. In other words, it has a lower variance but a higher bias. This dilemma is referred to as the **bias-variance tradeoff**.

Looking back at the plot above, try to answer the following questions:

- When k is low, do we have a high or a low bias? And what about the variance?
- When k is high, do we have a high or a low bias? And what about the variance?

We will now quantify the bias and variance by creating several datasets and averaging over them, following the equations above. Notice that computing the expressions above is only possible here because we know the ground truth in this example, that is, we know what the perfect model $h(\mathbf{x})$ should look like.

# Playing around with the plot

Understanding bias and variance is critical for any method you use, and this will come back in many of the following lectures. Being able to answer some of the following questions can help you understand what is going on:

- Why is the variance small for a large $k$?

**Answer**

With higher values of $k$, we are averaging over a larger neighborhood. Intuitively, you can imagine that if our noise is high, generating multiple datasets of the same size will result in very different datasets, especially if $N$ is small. But over larger and larger neighborhoods, it makes sense that this noise we are adding tends to cancel out between points (remember the expected value of our noise is zero). It is therefore more likely that our resulting model will not change all that much for different datasets. This is the definition of a model with low variance (and therefore high bias!).

- Why is the bias not exactly zero everywhere when $k = 1$?

**Answer**

The bias should be zero at the locations of every one of the $N$ data points and when we take expectations over an infinite number of datasets. Take a look at what happens with the bias between data point locations and for different numbers of datasets.

- Why does the bias increase when choosing a high frequency $freq$?

**Answer**

The complexity of our ground truth increases with frequency. So it makes sense that we would then need a more complex model to capture it properly. This in turn means models with the same $k$ will likely show an increase in bias. Note that this is not the case for $k = 1$.

**TIP**: When playing with the plot, be aware that training the model with $2000$ different datasets might take a while! When moving the sliders set that to a lower number of datasets and only click the '2000 datasets' button when you are done with the sliders.

# Final remarks

So far, we have looked at our k-nearest neighbors regressor, which is relatively easy to understand and play around with. However, this method is not often used in practice, as it scales poorly for large datasets. Furthermore, computing the closest points becomes computationally expensive when considering multiple inputs (high dimensions). In a limited data setting, taking the average of the nearest neighbors will often lead to poor performance. k-nearest neighbors is also sensitive to outliers. In the following chapter, we will apply what we have learned to a different model, namely, linear regression.

**Attribution**

This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

# Linear Basis Function Models and Regularization

## Contents


- Linear Basic Function Models
- Regularization


This page contains interactive python element: click 🚀 –> **Live Code** in the top right corner to activate it.

## Linear Basic Function Models

So far, we have been using a non-parametric model with k-nearest neighbors, meaning we needed access to the whole training dataset for each prediction. We will now focus on parametric models, namely linear models with basis functions. Parametric models are defined by a finite set of parameters calibrated in a training step. All we need for a prediction then are the parameter values. There is no longer a need to carry the whole dataset with us;

the information used to make predictions is encoded in the model parameters. Once again, we will employ the simple sine function to demonstrate the concepts presented in this chapter.

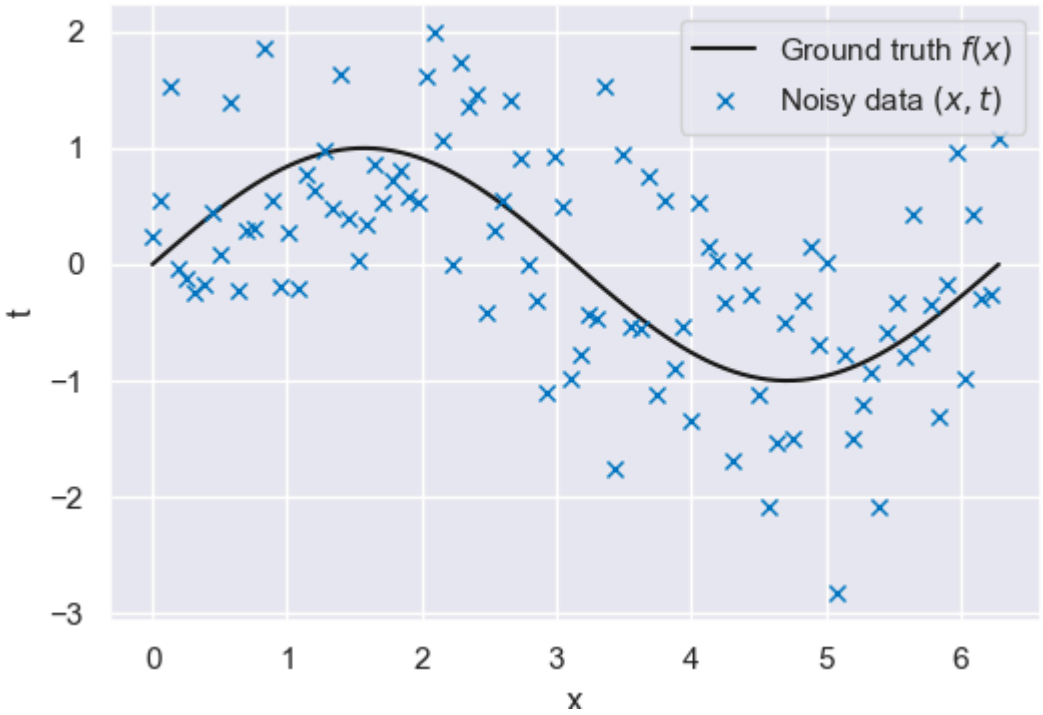



*Fig. 6.27* Ground truth vs noisy data

# Linear model

The key idea behind linear regression models is that they are linear in their parameters $\mathbf{w}$. They might be linear in their inputs as well, although this does not necessarily need to case, as we will see later on in this chapter. The simplest approach is to model our target function $y(x)$ as a linear combination of the coordinates $x$:

$$y(x, \mathbf{w}) = w_0 + w_1 x_1$$

In the one dimensional case, this is equivalent to fitting a straight line through our datapoints. The parameter $w_0$, also referred to as bias (**not to be confused with the model bias from the previous chapter**), determines the intercept, $w_1$ determines the slope. The introduction of a dummy input $x_0 = 1$ allows us to write the model in a more concise way:

$$y(x, \mathbf{w}) = w_0 x_0 + w_1 x_1 = \mathbf{w}^T \mathbf{x}$$

We will use the least squares error function from the previous chapter to fit our model, but will first show how this choice is motivated by a Maximum likelihood approach.

# Maximum Likelihood Estimation

Oftentimes, it is assumed that the target $t$ is given by a deterministic function $y(\mathbf{x}, \mathbf{w})$ with additive Gaussian noise so that

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{with} \quad \epsilon \sim \mathcal{N}(0, \beta^{-1}),$$

with precision $\beta$ (which is defined as $1/\sigma^2$). Note that this assumption is only justified in case of a unimodal conditional distribution for $t$. It can have grave influence on the model accuracy and its validity therefore must be carefully assessed.

As seen in the previous chapter, for the square loss function, the optimal prediction for some value $\mathbf{x}$ is given by the conditional mean of the target variable $t$. In this case, this conditional mean is given by:

$$\mathbb{E}[t|\mathbf{x}] = \int t p(t|\mathbf{x}) \mathrm{d}t = y(\mathbf{x}, \mathbf{w}).$$

Given a new input $\mathbf{x}$, the distribution of $t$ that follows from our model is

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

Consider now a dataset $\mathcal{D}$ consisting of inputs $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and targets $\mathbf{t} = \{t_1, \ldots, t_n\}$. Assuming our datapoints are drawn independently from the same distribution (*i.i.d. assumption*), the likelihood of drawing this dataset from our model is

$$p(\mathcal{D}|\mathbf{w}) = \prod_{n=1}^{N} \mathcal{N}(t_n|y(\mathbf{x}_n, \mathbf{w}), \beta^{-1}),$$

also referred to as the likelihood function. Taking the logarithm and expanding on the classic expression for a multivariate Gaussian distribution gives:

$$\ln p(\mathcal{D}|\mathbf{w}) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^T\mathbf{x}_n, \beta^{-1}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta \underbrace{\frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{w}^T\mathbf{x}_n)^2}_{E_\mathcal{D}}$$

where we can identify our square-error loss function in the last term. Note that the first two terms are constant for a given dataset and have no influence on the parameter setting $\bar{\mathbf{w}}$ that maximizes the likelihood. Those optimal parameters values $\bar{\mathbf{w}}$ can be obtained by setting the gradient of our loss function w.r.t. $\mathbf{w}$ to zero and solving for $\mathbf{w}$.

$$\nabla_\mathbf{w} E_\mathcal{D} = \frac{1}{N}\sum_{n=1}^{N}\big(t_n - \mathbf{w}^T\mathbf{x}_n\big)\mathbf{x}_n \overset{!}{=} 0$$

It is convenient to concatenate all inputs to a *design matrix* $\mathbf{X} = [\mathbf{x}_1^T, \ldots, \mathbf{x}_N^T]^T$. Solving for $\mathbf{w}$ gives

$$\bar{\mathbf{w}} = \big(\mathbf{X}^T\mathbf{X}\big)^{-1}\mathbf{X}^T\mathbf{t}$$

which is the classical expression for a least-squares solution you have by now seen many times during the course.

# Data normalization

Before we move on to fitting the model, we normalize our data. **This step is recommended for most machine learning techniques and is often even necessary.** A non-normalized dataset almost always leads to a numerically more challenging optimization problem. Part of model selection is to ensure that our basis functions show the desired behavior in the relevant parts of the domain. We center and rescale our data, a process referred to as standardization, to operate in the vicinity of the origin only. The standardized dataset $\hat{\mathcal{D}} = (\hat{x}, \hat{t})$ is obtained by subtracting the sample mean $\mu$, and dividing by the sample standard deviation $\sigma$ of the data:

$$\hat{x} = \frac{x - \mu_x}{\sigma_x} \quad \text{and} \quad \hat{t} = \frac{t - \mu_t}{\sigma_t}.$$

Take a look below at the standardized and unstandardized data. Note that the standardization of the target $t$ has a marginal effect, as the sine function is already centered at 0 and almost shows a standard deviation of 1. A 4 by 4 square has been added to indicate the region from $-2\hat{\sigma}$ to $2\hat{\sigma}$. As you can see, all input and output variables fall roughly in this

interval, and this property allows for more stability when applying numerical solvers to the problem.

Note that **there is no strictly correct way to shift and scale input data**. Depending on the distribution of the data, a min-max scaling or a quantile scaling might lead to a better numerical setup. The dataset's structure needs to be assessed carefully to make an informed decision on normalization.

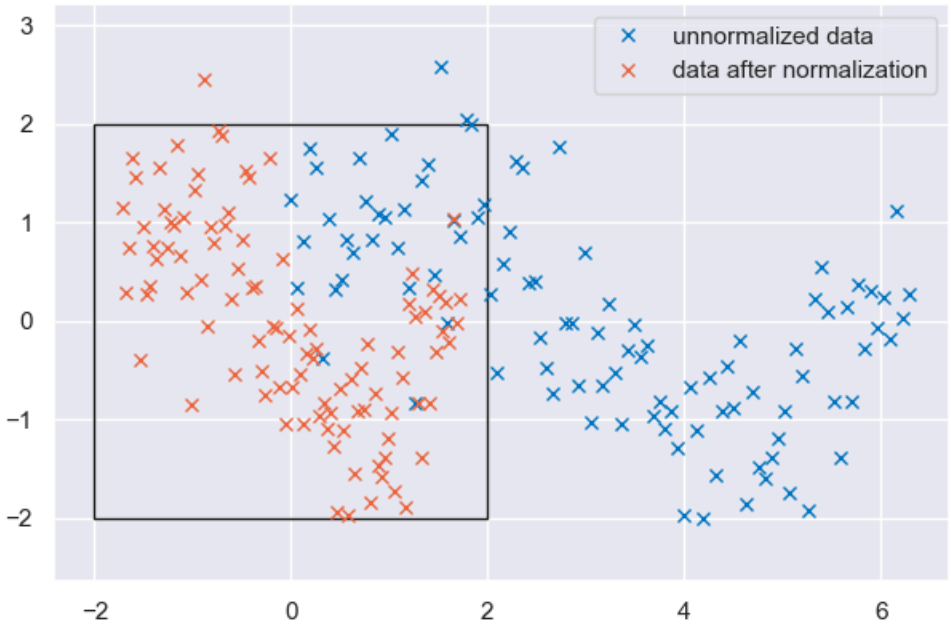



*Fig. 6.28* Unnormalized and normalized data

**Coding in python**

Different packages will offer distinct functionalities to perform the normalization. In the case of `scikit-learn`, the package `sklearn.preprocessing` offers an wide range of normalizers:

- `sklearn.preprocessing.StandardScaler` implements the normalization scheme from our linear models page, based on sample mean and standard deviation;
- `sklearn.preprocessing.MinMaxScaler` uses dataset bounds to scale each feature based on its minimum and maximum value across the complete training dataset. This makes each separate input feature fall within a $[0, 1]$ interval;
- `sklearn.preprocessing.MaxAbsScaler` scales each feature based only on their maximum value across the dataset.
- `sklearn.preprocessing.RobustScaler` implements a specialized normalization scheme that makes the trained models less sensitive to the presence of outliers in the training dataset

Different normalizers are more suitable for different applications. It is usual to experiment with different normalizers when looking at a new problem. Regardless, using normalizers is a straightforward affair:

# Generalized linear models

Let us first train the model above on our nonlinear dataset:

*Fig. 6.29* Linear prediction

It is clear from the plot that a linear model with linear features lacks the flexibility to fit the data well. A bias-variance decomposition analysis for this model would show that it has little variance but shows a strong bias. We now consider nonlinear functions of the input $x$ as features/regressors to increase the flexibility of our linear model. A common approach is to use a set of polynomial basis functions,

$$\phi_j(x) = x^j,$$

but numerous other choices are possible. The full formulation for a model with $M$ polynomial basis functions is thus

$$y(x, \mathbf{w}) = w_0 x^0 + w_1 x^1 + w_2 x^2 + \ldots + w_M x^M,$$

which shows how the model is still linear w.r.t. $\mathbf{w}$, even though it is no longer linear in the input parameters. The design matrix for this more general case reads

$$\mathbf{\Phi}_{ij} = \phi_j(x_i).$$

As was the case for $\mathbf{X}$, we need to ensure that $\mathbf{\Phi}$ has full column rank. This is not always the case the case, for example if we have more basis functions that data points, or if our basis functions are not linearly independent.

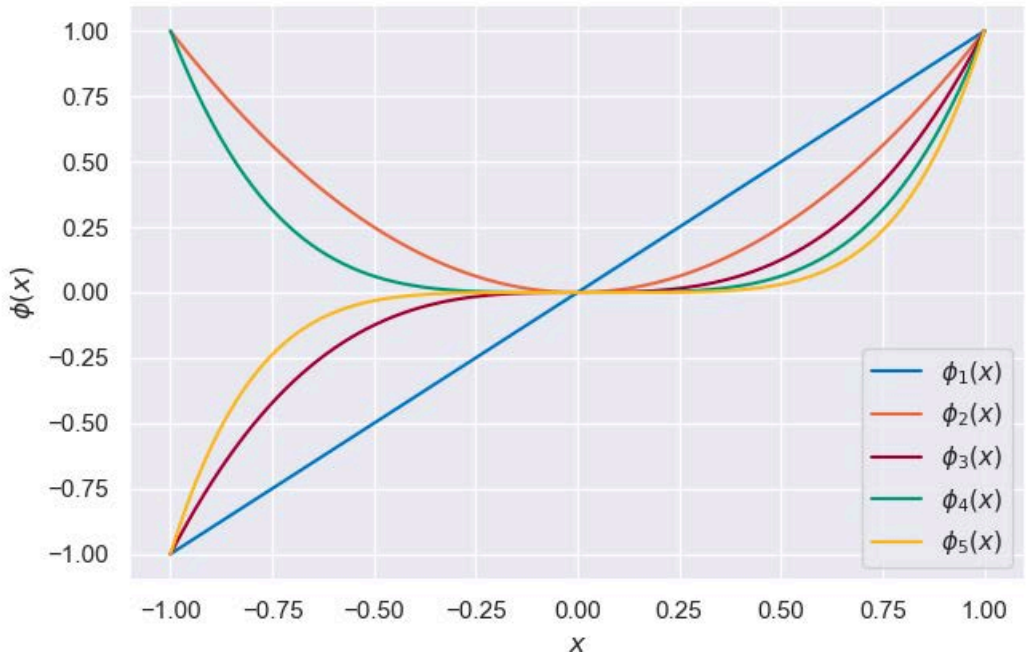



*Fig. 6.30* Polynomial basis functions

We obtain the linear model with nonlinear basis functions by replacing the coordinate vector $x$ with the feature vector $\boldsymbol{\phi}(x)$

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j \phi_j(x) = \mathbf{w}^T \boldsymbol{\phi}(x).$$

The solution procedure remains the same, and we can solve for $\bar{\mathbf{w}}$ directly

$$\bar{\mathbf{w}} = \left(\boldsymbol{\Phi}^T \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^T \mathbf{t}.$$

Let's take a look at the linear model with polynomial regressors.

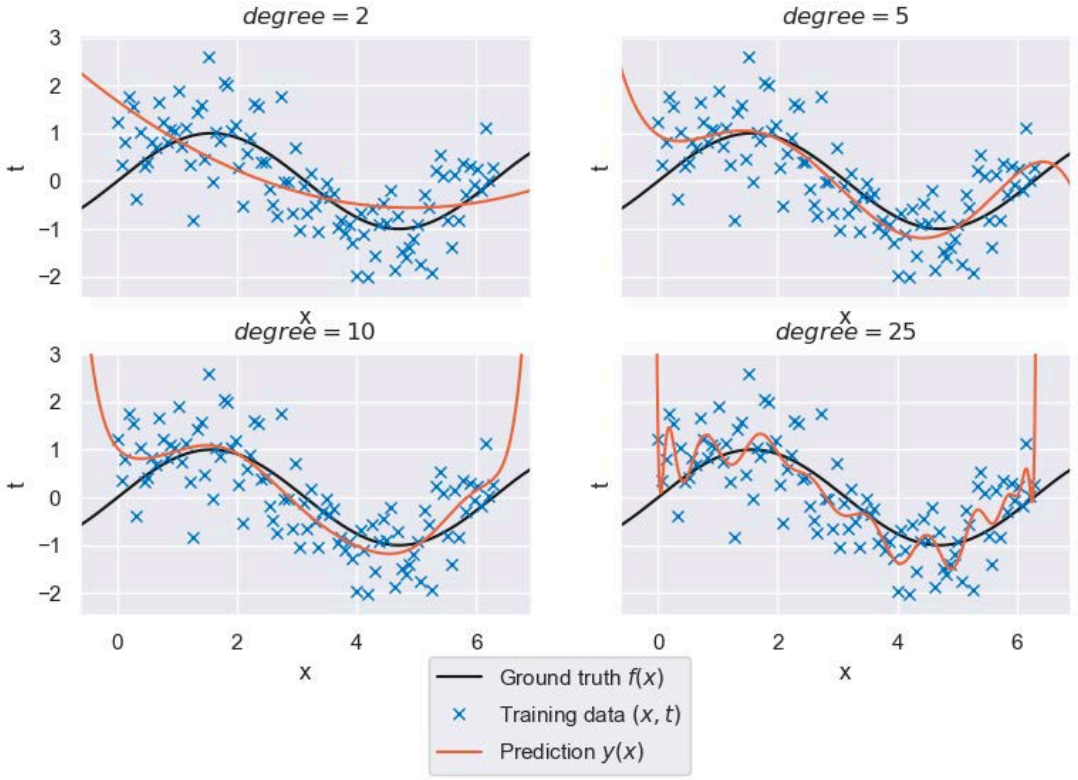



*Fig. 6.31* Generalized linear regression for polynomials of degree $p$

That is looking much better already. However, the quality of the fit varies significantly with the degree of the polynomial basis. There seems to be an ideal model complexity for this specific problem. Try out the interactive tool below to get an idea of the interplay of the following variables:

- $p$, the degree of the polynomial basis
- $N$, the size of the training data set
- $freq$, the frequency of the underlying truth
- $\varepsilon$, the level of noise associated with the data
- The seed can be updated to generate new random data sets
- The truth can be hidden to simulate a situation that is closer to a practical setting

A few questions that might have crossed your mind when playing with the tool:

- With a small amount of data ($N \leq 11$), what happens if we have as many data points as parameters? $(p + 1 = N)$

**Answer**

The model fits all data points exactly, since the least-squares problem is neither over- nor underdetermined. Note how the resulting model looks nothing like the ground truth, and how interpolating between data points usually gives low-quality predictions. These are signs of overfitting. Try it out for different noise levels to see something interesting!

- With a small amount of data ($N \leq 11$), what happens if we have more model parameters than data? ($p + 1 > N$)

**Answer**

The system is now underdetermined and therefore has no unique solution. The least-squares problem is also nearly singular and prone to numerical issues.

- We only have access to data in the interval $[0, 2\pi]$. How well does our model extrapolate beyond the data range?

**Answer**

Unsurprisingly, higher-order models do not extrapolate well at all. In general it is a bad idea to use machine learning models in extrapolation. But try to increase the dataset size and select a more parsimonious model with lower $p$ and see what happens in extrapolation. There is still a lot we can do to carefully craft models that at the very least do not unreasonably deviate from the trends on the dataset when extrapolating.

# Other choices of basis functions

As mentioned previously, the polynomial basis is just one choice among many to define our model. Depending on the problem setting, a different set of basis functions might lead to better results. Another popular choice is the radial basis functions (also called Gaussian basis functions), given by

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2\ell^2}\right\} \quad \text{for} \quad j = 1, \ldots, M$$

where $\phi_j$ is centered around $\mu_j$, $l$ determines the width, and $M$ refers to the number of basis functions.

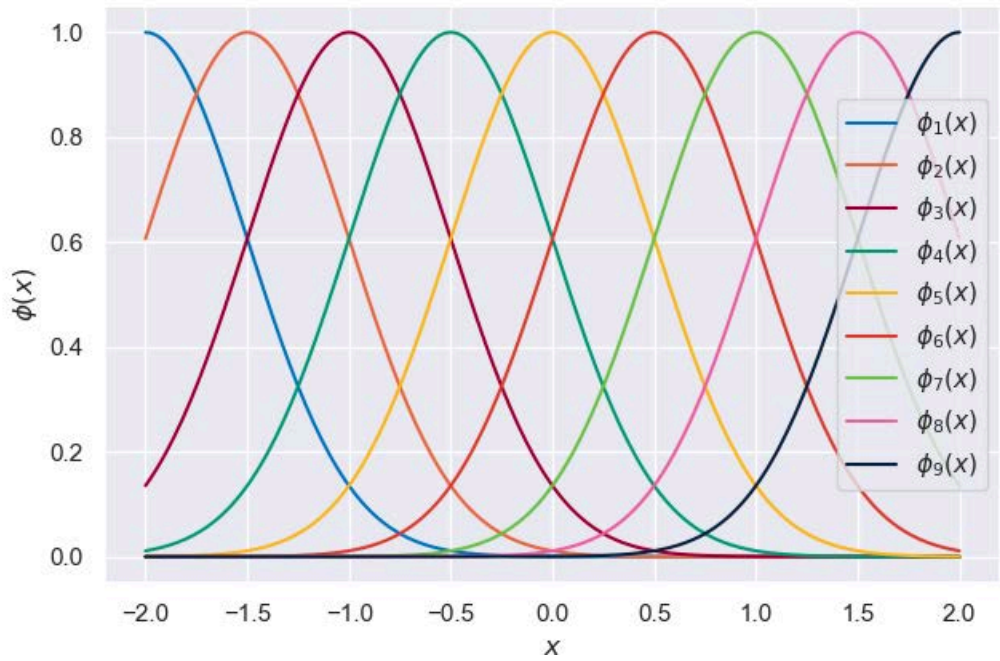



*Fig. 6.32* Radial basis functions

One of the attributes of this model is the locality of its individual functions. This means data in one part of the domain will not impact predictions in other parts of the domain. Periodicity can be achieved with a Fourier basis. Wavelets are popular in signal processing since they are localized in both frequency and space. **It is up to the user to determine which basis function properties are desired for a given problem**, and this is an important part of model selection.

Let's see how well the linear model with radial basis functions performs on the sine wave problem. Keep in mind that the lengthscale parameter corresponds to the lengthscale in the standardized space.

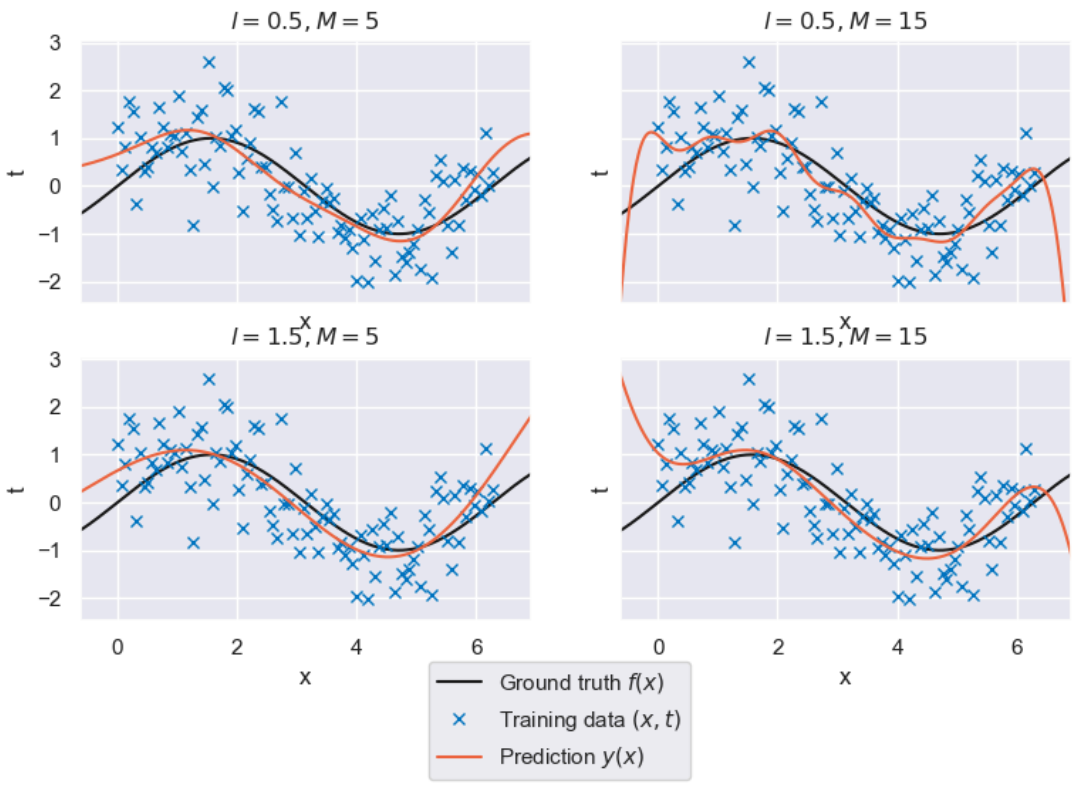



*Fig. 6.33* Generalized linear regression for radial basis functions with varying $M$ and $l$

The figure above shows four different combinations of the hyperparameters (number of basis functions and length scale). The quality of the fit depends strongly on the parameter setting, but a visual inspection indicates our model can replicate the general trend.

# Regularization

We've seen that the least squares approach can lead to severe over-fitting if complex models are trained on small datasets. Our strategy of limiting the number of basis functions a priori to counter overfitting puts us at risk of missing critical trends in the data. Another way to control model flexibility is by introducing a regularization term. This additional term essentially puts a penalty on the weights and prevents them from taking too large values unless supported by the data. The concept of regularized least squares will be demonstrated on the usual sine function.

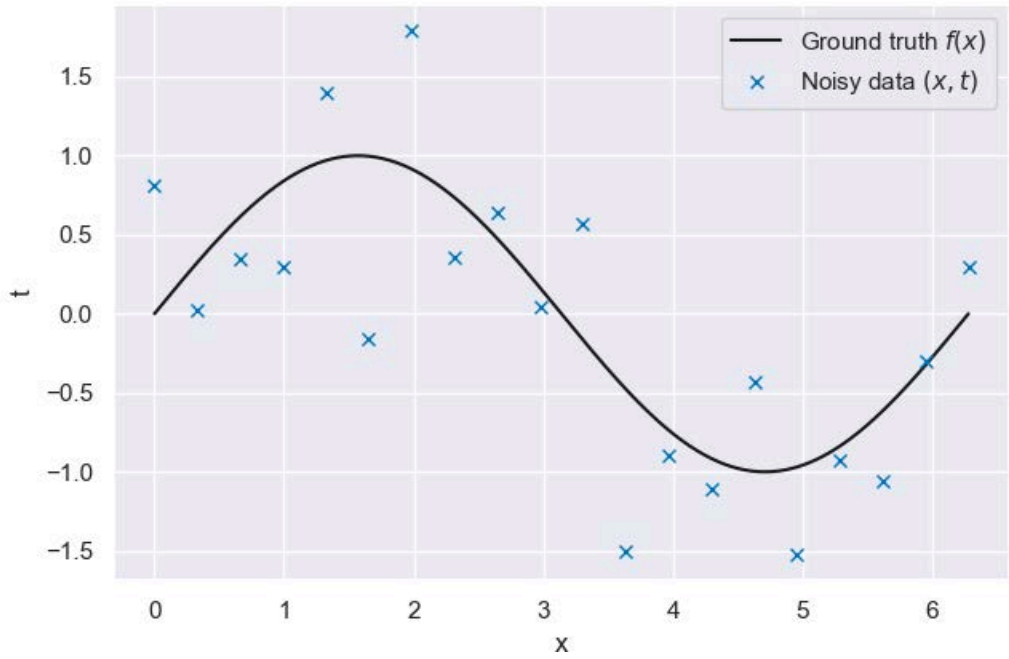



*Fig. 6.34* Noisy data generated from a ground-truth sine wave

We extend the data-dependent error $E_D(\mathbf{w})$ with the regularization term $E_W(\mathbf{w})$:

$$E(\mathbf{w}) = E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

with regularization parameter $\lambda$ that controls the relative importance of two terms comprising the error function. A common choice for the regularizer is the sum-of-squares:

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w}.$$

The resulting model is known as $L_2$ regularization, ridge regression, or weight decay. The total error therefore becomes

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}(x_n)\right)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}.$$

A useful property of the $L_2$ regularization is that the error functions is still a quadratic function of $\mathbf{w}$, allowing for a closed form solution for its minimizer. Setting the gradient of the regularized error function with respect to $\mathbf{w}$ to $0$ gives

$$\bar{\mathbf{w}} = \left(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda\mathbf{I}\right)^{-1}\boldsymbol{\Phi}^T\mathbf{t}.$$

# Determining the regularization parameter

We can use the formulation above in an interactive plot where you can control the dataset size, number of radial bases and the magnitude of the regularization parameter $\lambda$:

Play around with the number of radial basis functions and the regularization parameter. Try to answer the following questions:

- What happens for a low number of RBF, and what happens for a high number of RBF?

**Answer**

As you might expect, more radial basis functions means a more complex model, and therefore more prone to overfitting.

- How does this affect the value of $\lambda$ that yields the (visually) best fit?

**Answer**

Models with high variance will generally require more regularization to reach the same level of bias. So increasing the number of RBFs should require a higher value of $\lambda$ in order to reach the same level of model complexity.

- How does a larger dataset affect the optimal regularization parameter $\bar{\lambda}$?

**Answer**

As we increase $N$ and therefore let the model see more data, it becomes more tolerant to a higher range of values of $\lambda$. That is, even very low values of $\lambda$ will not lead to overfitting, as the data is now there to naturally limit model complexity. Again, overfitting is a problem which is much worse when we do not have a lot of data.

The pressing question is, of course: **how do we determine $\lambda$?**

The problem of controlling model complexity has been shifted from choosing a suitable set of basis functions to determining the regularization parameter $\lambda$. **Optimizing for both $\mathbf{w}$ and $\lambda$ over the training dataset will always favor flexibility, leading to the trivial solution $\lambda = 0$** and, therefore, to the unregularized least squares problem. Instead, similar to what we did before, the data should be partitioned into a training set and a validation set. The training set is used to determine the parameters $\mathbf{w}$, and the validation set to optimize the model complexity through $\lambda$.

For the visualization below we use a pragmatic approach and sample another dense validation set from our ground truth instead of splitting the original training dataset. We then loop over several values of $\lambda$ until we find the one corresponding to the minimum validation error. This is then our selected model. This leads to a best fit for $\lambda = 0.048301$ with MSE $= 0.267945$

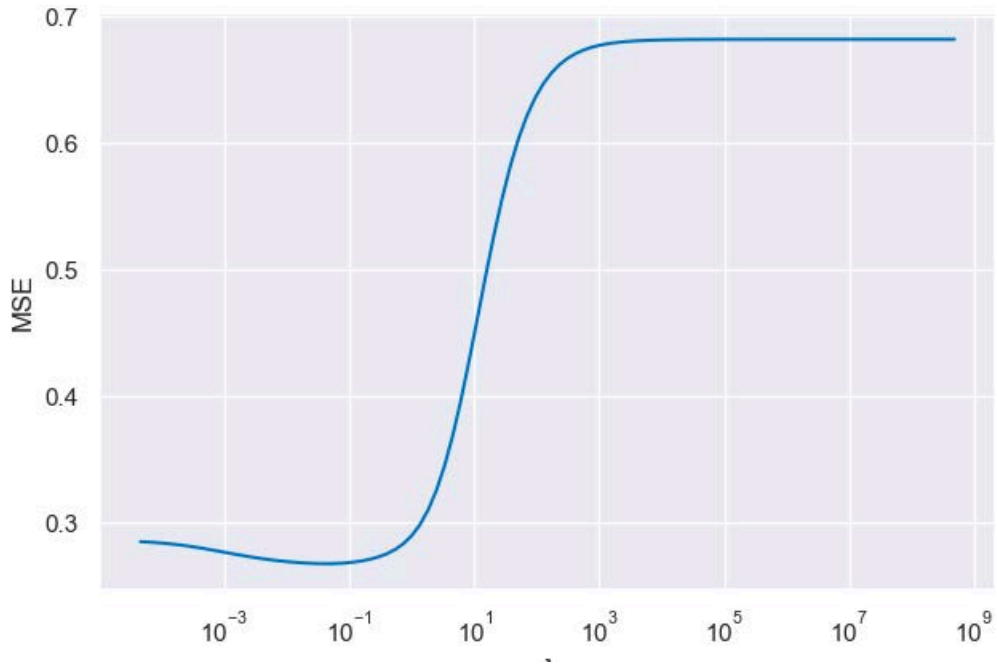



*Fig. 6.35* Minimum validation error for multiple values of $\lambda$

We then take a look at how the optimally regularized model looks like:

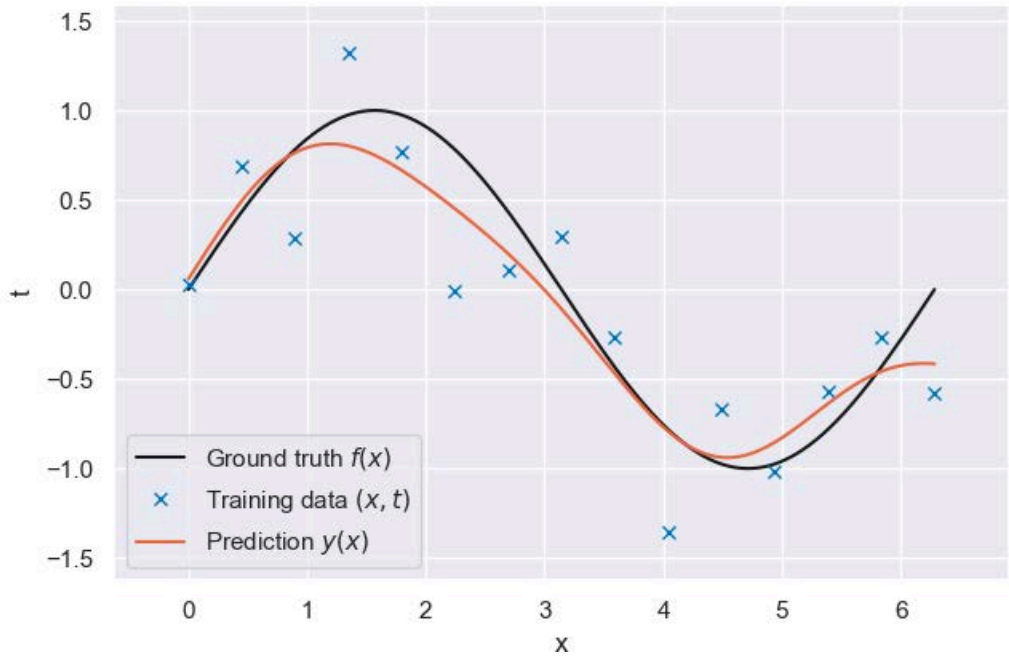



*Fig. 6.36* Optimally regularized model

That looks much better than the solution of the unregularized problem. Luckily, this problem has a unique solution $\bar{\lambda}$. Can you explain why the error first decreases but keeps growing again at some point? Do you always expect this non-monotonic characteristic when looking at the **training** error as a function of the regularization parameter?

> **Attribution**
>
> This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

# Stochastic Gradient Descent

## Contents


- Introduction


This page contains interactive python element: click 🚀 –> Live Code in the top right corner to activate it.

## Introduction

For now, we used the same dataset at once. In some situations, it might be beneficial or necessary to look at only a part of the dataset, e.g., when

- $N$ is too large and computing $(\mathbf{\Phi}^T\mathbf{\Phi})^{-1}$ becomes prohibitively expensive
- the model is nonlinear in $\mathbf{w}$, and $\mathbf{w}_{\text{ML}}$ does not have a closed-form solution
- the dataset is arriving sequentially (e.g., in real-time from a sensor)

Instead of solving for $\mathbf{w}$ directly, we could employ an iterative optimization strategy. Let's first take a look at the data part of the error function, and its gradient with respect to $\mathbf{w}$:

$$E_D = \frac{1}{2N}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}_n\right)^2 \quad \text{with gradient} \quad \nabla E_D = -\frac{1}{N}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}_n\right)\boldsymbol{\phi}_n.$$

The standard formulation does not include the division by the dataset size, the stepsize is purely regulated through the learning rate. However, normalizing the gradient with $N$ makes the influence of the learning rate more consistent when considering different datasets. With a standard gradient descent algorithm, the update rule for the weights is given by

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_D$$

with a fixed *learning rate* $\eta$. The costs for the gradient computations are independent of the dataset size $N$, when only considering subset $\mathcal{B}$ of our dataset with $N_\mathcal{B}$ data points. If we pick a random subset $\mathcal{B}$ for each iteration of the optimization scheme, we have derived the *stochastic gradient descent* algorithm. Together with its numerous variants, this algorithm forms the backbone of many machine learning techniques. Most deep learning libraries, such as `TensorFlow` or `PyTorch` offer implementations of these algorithms.

We looked at the unregularized model to introduce SGD, but the extension to the regularized model is straightforward. Remember, in this case, the objective function is given by

$$E(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}_n\right)^2 + \frac{\lambda}{2N}\mathbf{w}^T\mathbf{w},$$

and its gradient with respect to $\mathbf{w}$ reads

$$\nabla E = \frac{1}{N}\left(-\sum_{n=1}^{N}\left(t_n - \mathbf{w}^T\boldsymbol{\phi}_n\right)\boldsymbol{\phi}_n + \lambda\,\mathbf{w}\right).$$

When looking at the expresssion in the outer bracket, it becomes clear there are **two competing terms**: the first term pulls the weights towards the data, and the second term pulls them towards zero. Looking at the gradient, you can also see why ridge regression often is referred to as weight decay. The larger the weights become, the stronger the regularization term pulls them towards zero. If the data would not support a value for a certain weight, its presence in the gradient will lead to the weight decaying at a rate proportional to its magnitude.

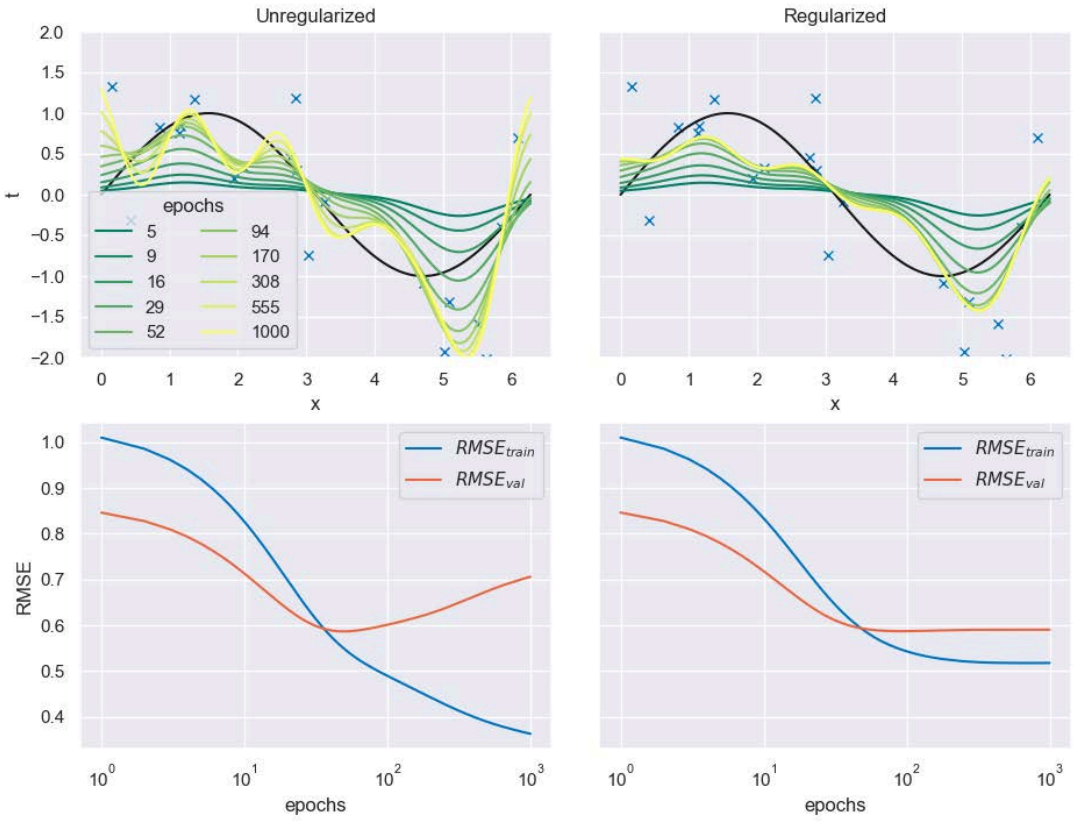


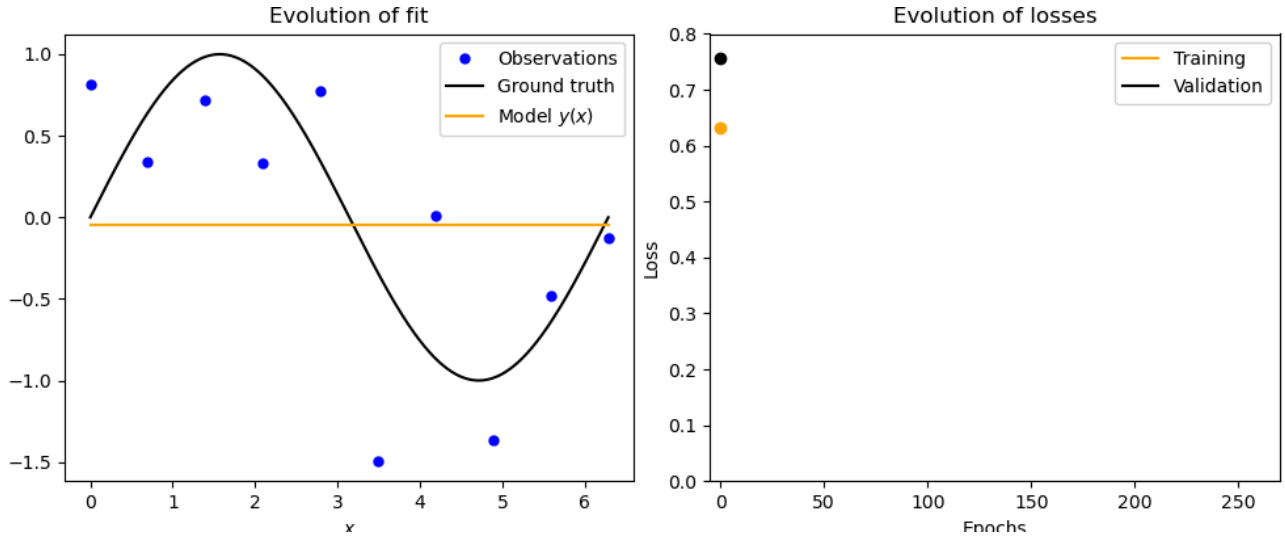

*Fig. 6.37* Training process of two models with different levels of regularization, predictions coming from model snapshots at different epochs.

You can see in the top figure that our predictions seem to converge towards a particular shape. The remaining discrepancy between our final model and the true function $f$ is due to our general model bias, and the particular dataset we drew.

You can see that our validation error increases at some point, indicating that overfitting might occur. Again, note how the training error cannot feel that, and just decreases monotonically. SGD with minibatches already has a slight regularizing effect. Other remedies include the L2-regularization technique discussed discusses previously, early stopping, or collecting more data.

Finally, it should be noted that the step size of the SGD must be chosen carefully. Try out for yourself what happens when you choose very small or large stepsizes by adapting the learning rate. Even though this optimization problem is well-defined and has a global minimum, SGD is not guaranteed to converge to it. Luckily, the most popular variants such as AdaGrad, RMSProp, and Adam feature some form of adaptive stepsize control, improving convergence rate and robustness. One usually starts with a larger stepsize to approach the minimum quickly. After that, the stepsize is reduced continuously to reliably uncover the exact location of the extremum.

In the following animation you can see another example of extreme overfitting being observed during SGD iterations:




Observing when the validation loss starts to increase is a useful sign of the onset of overfitting. Stopping the training process when this is detected is the core of so-called **early stopping** strategies implemented in popular machine learning packages.

> **Attribution**
>
> This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

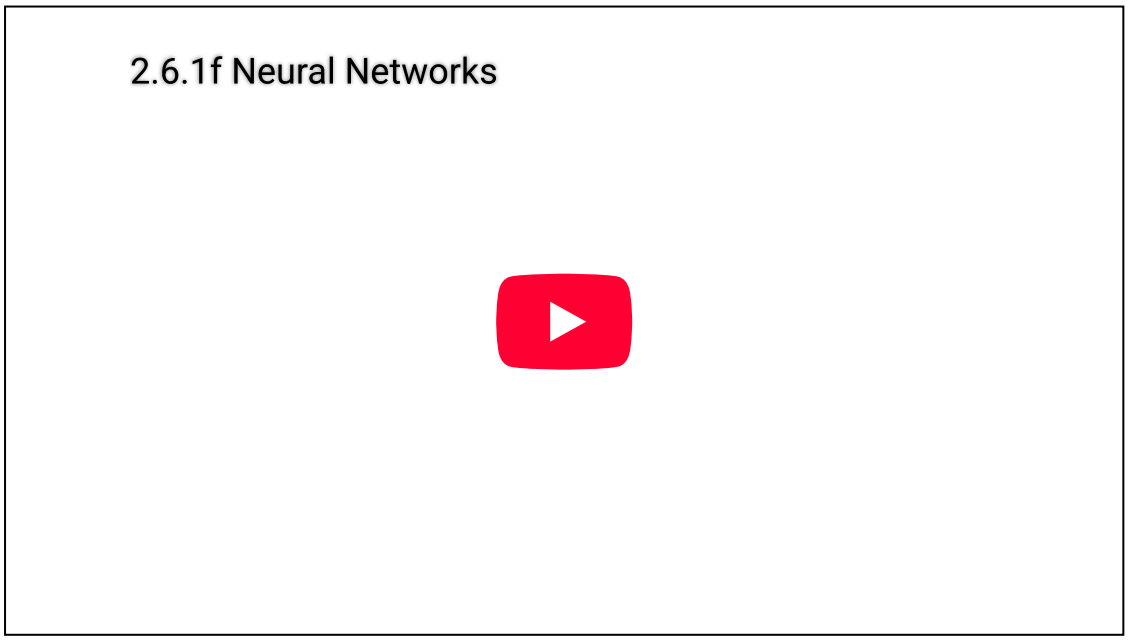
# Feedforward Neural Networks

## Contents


- Introduction
- Neural network architecture
- Model flexibility
- Early stopping
- Manual model selection
- Wrap-up


This page contains interactive python element: click 🚀 –> Live Code in the top right corner to activate it.

## Introduction




Recall that in the previous chapters we have applied linear models with basis functions

$$y(x, \mathbf{w}) = \sum_{j=0}^{M} w_j \phi_j(x) = \mathbf{w}^T \boldsymbol{\phi}(x).$$

Here $\mathbf{w}$ are the flexible parameters, and $\boldsymbol{\phi}$ the basis functions.

Because a linear model is linear in its parameters $\mathbf{w}$, we could solve for $\bar{\mathbf{w}}$ directly

$$\bar{\mathbf{w}} = \left(\mathbf{\Phi}^T \mathbf{\Phi}\right)^{-1} \mathbf{\Phi}^T \mathbf{t},$$

where $\mathbf{\Phi}$ is the collection of basis functions evaluated in all data points. The basis functions need to be chosen *a priori* for this approach. When the phenomenon to be modeled is complex, relying on pre-defined basis functions might not give sufficient accuracy. We can overcome this issue with a more flexible model. Aside from the pragmatic strategy of increasing the number of basis functions, we can also achieve more flexibility by replacing the basis functions with parametric functions. In this chapter we will dive into one variant of this concept, namely **neural networks**. The underlying problem will stay the same: we are trying to learn a process based on a limited number of noisy observations $\mathcal{D} = \{\mathbf{X}, \mathbf{t}\}$. Following decision theory, we need to minimize the mean squared error loss function

$$E_D = \frac{1}{N} \sum_{n=1}^{N} \left(t_n - y(x_n, \mathbf{w})\right)^2$$

where $y(x, \mathbf{w})$ now comes from a neural network.

# Neural network architecture

A neural network consists of neurons connected by weights, with information flowing from input neurons towards output neurons. In **supervised learning**, the states of the input and output neurons are known during training. There are additional neurons in layers in between the inputs and outputs, forming a so-called hidden layer. Neurons are separated into layers, where all neurons of one layer depend on (at least) the neurons of the previous layer.

The state of a neuron is determined by a linear combination of states $z$ from the previous layer with their connecting weights $w$

$$a_j^{(l)} = \sum_i^D w_{ji}^{(l)} z_i^{(l-1)} + w_{j0}^{(l)}$$

where $w_{j0}^{(l)}$ are so-called biases, allowing the model to have an offset. Make sure not to confuse this quantity with the model bias from the *bias-variance* tradeoff discussion. This linear combination of states is followed by a nonlinear transformation with an activation function $h(\cdot)$:

$$z_j^{(l)} = h(a_j^{(l)}).$$

In the plot below, we can see the identity (or linear), sigmoid, hyperbolic tangent (tanh), and rectified linear unit (relu) activation functions applied on an arbitrary state $z$ in the $[-4, 4]$ range.

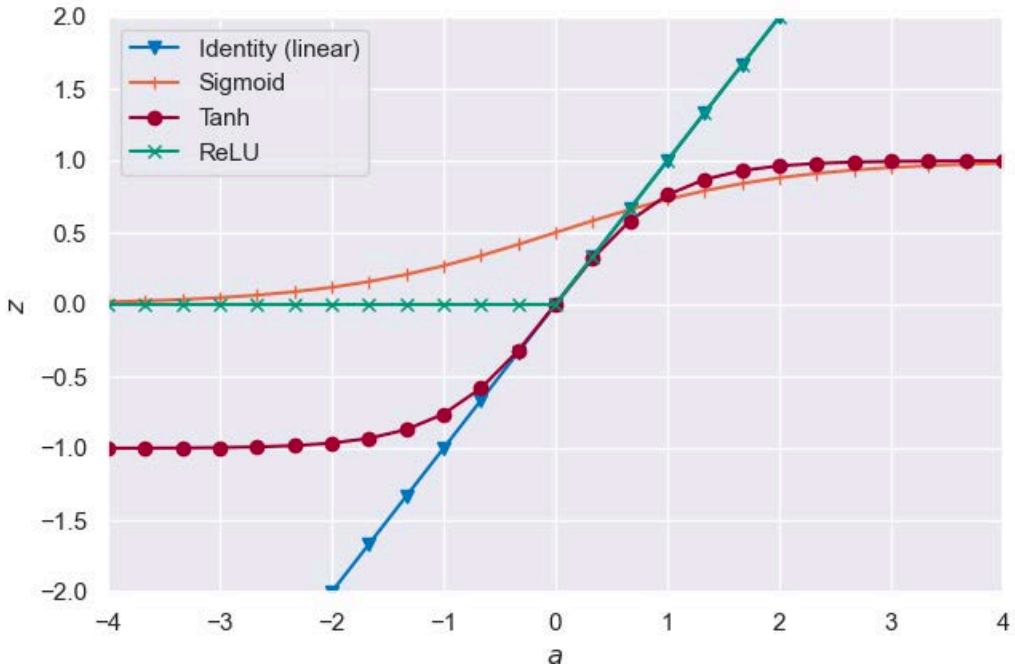



*Fig. 6.38* Activation functions

The number of layers in a neural network commonly refers to the number of hidden layers. Following the aforementioned setup of compounding linear transformations with nonlinear activations, the output of a two-layer neural network can be written as:

$$y(x, \mathbf{w}) = h^{(out)} \left( \sum_{k=1}^{K} w_k^{(3)} h^{(2)} \left( \sum_{j=1}^{J} w_{kj}^{(2)} h^{(1)} \left( \sum_{i=1}^{I} w_{ji}^{(2)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) + w_0^{(3)} \right)$$

Since the activation function can be nonlinear and quantities proportional to the weights pass through them, the model is evidently no longer necessarily linear w.r.t. the weights and, in general, no closed-form Maximum Likelihood solution can be found. Compare this with the linear basis function models from before. Instead of seeking an analytical solution that no longer exists, some sort of gradient-based optimization scheme, as discussed in the previous chapter in the form of SGD, is required calibrate the weights.

When your dataset contains multiple inputs or outputs, this model can easily be extended by including multiple neurons in the input or output layer, and the other procedures stay the same. Generally, the activation function of the outputs $h^{(out)}$ is linear and the activations in hidden layers are of a nonlinear type.

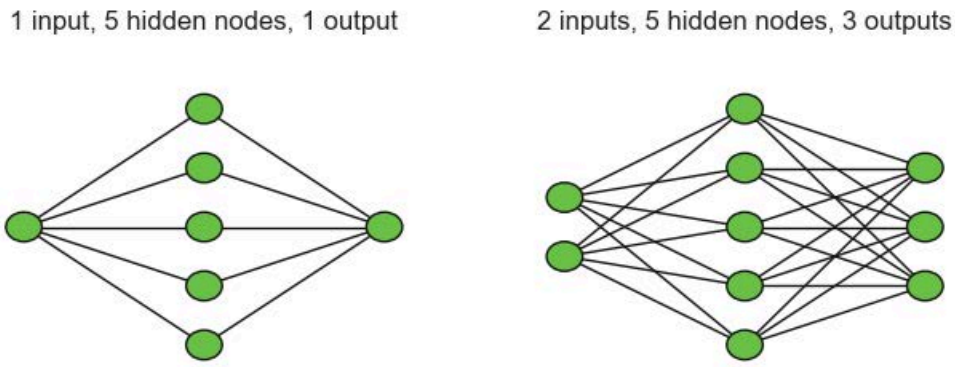



*Fig. 6.39* Two neural networks

**Coding neural nets**

Feedforward neural networks are often also called **Multilayer Perceptrons (MLP)** in machine learning software packages. This is also the case for `scikit-learn`, the package you will be using during the workshop this week. In that package, a neural network model for regression can be instantiated by using the `MLPRegressor` class.

Refer to the documentation of `scikit-learn` to see which arguments you can pass when creating a new `MLPRegressor`. These include the number of layers in the network, the number of units in each layer, the type of activation function, etc.

# Model flexibility

The flexibility of a feedforward neutral network can be adapted by varying the number of neurons per hidden layer, or by adding more hidden layers. Both options lead to an increase

in the number of parameters $\mathbf{w}$. When a neural network has too few parameters, it generally puts us at risk of underfitting the data, whereas having too many parameters can quickly lead to overfitting. Since they control model complexity, the number of layers and neurons per layer are therefore hyperparameters, which need to be calibrated. Once again, remember that **hyperparameters are calibrated with validation data**. Simply minimizing training error w.r.t. these hyperparameters will always lead to huge and severely overfit models, especially when we do not have a lot of data available.

**Coding neural nets**

It is usual to perform a so-called *grid search* when performing model selection for models with more than one hyper-parameter. The strategy is to train a separate model for **each combination** of hyperparameters and pick the one with the lowest value for validation loss. Conceptually, code for this could look something like:

```
hp1vals = [val1, val2, ...]
hp2vals = [val1, val2, ...]
val_losses = np.zeros(len(hp1vals),len(hp2vals))

for i, hp1 in enumerate(hp1vals):
    for j, hp2 in enumerate(hp2vals):
        # build the model
        model = Model(hp1,hp2)

        # train the model
        lowest_val_loss = train_model(model, X_train, t_train, X_val, n_epochs)

        # store loss
        val_losses[i,j] = lowest_val_loss

# pick model with lowest entry in val_losses
```

In the following interactive plot you can study the influence of the number of neurons per layer on model prediciton. The number of hidden layers is fixed at two. You have to click the re-run button to retrain the model after varying the parameter. **Be aware that the required computations can take a few moments to run.** Hide the table of contents to see the full image.

As you might have noticed, using the default setting of three neurons per layer was not enough for the network to learn the underlying trend in the data. Which number of neurons per layer gave you a visibly good fit? Did you ever spot overfitting?

Pay close attention to what we show on the right-hand plot during training:

| Line color | Quantity | Expression |
|---|---|---|
| Blue | Training set loss ($40\%$ of $N$) | $E_D = \frac{1}{N_{\text{train}}} \sum_{n=1}^{N_{\text{train}}} \left(t_n - y(x_n, \mathbf{w})\right)^2$ |
| Purple | Validation set loss ($60\%$ of $N$) | $E_D = \frac{1}{N_{\text{val}}} \sum_{n=1}^{N_{\text{val}}} \left(t_n - y(x_n, \mathbf{w})\right)^2$ |
| Black | Expected loss (numerical integration) | $\mathbb{E}[L] = \int \int \left(t - y(x, \mathbf{w})\right)^2 p(x, t) dx dt$ |

The black line is a very precise version of our error, but one that requires a lot of data to obtain. Since analytical integration is not possible due to the complex nature of $t$ and $y(x, \mathbf{w})$, we are forced to compute it through numerical integration, in this case by using $1500$ equally-spaced predictions in the range $[0, 2\pi]$. Although interesting to include here for educational purposes, it is obvious that **we will not have access to this measure in practice**.

What we have instead is the validation loss (purple line) computed as a relatively crude Monte Carlo approximation of the expected loss, since we are only using a very small number of data points to compute it. In practice, we need to rely on this approximation to make model selection decisions!

**Coding neural nets**

As with any other model this week, we need to perform a training-validation-test split to our original dataset in order to arrive at a robust machine learning estimator for $y(x)$. A simple (incomplete) implementation of this procedure could look like:

```
import numpy as np

# fix the seed of the RNG, for reproducibility
np.random.seed(42)

# get an index array with randomized order
perm = np.random.permutation(N)

# compute the number of samples for training/validation/test
n_train = ?
n_val = ?
n_test = ?

# slice the permutation vector from before
idcs_train = perm[0:ntrain]
idcs_val = perm[?:?]
idcs_test = perm[?:?]

# use the indices to slice X and t
X_train = X[idcs_train]
t_train = t[idcs_train]
# ...
```

# Early stopping

Choosing a high number of neurons increases the number of parameters and, therefore, slows down training. In addition, it can make the model too flexible and prone to overfitting. It is therefore good practive to always monitor the predictive capability of a NN on a validation set. First, run the model below, then, select the model you think best fits the data by pulling the corresponding slider. At which epoch do you find the best model?

The example above is a good illustration of overfitting. Note how the training loss almost immediately becomes an unreliable estimate of the true expected loss (black line). In contrast, the validation loss (purple line) does not exactly agree with the black line but consistently follows the same trends. Crucially, **the validation loss correctly points to the model with the lowest true loss**, at around $3000$ training epochs. Move the model selection slider there and try to see why the true loss starts to increase from that point on.

It is possible to train a neural network for a long time, and then select the $\mathbf{w}$ that corresponds to the lowest validation error. An alternative, known as early stopping, uses the indication that the validation loss increases for a number of epochs as a stopping sign to halt training.

### Coding neural nets

To make early stopping work, we need to continuously monitor the validation loss, either every time the model is updated or at least once every epoch. Together with the fact that we train with **mini-batches** of our dataset (as explained in the video), this means we need to code a training loop that implements our SGD routine.

In `scikit-learn` we could call the `fit` function to fully train a network. However, in order to properly implement early stopping, keep track of our validation loss and see which mini-batches are being picked, we could code our own training loop. To accommodate for this, the `MLPRegressor` model offers the `partial_fit` function, that retrains the network only once for a given pair of `X_batch` and `t_batch`. Then one possible (incomplete) implementation could look like:

```
def train_model(model, X_train, t_train, X_val, n_epochs):
    for epoch in range(n_epochs):
        # slice the dataset into mini-batches
        batches = slice_dataset(X_train,t_train)

        for X_batch, t_batch in batches:
            # update network weights
            model.partial_fit(X_batch, t_batch)

        # compute the relevant losses
        train_y = model.predict(X_train)
        val_y   = model.predict(X_val)

        train_loss = loss_function(train_t,train_y)
        val_loss = loss_function(val_t,val_y)

        # append losses to the relevant lists
        t_losses.append(train_loss)
        v_losses.append(val_loss)
    return t_losse, v_losses
```

where care should be of course taken to properly normalize the datasets. The losses can then be plotted, the model with the best validation loss can be picked, etc.

# Manual model selection

In Linear Basis Function Models and Regularization we used $L_2$-regularization to control the model complexity. The application of this technique to neural networks is straightforward, and will therefore not be demonstrated here. Instead, we will focus on the impact of the number of trainable parameters and the number of samples on overfitting. The ability to display our models at different stages of the training phase will help us to find and inspect particularily good or bad models.

Train a number of different neural networks with varying hyperparameter settings, and try to understand the influence of all the the parameters on the resulting model. Try to answer the following questions:

- Is the model with the lowest validation error always the one that gives the best fit visually?

**Answer**

There is more to this question than meets the eye. Remember in practice we do not know what the ground truth is. Also remember that the validation set is sometimes just a crude approximation of the true loss. For specific settings of the sliders above, it will sometimes look like models with higher validation error are actually visually better when looking at the ground truth.

- Try out a model with linear (*identity*) activation function. Can you make sense of what you observe?

**Answer**

Recall the expression for the predictions of a neural network. Setting the activation function to linear will make the model collapse back into linear regression! Remember that even though we have a lot of weights here, they just get multiplied with each other and lead to equivalent weights for a very simple linear model.

- We plot our activation functions next to their selector buttons. How does the shape of your trained model correlate with the shape of your activation function?

**Answer**

How each activation function handles nonlinearities clearly affects our final model. Note how the *ReLU* function (bilinear) leads to a piecewise linear model in the end.

- In practical situations it is often difficult to visualize model predictions for the whole domain. Can you detect when a model is underfit based only on the training and validation loss?

**Answer**

This can be quite tricky in practice. A somewhat reliable sign would be to look at how the training and validation losses change with training epochs. If there is basically no change after a small number of epochs, it might pay off to try a more flexible model and see what happens. Of course, when doing this you should always keep an eye out for overfitting.

- For a well-trained flexible model with a large training size (N) the errors usually converge to a specific value. What is this value and why does this happen? Can we ever get rid of it?

**Answer**

Remember the discussion on bias/variance decomposition from before. We can decompose our loss into three parts: squared bias, variance and irreducible noise. The irreducible part of the loss will therefore always be there even if we can achieve the idealized model $h(x) = \mathbb{E}[t|x]$ (by having enough flexibility and a lot of data to train the model with). Unless we find a way to explain this irreducible observation noise (e.g. by observing another variable of interest), we can never get rid of it.

- How well does the model predict **outside of the training range**?

**Answer**

Once again, machine learning models with purely data-driven bias/variance should be used in extrapolation with extreme care. Here we see how complex models give senseless predictions outside of their training range. More advanced deep learning models can somehow break this curse and perform well in extrapolation, under specific choices of architecture, datasets and training procedures. Look online for “zero-shot learning” if you are curious about this.

# Wrap-up

In these videos you have seen a non-parametric model, namely k-nearest neighbours, and have learned about the bias-variance trade-off. Linear regression was shown as a parametric model that is linear in its parameters and has a closed form solution. Ridge regression has been introduced to prevent overfitting. Stochastic gradient descent has been shown as a way to train a model in an iterative fashion. In this final chapter, we explored a model with a nonlinear dependence on its parameters. You now understand the underlying principles of a broad set machine learning techniques, and know how to distinguish naive curve fitting from extracting (or learning) the relevant trends and patterns in data.

**Attribution**

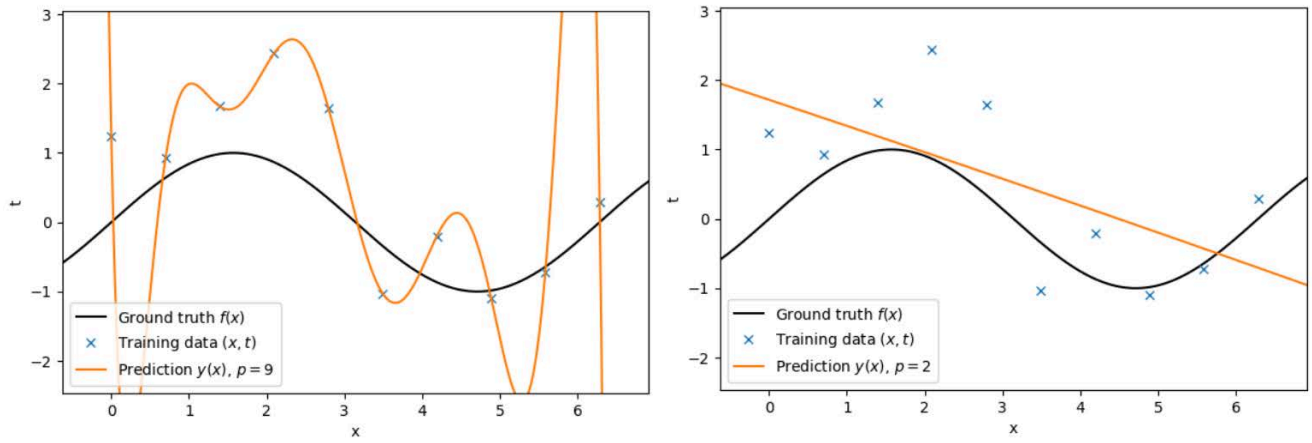
This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

# Review and Quiz

## Contents


- Multiple choice questions:
- Exam example question


## Multiple choice questions:




Look at the two models above. Both are trained with $N = 10$ data points but with models with
different levels of complexity. From the options below, mark all that apply:

- [ ] The model on the right is much less sensitive to specific realizations of a $N = 10$ dataset than the model on the left
- [ ] Comparing the training loss of both models, the one on the right would have a much lower value.
- [ ] The model on the left has high bias and very low variance
- [ ] The model on the left is more sensitive to noise than the one on the right

Check

# Exam example question

In the exam you will also be tested on open questions. Here is an example question:

**Open question 1:**

What are the reducible and the irreducible loss?

Example answer:

The irreducible loss represents the variance that will always be present in our model because we make noisy observations. Even if we had the perfect model, we would still have an expected loss greater than zero due to the noise inherent in our data. The reducible loss is the difference between our current model and the perfect model.

> **Attribution**
>
> This chapter is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius. Find out more here.

# Extreme Value Analysis

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre. Find out more here.

The preceding chapters have focused on a variety of data-driven and physics-based modelling techniques which we primarily used to interpolate between known data (e.g., machine learning), or make predictions about phenomena where randomness did not play a significant role (e.g., finite volume or finite element methods applied to simple physics problems). Most of the methods and problems considered ignored (or greatly simplified) the stochastic nature of the underlying processes. When uncertainty was considered explicitly, it focused primarily on error and epistemic types, for example, the inclusion of various types of noise in Time Series Analysis, or measurement precision in Observation Theory. In these cases, the focus was on applications that were governed by variations around a central value (as opposed to the tails of the distribution), which are often modelled sufficiently using a Gaussian distribution.

The chapter on Continuous Distributions introduced additional asymmetric parametric distributions, such as the Gumbel or Exponential, which are better able to represent the observations that have a small frequency in a data set (i.e., rare events). Regardless of their flexibility, these distributions are not possible to validate for cases where data simply does not exist, which is where concepts introduced in this chapter become useful.

Continuous parametric distributions are relatively simple models to apply, and allow one to make inferences of values of the modelled random variable that occur infrequently, or not at all, within the available observations (i.e., the data set) due to a key concept: the *tail of the distribution.* When extrapolating to values outside a set of observations, fitting of the parametric distribution to the tail is crucial to provide a reasonable extrapolation. Consider the following figure, which was covered earlier in this book:

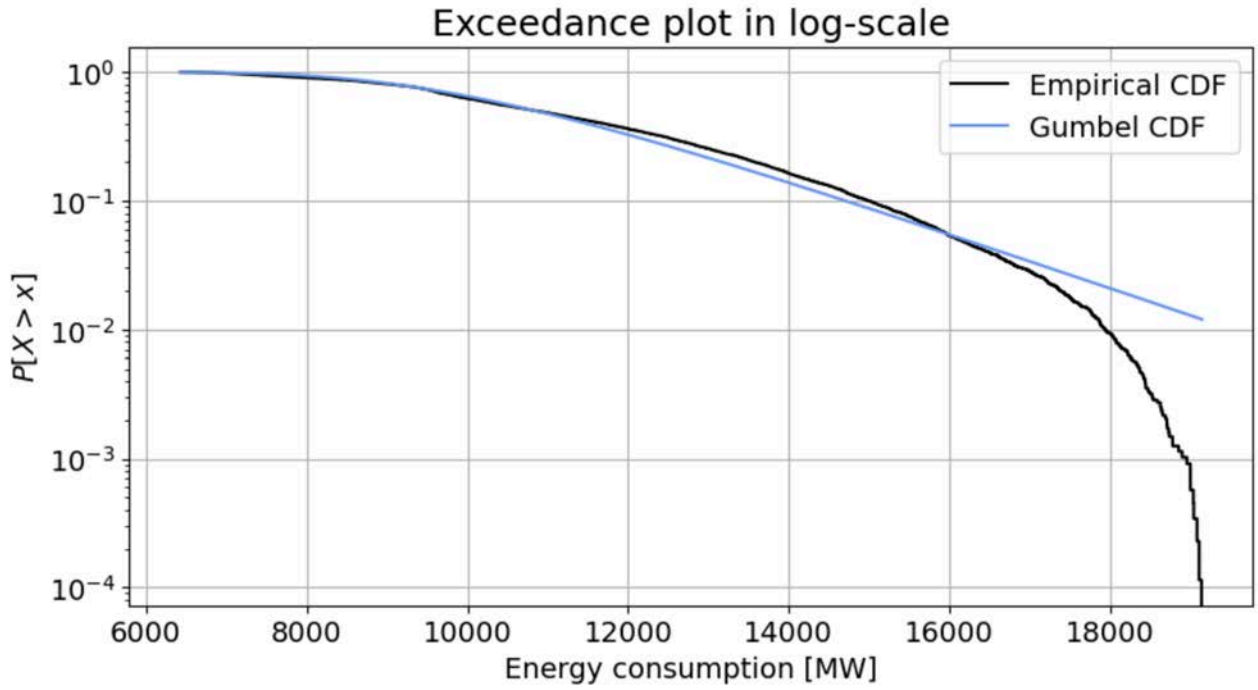



*Fig. 7.1* Exceedance plot in semi-log scale comparing a continuous parametric distribution to data of energy consumption. Note in particular the divergence between data and model above 16,000 MW, or below probability of exceedance of 0.06.

In the example above right-tailed Gumbel distribution was fit to observations on energy consumption, where the Gumbel distribution was selected to represent the positive skewness (right tail) present in the data. While the behavior around the central moments was well represented, the tail was not properly modelled, which is visible in the divergence between data and model on the right-hand side of the plot.

**But what happens if my main interest is the tail?**

In many engineering applications there is typically an interest in the tails of the distributions. For instance, flood protection systems will be designed to withstand extreme rainfall events or extreme river discharges (low exceedance probabilities), not only daily conditions (high exceedance probabilities); these extreme events are located in the tails of the distribution. Moreover, by definition, extreme events are typically scarce in our datasets, as they occur infrequently. The available time series are usually short (e.g., 20 years) in comparison with the design events that the system needs to withstand (e.g., 1,000 years event).

**Definition**

**Extreme Value Analysis** (EVA) focuses on those events located at the tails of the distribution (extreme events) and provides a framework to identify and model the stochastic behavior of these extreme events such that events which have not been observed can be inferred.

In the following chapters, the formal concept of *extreme* will be introduced, as well as the techniques to sample them, without data, and probabilistically model them.

**Extreme Value Analysis Outline**

We will cover two methods of performing an extreme value analysis: **block maxima** and **peak over threshold,** both of which are applied in the same way:

1. Identify a random variable of interest that can be modelled with a continuous parametric probability distribution
2. Prepare and explore the data set that will be used to fit a distribution
3. **Sample** the data to collect a set of the *extreme* observations
4. Fit a distribution to the sampled set of data
5. Repeat the sampling and fitting process as needed, evaluating the alternatives, until a suitable model is identified

As you can see above, only step 3 is a significant departure from what was covered earlier in the continuous distribution chapters.

**MUDE exam information**

You are expected to understand the general motivation of extreme value analysis and be able to use it to choose and use an appropriate distribution for making inferences about a random variable of interest. It is important to know the key steps of the methods in general (for example, the sampling process), as well as the differences and advantages/disadvantages of block maxima and peak over threshold approaches.

Boxes such as this are used throughout the chapters to indicate where specific theory is provided only as background information (i.e., not required for the exam).

At the end of these chapters you can also find supplementary videos that cover the same material as the book (although there is not a 1-to-1 relationship between pages and videos).

# Concept of Extremes

If you hear the word “**extreme**”, the first thing that may come to your mind are extreme sports or natural disasters, such as a hurricane or a typhon. That gives us an intuition of what is an extreme observation in probability theory. Let’s see it in further detail with a dummy example.

We want to perform a market study of the prices of sold t-shirts to start a new clothing brand. We perform a survey and build the probability density function (pdf) of prices that we can see below. The peak of the distribution, the mode, would correspond to fast fashion brands since they are afordable and sell a lot of product. If we start moving towards the right tail of the distribution, high end brands appear, which are a bit more expensive but still sell a reasonable amount of product. If we keep moving towards the right, we will find luxury brands which are really exclusive, so they sell little amount of product at very high prices. Finally, even further in the tail, we would find the world record: the most expensive t-shirt sold in the world, the one which Maradona wore when did the goal with the “hand of God” help, sold by ≈ $9M. It is extreme, right?

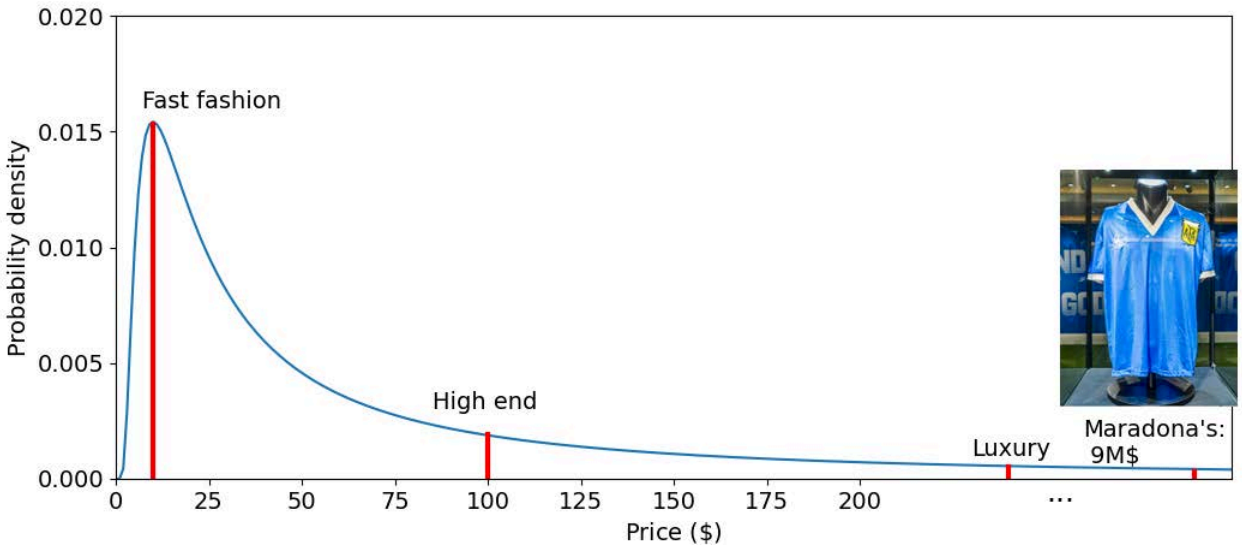



*Fig. 7.2* Pdf of t-shirts price.

Based on the above example, we can define an extreme in probability theory as…

**Extreme observation**

An extreme is an observation which deviates from the mean and it is, thus, located in the tail of the distribution.

Note that extremes are not only maxima (e.g.: maximum river discharge or maximum traffic load), but also minima (e.g.: droughts or minimum energy consumption in a network) depending on the direction that we move within the distribution function.

**Why are we interested in extremes?**

As engineers or geoscientist, we design interventions and infrastructures to withstand scenarios which are linked to extreme conditions (does it ring the bell “Ultimate Limit State”?). For instance, if we are designing a bridge, we will be interested not only on the daily loads of the cars, but also on the maximum loads that the bridge will face along its design life (e.g.: several large trucks crossing at the same time).

**And how can we assess extreme conditions?**

Extreme Value Analysis (EVA) allows us to quantify the needed extremes for design. Typically, we have (limited) historical data (e.g.: few recorded years of traffic loads) and we need to quantify extremes which haven’t been observed yet. EVA allows us to model the stochastic behavior of extreme events and infer those which haven’t been observed (extrapolate).

For instance, going back to our toy example of the t-shirts, EVA allows us to get a better understanding of how the expensive t-shirts behave. We focus on modelling the expensive t-shirts and forget about the cheap options. The probabilistic model developed with EVA will allow us to gain information about prices higher than those already recorded in our dataset and gain insight on how the market might evolve.

In the following sections, you will see how to select extreme observations within a database (timeseries of observations) and select, fit and use probability distribution functions to characterize their uncertainty and infer the needed extreme values for design.

# Return Period

## Contents


- Concept of return period
- Let's practice!


> But how extreme do I need my design conditions to be?

The Return Period ($RT$)[1] is a widely used concept in guidelines and recommendations in the Engineering and Geosciences field to parameterize the safety level that a given structure or intervention needs to fulfill. Structures are designed to withstand external loadings (e.g.: floods or wave storms) whose distribution function is quantified using Extreme Value Analysis. The magnitude of the extreme event that the structure needs to withstand is then determined according to a given $RT$.

## Concept of return period

In short, $RT$ is the expected time between two exceedances of extreme events. This is, if I have a river discharge $q = 100m^3/s$ with a $RT = 5\ years$, it means that, on average, I will observe river discharges which exceed $100m^3/s$ each 5 years.

**But how can I calculate it? Let's derive it!**

We want to estimate the time, on average, at which an event higher than a given value or threshold (design value, $z_q$) occurs, and we know that the exceedance probability is $P[X > z_q] = p$. We will assume that the process is stationary, so $p$ remains constant.

So, for each time unit (trial), we check if the observed value exceeds our design value (success) or not (failure). Does it ring the bell? **It's a Bernoulli process!**

The probability that an event $z_0$ higher that $z_q$ occurs at a generic year $t$ is calculated as follows. We don't exceed the threshold in $t - 1$ trials, so we have to multiply by $(1 - p)(1 - p) \dots t - 1$ times. We exceed the design value at time $t$, so we multiply by $p$.

Note that we are assuming that each trial is independent, so exceeding the threshold in time $t$ is not dependent on the previous trials.

$$f(t) = P[z_0 \, at \, t] = (1-p)(1-p)\dots(1-p)p = (1-p)^{t-1}p$$

The obtained equation is a Geometric distribution which models the number of trials up to the first success. If we calculate the expectation of such distribution (the expected number of trials until we exceed the design value), we obtain the definition of return period

$$RT(t) = 1/p$$

So we have to wait 1/p times before the next occurrence of the extreme event.

# Let's practice!

An engineer wants to design a water treatment plant. Within that design process, the engineer needs to determine the maximum design concentration of nitrogen. According to regulations, a return period $RT = 25$ years has to be considered.

What is the yearly non-exceedance probability associated to that design return period?

**Answer**

The yearly exceedance probability is the inverse of the return period so

$$P[X > x] = 1/25 = 0.04$$

Thus, the yearly non-exceedance probability is

$$P[X \leq x] = 1 - P[X > x] = 1 - 0.04 = 0.96$$

---

For a more formal definition of the concept of return period, you can refer to Serinaldi (2015)

# Sampling Techniques

## Contents


- How can I select the extreme observations within my timeseries?


> Why do we need sampling techniques for extremes?

In the previous sections, you have seen the basic concepts needed to address Extreme Value Analysis (EVA), such as what is an extreme observation or the return period ($RT$). Here, we will start the former EVA by presenting the sampling techniques for extremes. In order to illustrate it, we will use a case study: a breakwater design. Note that this procedure can be extrapolated to any loading variable.

Breakwaters are structures which protect the basins of ports and have to withstand wave storms. Thus, the main loading is the significant wave height ($H_s$) and we need to determine $H_s$ for a given $RT$, for instance $RT = 100\ years$. For our case of study, we are going to use a time series of $H_s$ recorded by a buoy in the Mediterranean sea. In such buoy, we have 20 years of hourly recordings. In the image below, four of those years are plotted.

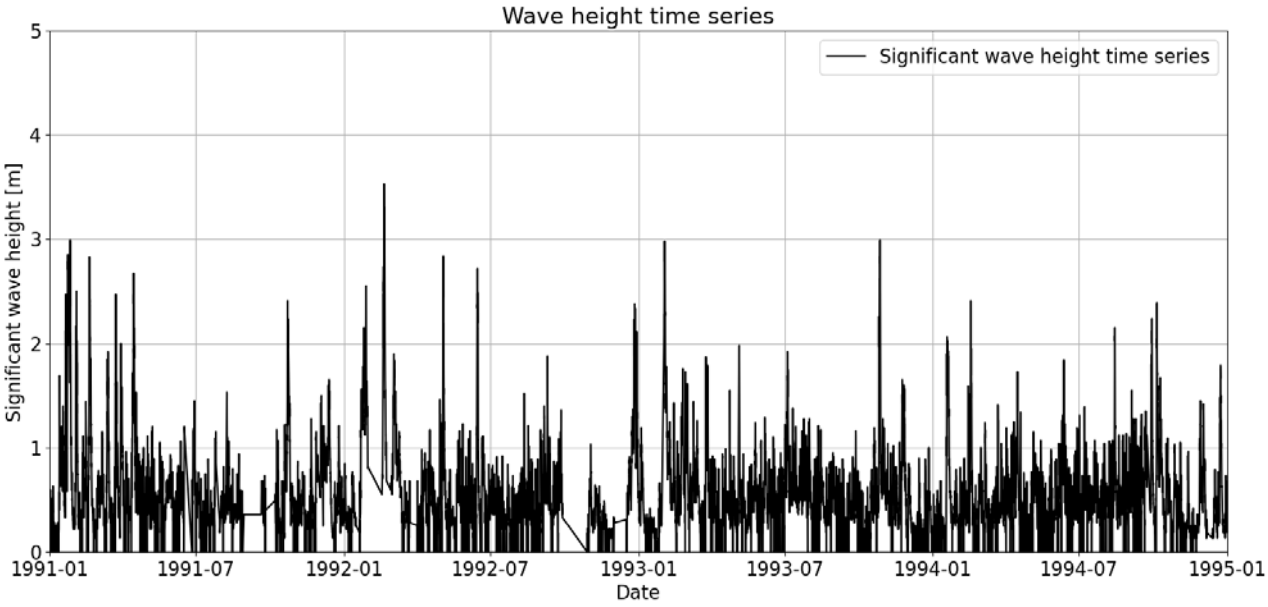



*Fig. 7.3* Significant wave height ($H_s$) time series.

# How can I select the extreme observations within my timeseries?

Once we have the time series, the question arises: which values should I use for my analysis? Not all of them are extremes, so I need to extract them from my historical data.

To this end, we can use two sampling techniques:

1. Block Maxima (BM)
2. Peak Over Threshold (POT)

# Block Maxima & GEV

One approach to Extreme Value Analysis is Block Maxima: where a time series is broken into equal units of time, and the maximum observation in each unit is selected. In this chapter we will see that the set of maximum values from each unit (the *samples*) is theoretically linked to the Generalized Extreme Value (GEV) distribution.

# Block Maxima

## Contents


- Advantages and considerations for use
- Let's code it


Block maxima consists of defining a time block, for instance a year, and selecting the maximum observation within that time block. Thus, the number of sampled extremes would be equal to the number of blocks.

Let's apply it to our time series. We are going to use a block = 1 year, so we will obtain 20 extreme observations for the subsequent phases of our analysis. In the figure below, you can see how the application of this method looks.

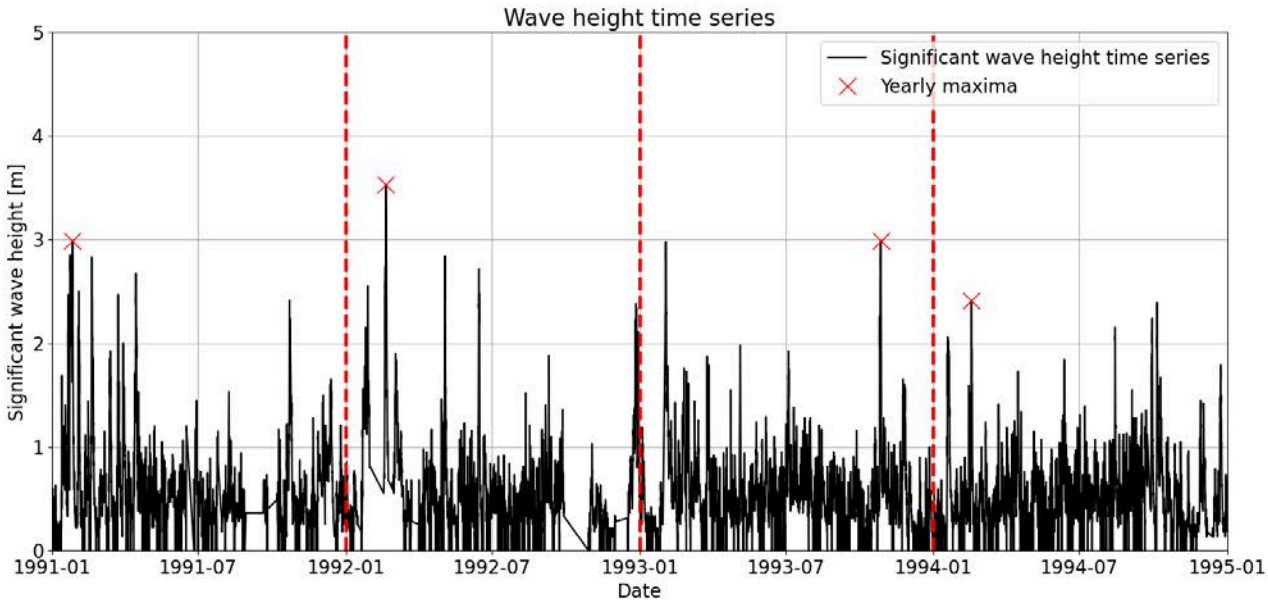



*Fig. 7.4* Application of Block Maxima to $H_s$ time series with a time block of 1 year (Yearly Maxima).

# Advantages and considerations for use

Typically, in the Civil Engineering field, time blocks of 1 year are used, since natural phenomena with annual seasonality are usually studied (e.g., wave storms or floods). This

also simplifies the quantification of yearly exceedance probabilities, as you will see in subsequent sections.

When introducing the concept of $RT$, we saw that one of the basic assumptions of EVA was that extremes follow a Bernoulli process and, thus, are independent and identically distributed (*iid* conditions). This implies that when sampling our extremes, we need to ensure that they are independent. At this point is where Block Maxima has a great advantage: if the time block is long enough, we can guarantee that the sampled observations are independent. For instance, if we use Yearly Maxima (one observation each year), we can be sure that the extreme observation of the year $y$ is independent from the extreme observations in years $y-1$ and $y+1$. Also, it is very easy to implement.

**Block Maxima**

Advantages:

- Easy to guarantee that sampled extremes are *iid*.
- Simple implementation and coding.

Disadvantages:

- Only one observation per block, so we may be neglecting extreme observations located within the same time block.
- Not appropriate for short time series.

# Let's code it

In order to exemplify how to actually implement Block Maxima,pseudo code is presented.

## Pseudo code

```
read observations

for each year i:
    obs_max[i] = maximum observations in year i
end
```

# Asymptotic Model

## Contents


- Let's see it with an example!
- In pseudo code


In the previous section, we have seen the first technique to sample extreme observations: Block Maxima. But which distribution do these extreme observations follow? In order to better answer this question, here we introduce the asymptotic model.

**Asymptotic model**

- Given $X = (X_1, X_2, \ldots, X_n)$ as a sequence of independent and identically distributed random variables which follow the distribution function $F(x)$.
- Being the maximum of the process $M_n = max(X_1, X_2, \ldots, X_n)$

The theoretical distribution of $M_n$ is $F(x)^n$

We can prove it by calculating the probability of the maxima of the sequence ($M_n$) being lower or equal than $x$. If the maximum value of a sequence is below a given $x$, all the values of the sequence will also be below such $x$. Thus, it is the same as calculating the probability of each value of the sequence $X$ being lower of equal than $x$. Since we have stated that $X$ is a sequence of independent and identically distributed observations, the joint probability can be computed as the product of the individual probabilities. Hence, the distribution of the extremes is $F(x)^n$.

$$P[M_n \leq x] = P[max(X) \leq x] = P[X_1 \leq x, \ldots, X_n \leq x] = P[X_1 \leq x] \ldots P[X_n \leq x$$

## Let's see it with an example!

We are going to generate 100 samples with length N = 30 using a Normal distribution $N(6, 1)$. From each sample (block), we are going to store the maximum observation ($x_{max}$). Note that we are applying Block Maxima here. In the figure below, you can see in grey the full samples and in blue the maxima from each sample.

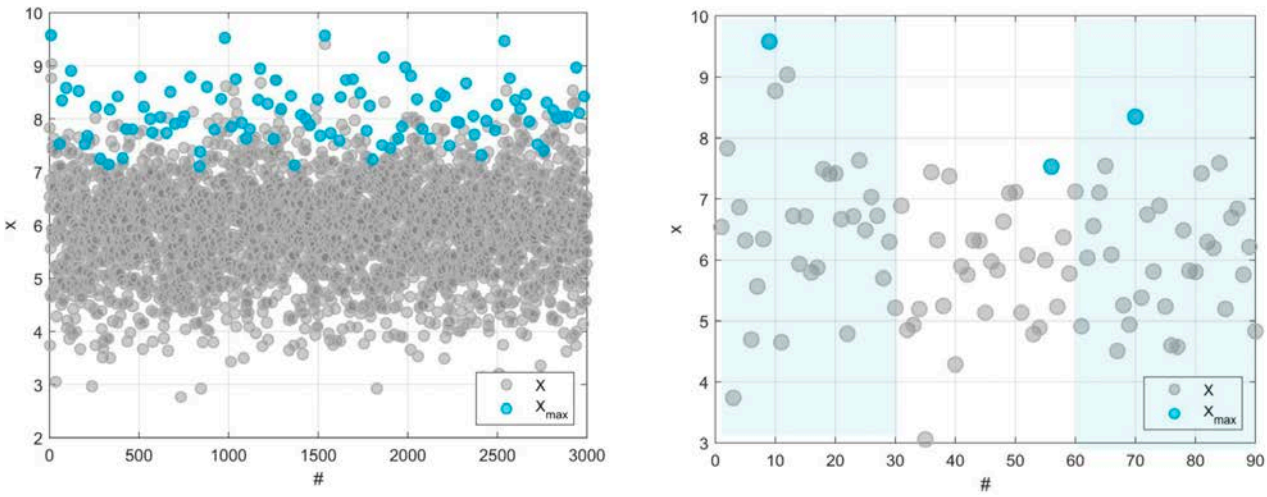



*Fig. 7.5* Random samples using $N(6,1)$ and maxima from each sample.

If we plot the histogram of the samples and the plot of the maxima from each sample (see left image below), we can see that the histogram of the maxima is placed on the right tail of the histogram build with all the samples. Also, in the empirical cumulative distribution function (cdf) plot, we can see that the samples follow a $F(x) \sim N(6,1)$ as expected, while the maxima follow $F(x)^n$ as we previously stated.

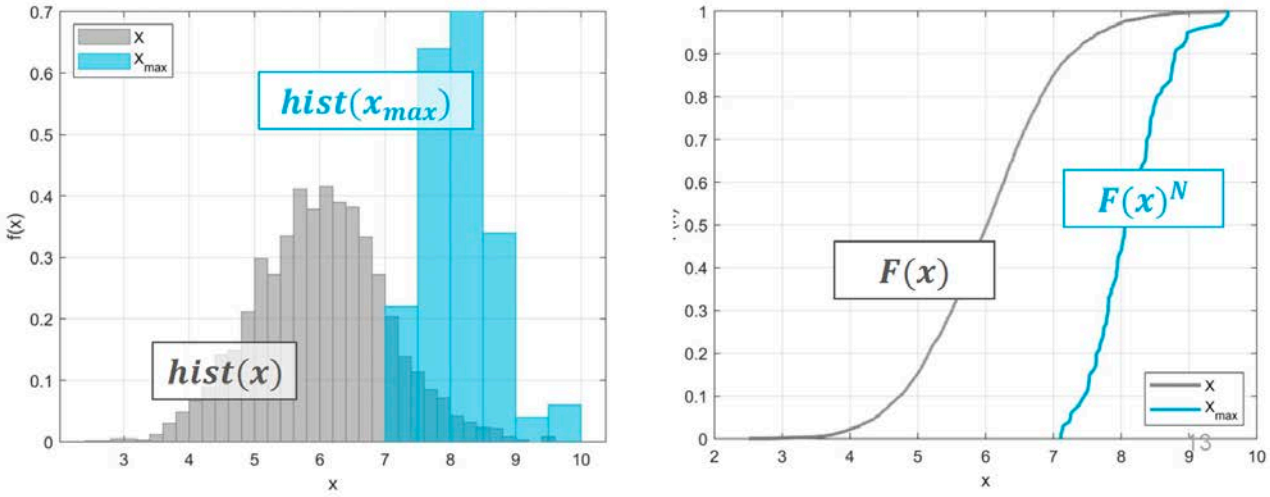



*Fig. 7.6* Histograms of the random samples and their maxima.

# In pseudo code

If you want to repeat that analysis yourself, you can do it implementing the following pseudo code.

```
mean = 6
standard deviation = 1
N = 30
for i in range(100):
    x[i] = N normal samples with mean and standard deviation
    x_max[i] = maximum in x[i]
end

plot histogram(x), histogram(x_max)
plot ecdf(x), ecdf(x_max)
```

# GEV Distribution

## Contents


- Definition and parameters of GEV distribution
- Domains of attraction
- Let’s apply it!
- Let’s practice!


After sampling the extreme observations in our time series, we want to model the statistical behavior of those maxima. Thus, we are interested in modelling the maximum of the sequence $X = X_1, \ldots, X_n$, $M_n = max(X_1, \ldots, X_n)$, where $n$ is the number of observations in a given block. Assuming that those maxima are independent and identically distributed, we can prove that for large $n$ the cumulative distribution function of **those maxima tend to the Generalized Extreme Value (GEV) family of distributions, regardless the distribution of** $X$.

$P[M_n \leq x] \rightarrow G(x)$

## Definition and parameters of GEV distribution

Generalized Extreme Value distribution is defined as

$G(x) = exp\big(-[1 + \xi\frac{x-\mu}{\sigma}]^{-1/\xi}\big) \qquad (1 + \xi\frac{x-\mu}{\sigma}) > 0$

where $-\infty < \mu < \infty$ is the location parameter, $\sigma > 0$ is the scale parameter, and $-\infty < \xi < \infty$ is the shape parameter. Let’s see the effect of these parameters on the distribution function.

In the figure below, you can see the effect of the location parameter, $\mu$. A higher value of $\mu$ leads to a shift towards the right.

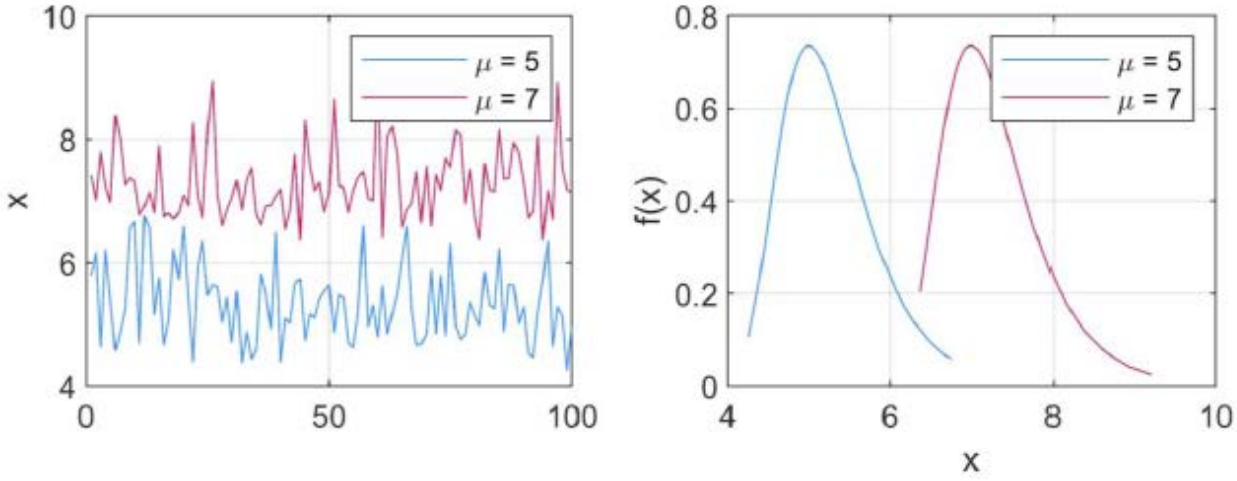



*Fig. 7.7* Influence of the location parameter in GEV distribution.

A higher value of the scale parameter, $\sigma$, leads to a wider distribution, as shown below.

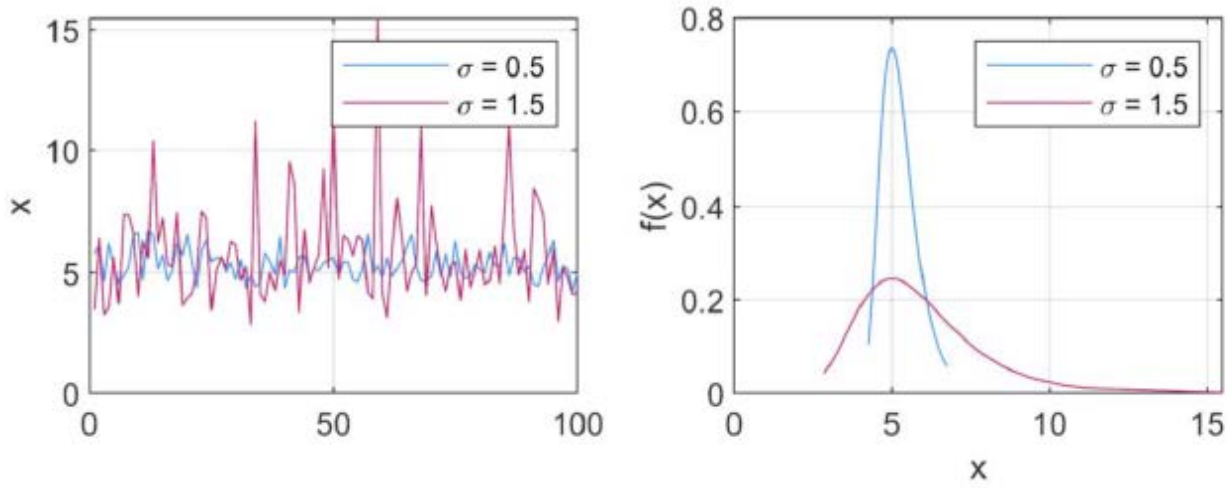



*Fig. 7.8* Influence of the scale parameter in GEV distribution.

Finally, the shape parameter, $\xi$, determines the tail of the distribution. This is, the behavior of the distribution at the tail changes based on the value of $\xi$. If $\xi \to 0$, the GEV is called type I or Gumbel distribution and presents an exponential decay at the tail. If $\xi > 0$, the GEV is a type II or Fréchet distribution and the tail presents a polynomial decay. Finally, if \x $< 0$, the GEV is a type III or Reverse Weibull distribution, and it presents an upper bound.

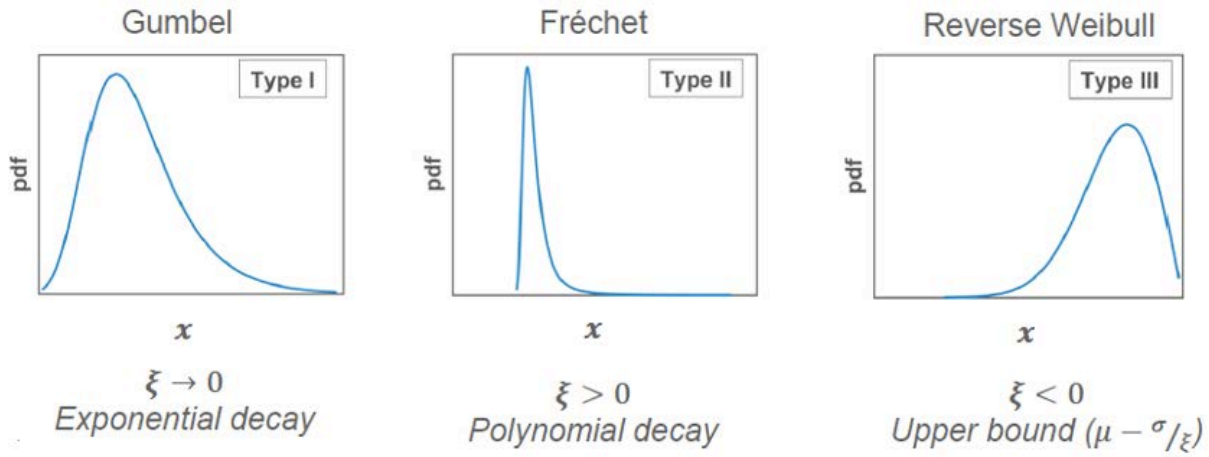



*Fig. 7.9* Influence of the shape parameter in GEV distribution.

If we plot the exceedance probabilities, we can better see the role of those differences in the tail. Gumbel distribution is said to have a “light tail” since its tail is somewhere in between Reverse Weibull and Frèchet distribution. Small changes in the exceedance probability lead to high differences in the values of the magnitude of interest ($x$ in the plot below) when analyzing the Frèchet distribution. Indeed, we say it has “a heavy tail”. Finally, for the Reverse Weibull we can clearly see the upper bound around $x = 8$.

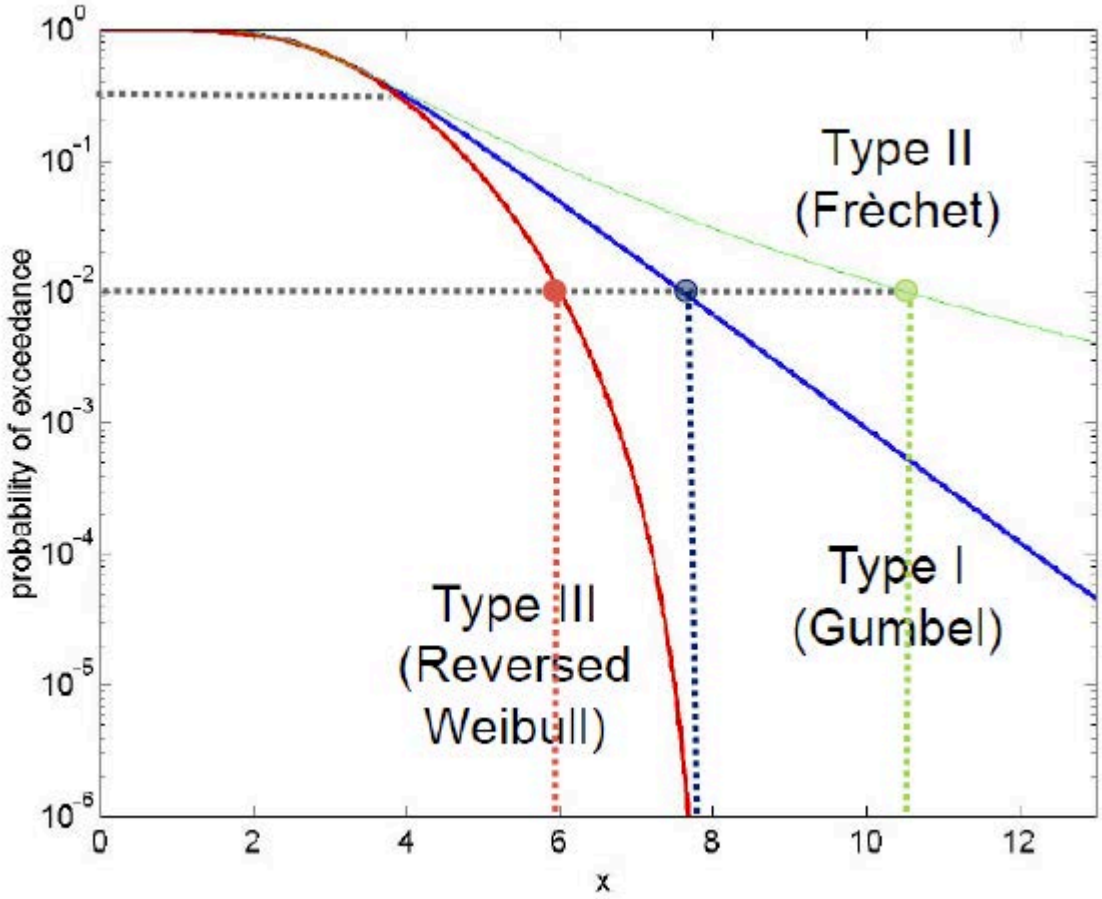



*Fig. 7.10* Tail behavior of GEV distribution.

# Domains of attraction

We said that we can prove that for large $n$ the cumulative distribution function of the values sampled using Block Maxima tend to the Generalized Extreme Value (GEV) family of distributions, regardless the distribution of the $X$ (parent ditribution). However, we can identify domains of attraction between the parent distributions and the type of GEV or asymptotic type.

| **Parent Distribution** | **Asymptotic type maximum** |
|---|---|
| Normal, Exponential, Gamma, Lognormal, Weibull | Gumbel |
| Pareto, Cauchy, Student-t (fat tail) | Fréchet |
| Uniform, Beta (short tail) | Reverse Weibull |

# Let's apply it!

Now that you are familiar with the GEV, we can perform the whole EVA process using Block Maxima and GEV and infer extreme values of $H_s$ that we haven't observed yet. Remember that in our example we were interested on the value of $H_s$ for a $RT = 100\ years$, but we only had 20 years of hourly recordings in our buoy. Thus, we wanted to apply EVA to infer the value of such extreme event for our design. First, we applied Yearly Maxima to sample the extreme observations. Here, we will fit a GEV to those 20 observations and use it to determine the design value of $H_s$. The process is illustrated using pseudo code below:

```
read observations

for each year:
    max observations = maximum observation in year
end

fit GEV(max observations)

check fit (e.g., QQ-plot or Kolmogorov-Smirnov test)

inverse GEV to determine the design event
```

Let's see it now step by step applied to our data. First, we fit a GEV using Maximum Loglikelihood Estimator to our 20 sampled extremes using Yearly Maxima.

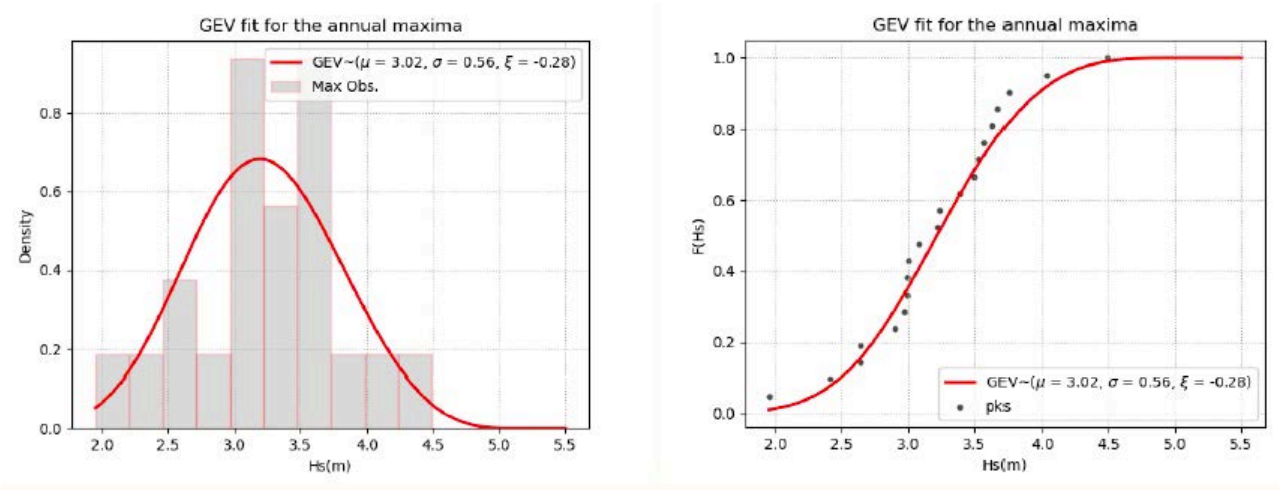



*Fig. 7.11* Fitting GEV to the 20 sampled extremes using Block Maxima.

We can see that $\xi$ is negative, thus it is a GEV type III. Once it is fitted, we need to check how well the theoretical model represents our data. To that end, you can use goodness-of-fit techniques, such as QQ-plot or Kolmogorov-Smirnov test. Here, we assume GEV is a good model and further use it to infer $H_s$ for a $RT = 100\ years$.

$RT = 1/p_{f,y}$, being $p_{f,y}$ the yearly probability of exceedance. Thus, we need to calculate the value of $H_s$ with a $p_{f,y} = 1/100 = 0.01$ or a non-exceedance probability of 0.99. To this end, we can use the inverse of GEV. Working out the equation defining the GEV,

$$z_p = G^{-1}(1 - p_{f,y}) = \begin{cases} \mu - \frac{\sigma}{\xi}[1 - \{-log(1 - p_{f,y})\}^{-\xi}] & for\ \xi \neq 0 \\ \mu - \sigma log\{1 - p_{f,y}\} & for\ \xi = 0 \end{cases}$$

Which leads to $H_{s,RT=100\ years} \approx 4.5m$. **Congratulations! You have performed your first Extreme Value Analysis!**

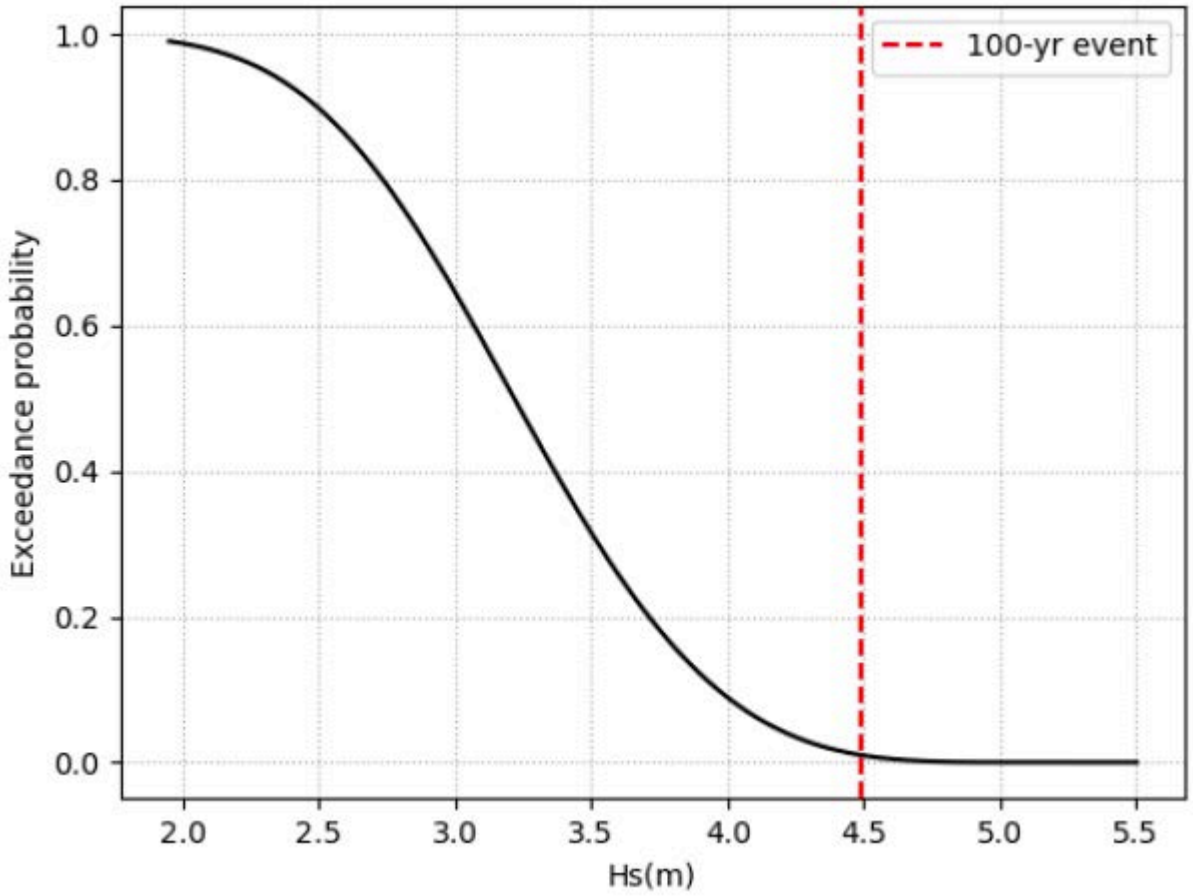



*Fig. 7.12* Return level using GEV.

# Let's practice!

An scientist is assessing the concentrations of atmospheric $CO_2$ in a city to see if they fulfill the tolerability limits for the citizens. According to previous studies, a concentration of 40,000 ppm of $CO_2$ is inmediatly harmful for humans. Thus, the scientist wants to assess the probability of exceeding this concentration.

Based on existing studies, the scientist knows that the distribution of yearly maxima of concentration of $CO_2$ follows a Generalized Extreme Value distribution with $\mu = 5,000$, $\sigma = 250$, and $\xi = 5$.

What is the probability of exceeding 40,000 ppm of $CO_2$?

**Answer**

Applying the CDF of the GEV, we can compute the non-exceedance probability as

$$P[X \leq 40,000] = exp\Big(-[1+\xi\frac{x-\mu}{\sigma}]^{-1/\xi}\Big) = exp\Big(-[1+5\frac{40{,}000-5{,}000}{250}]^{-1/5}\Big) \approx 0$$

Thus, the exceedance probability is

$$P[X > 40,000] = 1 - P[X \leq 40,000] = 1 - 0.76 = 0.24$$

What is the the return period of 40,000 ppm of $CO_2$?

**Answer**

The return period is the inverse of the yearly exceedance probability so

$$RT = 1/P[X > 40,000] = 1/0.24 = 4.23 years$$

Thus, the concentration of 40,000 ppm of $CO_2$ is exceeded in average every 4.23 years.

# Return Period & Design Life

## Contents


- Deriving the probability of failure along the design life
- Let's practice!


In the previous sections,the concept of Return Period ($RT$) and the first sampling technique for extreme observations, Block Maxima, were introduced. Here, $RT$ will be extended to cover the concepts related to the design life in the context of Block Maxima sampling technique.

If we go to guidelines and recommendations, there are four key concepts to parameterize safety in the design phase:

- Probability of failure in one year ($p_{f,y}$).
- Return period ($RT$), usually given in years. According to the previous definition, $RT = 1/p_{f,y}$.
- Design life ($DL$): lifetime of the infrastructure.
- Probability of failure in the design life of the structure ($p_{f,DL}$).

We have already seen $RT$ and $p_{f,y}$. Using Block Maxima with a time block = 1 year (called Yearly Maxima), we can fit a GEV distribution and obtain the non-exceedance yearly probabilities of the desired return levels. However, **how can we obtain the exceedance probability along the design life?**

> **MUDE exam information**
>
> In this section, the derivation of the formula which relates the above mentioned parameters is presented based on the Binomial model. You are not required to know the derivation, it is shown as background information so you know the assumptions which are done in the process. You need to understand the concepts of yearly probability of failure, return period, design life and the probability of failure in the design life. Also, you need to be able to apply the derived equation.

# Deriving the probability of failure along the design life

We already saw that extremes could be assimilated as a Bernoulli process: for each year (trial), we check if the observed value exceeds our design value (success) or not (failure). Thus, the number of exceedances over a design value (successes) in a given number of years (trials) will follow the Binomial distribution if each trial is independent.

Let's calculate the probability of observing an event $z_0$ higher than the design value $z_q$ at least once in $DL$ years ($p_{f,DL}$).

First, we calculate the probability of not failing along $DL$ applying the Binomial distribution as

$$p_X(0) = P[X = 0|DL, p_{f,y}] = \binom{DL}{0} p_{f,y}^0 (1 - p_{f,y})^{DL-0} = (1 - p_{f,y})^{DL}$$

The probability of failing at least once in $DL$ can be computed as $1 - p_X(0)$ (1 - no failure), so

$$p_{f,DL} = 1 - p_X(0) = 1 - (1 - p_{f,y})^{DL}$$

Applying $RT = 1/p_{f,y}$, we can rewrite $RT$ as function of $p_{f,DL}$.

$$p_{f,y} = 1 - (1 - p_{f,DL})^{1/DL}$$

$$RT = 1/p_{f,y} = \frac{1}{1-(1-p_{f,DL})^{1/DL}}$$

And now you can compute the design $RT$ based on the $DL$ and $p_{f,DL}$ recommended in design guidelines!

# Let's practice!

An scientist is designing a *thingamajig*. According to the recommendations, any *thingamajig* should be designed for a lifespan of 25 years with a probability of failure along the design life of $p_{f,DL} = 0.05$.

What is the design return period?

**Answer**

Applying the expression previously derived

$$RT = 1/p_{f,y} = \frac{1}{1-(1-p_{f,DL})^{1/DL}} = \frac{1}{1-(1-0.05)^{1/25}} \approx 488 \text{ years}$$

What is the design non-exceedance yearly probability?

**Answer**

If $RT \approx 488$ years, applying the definition of return period

$$RT = 1/p_{f,y} \rightarrow p_{f,y} = 1/488 = 0.002$$

Thus, the non-exceedance probability is

$$P[X \leq x] = 1 - 0.002 = 0.998$$

# Peak Over Threshold & GPD

A second approach to Extreme Value Analysis is Peak Over Threshold (POT): a threshold value is chosen and all observations in the time series that exceed the threshold are selected. In this chapter we will see that this method is theoretically linked to the Generalized Pareto distribution (GPD).

Compared to BM, the POT method is useful when time series are short and there may not be enough data to fit a distribution to the annual maxima: a 10 year record may have a lot of hourly observations, but will only produce 10 data points if you want to find a distribution for the annual maxima! We will also see that the selection of a threshold (and declustering time) adds considerably to the subjectivitity and judgement required when choosing a distribution to represent, and extrapolate, your data.

Our objective in this book is to introduce key concepts and theorems to give a thorough understanding of POT assumptions and analysis. Keep in mind that in engineering practice and research, there are many variations and alternatives to the methods presented here.

# Peak Over Threshold (POT)

## Contents


- Advantages and considerations for use
- Let's code it
- Let's practice


The second technique for sampling extremes is Peak Over Threshold (POT). This technique consists of selecting the **excesses over a defined threshold** with a minimum distance between them. Thus, it involves defining two parameters: the threshold, and the declustering time (minimum time/distance between two extreme observations). Note that when performing EVA with POT, we model the excesses over the threshold, not the actual value of the extreme event.

Let's apply it to our time series. We are going to use a threshold $th = 2.5m$ and a declustering time $dl = 48h$. This is, we will select the values higher than 2.5m that have a 'distance' between them of at least 48h. In the figure below, you can see how the application of this method looks like.

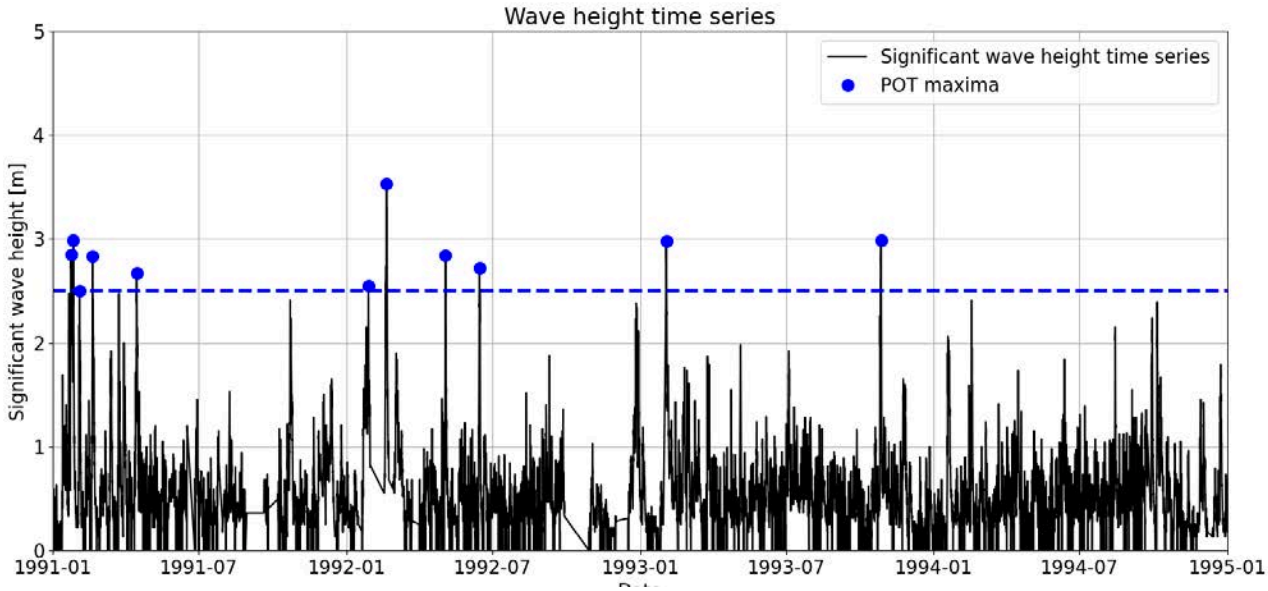



*Fig. 7.13* Application of POT to $H_s$ time series with $th = 2.5m$ and $dl = 48h$.

We can see in the figure a concentration of extremes during the first months of 1991 and 1992, even if we have set $dl = 48h$. This means that there were several storms in the

beginning of the year. When comparing POT with Block Maxima (see Figure below), we can see that more extremes are sampled using POT. The main disadvantage of Block Maxima is derived from its simplicity: only one maxima is sampled in each block. Thus, there may be other extreme observations (storms in this example) which are not sampled within the same block. Consequently, a higher number of extremes is usually sampled when using POT, although its application is a bit more complicated, as we will see in the following sections. For instance, in our example we obtained 20 extreme observations when applying Yearly Maxima (20 years of observations), while we sampled 54 extreme observations when using POT.

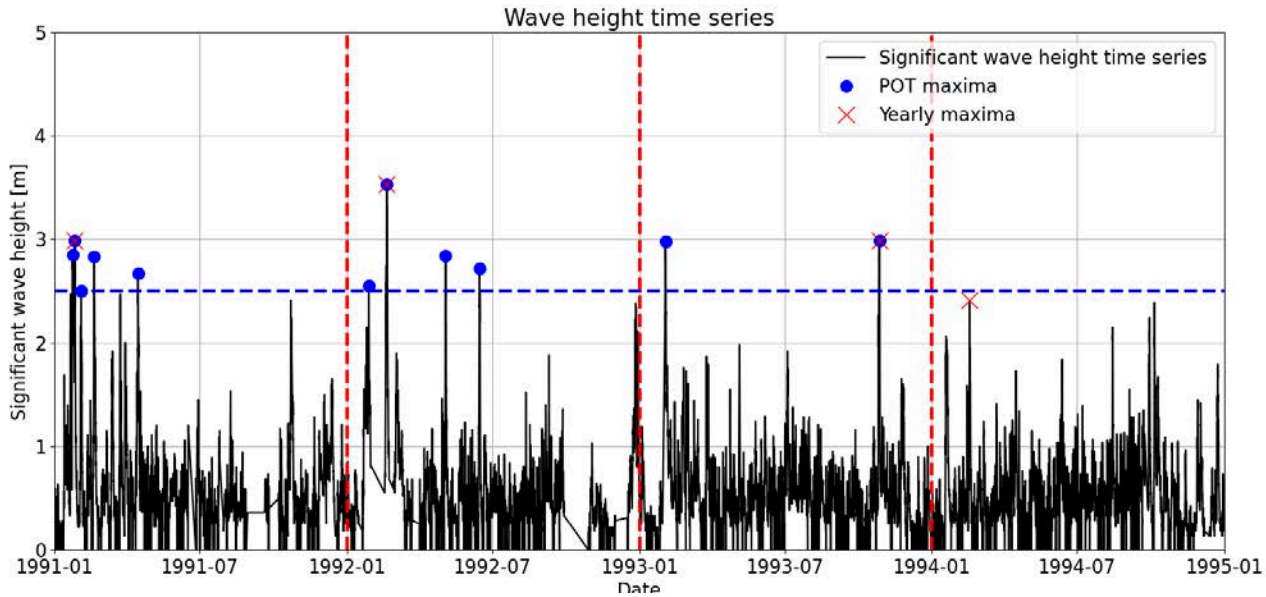



*Fig. 7.14* Comparison of Yearly Maxima and POT to $H_s$ time series with $th = 2.5m$ and $dl = 48h$.

# Advantages and considerations for use

We already saw that POT extracts more information from the time series (higher number of extremes sampled), being that a great advantage when working with short time series. However, its implementation is a bit more tricky.

The results of our analysis are dependent on the parameters we select. For instance, if $th$ is too low, I will be including in the analysis events which cannot be considered extremes. On the other hand, if $th$ is too high, I will be losing information and, thus, not taking advantage of all the power that POT has. In the following sections, we will see the hypothesis that an appropriate $th$ and $dt$ need to fulfill. However, as a rule of thumb, we can start the analysis using $th \approx 90 - 99$ percentile of our observations.

When introducing the concept of $RT$, we saw that one of the basic assumptions of EVA was that extremes are independent and identically distributed (*iid* conditions). This implies that when sampling our extremes, we need to ensure that they are independent. At this point is where POT requires a bit more work than Block Maxima: we need to select $dl$ so we ensure that the sampled observations are independent. Let's elaborate it a bit more.

A storm is a phenomenom which lasts several hours, so if we take a look at the time series of hourly $H_s$ during a storm (see Figure below) we see how registered $H_s$ goes up, stays around the maximum values for a while and, later, goes down. Therefore, we can say that there is a concentration of extreme values in the storms. For instance, there are 5 observations in a row above $th = 2.5m$. However, these extremes are caused by the same drivers and are thus dependent. If we included all of them in our EVA analysis, we would be violating the basic assumption of EVA of *iid* extremes. This phenomenom of concentration of extremes in time is called **clustering**: we say that extremes cluster in time, since an extreme phenomenom is not composed by a single observation.

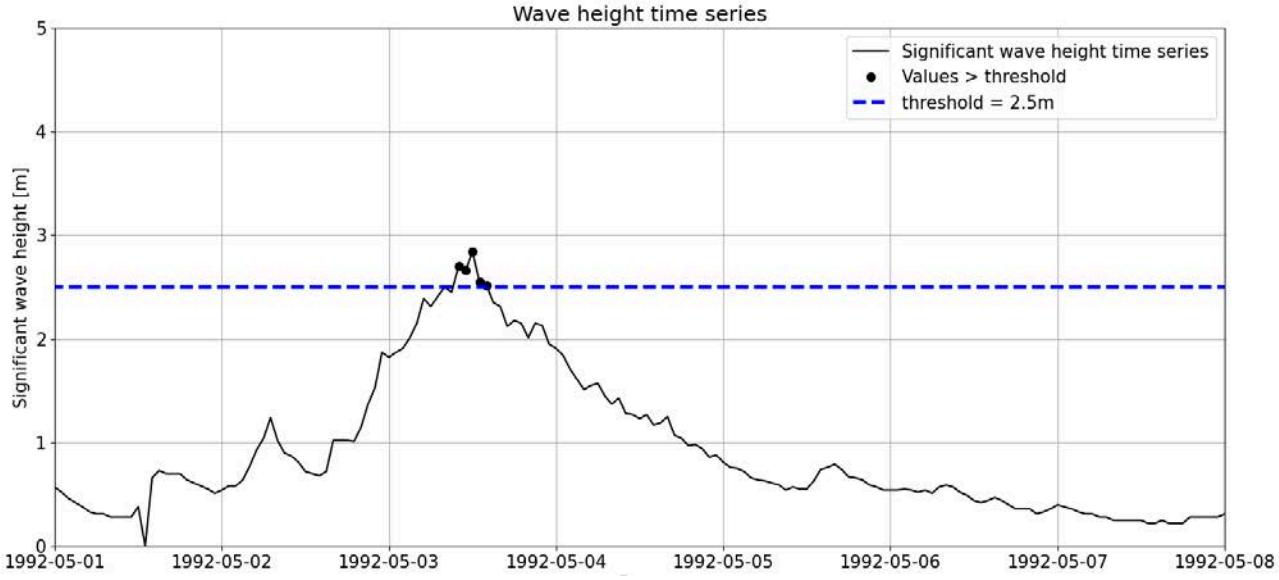



*Fig. 7.15* Comparison of Yearly Maxima and POT to $H_s$ time series with $th = 2.5m$ and $dl = 48h$.

And there is where $dl$ joins the party! If we ensure that the sampled extremes are far enough from each other in time ($dl$ big enough), we can make sure that they do not belong to the same storm, so we can consider them independent. The needed $dl$ depends on the physical phenomena which drive the extreme event that we are analyzing and the local conditions. For instance, an appropriate $dl$ for wave analysis can range from few hours to 72h, since storm duration depends on the local conditions.

**Peak Over Threshold (POT)**

Advantages:

- Usually a higher number of sampled extremes. Thus, it is appropriate for shorter time series.
- Appropriate when there's no clear seasonality in the extremes (physical insight in the phenomenom).

Disadvantages:

- Further analysis required to ensure that the sampled extremes are *iid*.
- Further computations needed, so the process is slower.

# Let's code it

In order to exemplify how to actually implement POT,pseudo code is presented.

## Pseudo code

```
read observations

th = 2.5
dl = 48 #in hours

excesses = (peaks in observations > th with distance >= dl) - th #we model the excesses o
```

# Let's practice

A scientist wants to analyze the extreme concentrations of nitrous oxide ($N_2O$) in the atmosphere as it is listed as one of the greenhouse gases to consider when assessing climate change. The scientist has a timeseries of 10 years. Which sampling method for extremes should he/she apply?

**Answer**

As the scientist has a short timeseries for characterizing extremes (only 10 years), he/she should apply a method which maximizes the information extracted from the data. If the scientist applies BM with a time block of one year, only 10 observations are sampled, which is very little information to quantify the distribution of extremes. Thus, the scientist should apply POT.

Another option would be to apply BM with a time block of a **smaller size** sampling then, for instance, the monthly maxima. Note that then the fitted GEV will provide the **monthly** non-exceedance probabilities.

To choose between these two options, we should consider criteria related to the physical process we are studying, such as whether the phenomenon presents seasonality or cycles along time.

# Intermezzo: Poisson

## Contents


- Deriving the Distribution
- Poisson distribution and Poisson process
- Poisson process to model extremes sampled with POT


As you already know, the Binomial distribution models the probability of $x$ successes in $n$ Bernoulli trials with a probability of success in each trial, $p$. When $n$ is large and $p$ is small, the Binomial distribution can be approximated by a Poisson distribution. Thus, the Poisson distribution is also based on the assumption of independence and identical distribution.

> **MUDE exam information**
>
> In this section, theory regarding the Poisson distribution is provided as a refresher from previous courses. You are not required to know the derivation of the distribution and you will not have questions about the Poisson distribution in the exam. You need to understand how the Poisson process is used to model extremes and the assumptions you make when you apply this model. You might have questions about these points in the exam.

## Deriving the Distribution

Let's start with the pmf of a Binomial random variable.

$$p_X(x) = P[X = x|n, p] = \binom{n}{x} p^x (1-p)^{n-x}$$

Now, let $\lambda = np$ be the expected number of successes in a series of $n$ Bernoulli trials with probability $p$.

$$p_X(x) = P[X = x|n, p] = \binom{n}{x} (\lambda/n)^x (1-\lambda/n)^{n-x}$$

$$= \frac{n(n-1)(n-2)...(n-x+1)}{n^x x!} \lambda^x (1-\lambda/n)^n (1-\lambda/n)^{-x}$$

If the number of trials is large, $n \to \infty$, and $x$ and $p$ are fixed and finite:

$\lim_{n\to\infty} \frac{n(n-1)(n-2)...(n-x+1)}{n^x} = 1$

$\lim_{n\to\infty}(1-\lambda/n)^{-x} = 1$

$\lim_{n\to\infty}(1-\lambda/n)^{n} = e^{-\lambda}$

So, we can simplify it as

$$p_X(x) = P[X = x|n, p] = \frac{\lambda^x\, e^{-\lambda}}{x!} \qquad for\, x = 0, 1, 2, \dots\ and\, \lambda > 0$$

$$p_X(x) = P[X = x|p] = 0 \qquad otherwise$$

which is the **Poisson distribution**! Therefore, if $n \to \infty$ and $p$ is small so $\lambda = np$, the random variable $X$ is Poisson distributed with parameter $\lambda$.

> **Binomial vs. Poisson distribution**
>
> Although both the Binomial and Poisson distribution are discrete distributions which measure the number of successes within a certain time block, Binomial is based on discrete events, while the Poisson is based on continuous events. That is, in Poisson distribution $n \to \infty$ and $p$ is very small, so you have an infinite number of trials with infinitesimal chance of success. In the Binomial distribution, the number of trials $n$ is finite.

# Poisson distribution and Poisson process

If we integrate the pmf of the Poisson distribution, we can obtain its cdf defined as

$$F_X(x) = \sum_{k=0}^{x} \frac{\lambda^x e^{-\lambda}}{x!} \qquad for\, x = 0, 1, 2, \dots\ and\, \lambda > 0$$

Since Poisson distribution is derived by imposing that $E[X] = \lambda$, it is not a surprising property. However, the Poisson distribution also satisfied $E[X] = Var[X] = \lambda$.

**Poisson-distributed variable**

The conditions for a random variable to follow a Poisson distribution are:

1. A series of Bernoulli trials is made with two possible outcomes: success or failure.
2. There is a **small** constant probability of success, $p$.
3. There is a **very large** number of trials, $n \to \infty$.
4. The outcomes of the trials are independent.
5. The random variable X is the total number of successes in $n$ trials and the order of the successes is irrelevant.

It is also referred as the Poisson approximation to the Binomial.

A Poisson process is a particular form of a random process, where the average time between events is known, but the exact timing of events is random.

**Poisson process criteria**

The governing criteria for a Poisson process are:

1. It is a Bernoulli process, so:

   a. There are only two possible outcomes which are mutually exclusive and collectively exhaustive: success vs. failure.

   b. There is a constant probability of success, $p$.

   c. Each trial is independent. That is, the result of the next trial does not depend on the previous one.
2. It is based on a continuous process (infinite number of trials, $n$)
3. The average number of events ($\lambda$) or average rate ($\nu = \lambda/t$, where $t$ is the length of the time block) is constant.
4. Two events cannot occur at the same time.

Under this Poisson process conditions, the Poisson distribution models the probability of a observing $x$ number of events in a fixed time block if these events occur with a known constant rate ($\nu$) and independently of the time since the last event.

# Poisson process to model extremes sampled with POT

Let’s go back to our example on the design wave for a breakwater. Each hour within my register (each trial in $n \to \infty$), I have a value of $H_s$ which can be over (success) or below (failure) my threshold. Since I am interested in extreme observations (the tail of the distribution), the probability of being above my threshold ($p$) is very small. I am interested in the yearly probability of failure ($p_{f,y}$), so my time block is one year. If I count the number of excesses over my threshold (number of successes) each year, **that is a Poisson process**!

**And what is interesting about this?**

It means that if I do my POT to sample extremes and I check if the number of extremes per year follow a Poisson distribution, I am ensuring that the sampled events are independent and, thus, my threshold and declustering time are appropriate.

Almost all the techniques to formally select the threshold and declustering time for POT are based on the assumption that the sampled extremes should follow a Poisson distribution. You will see some of these techniques in further detail in the following sections.

# Threshold & Declustering

## Contents


- Let's practice!


In the previous chapter, we saw the Peak Over Threshold (POT) technique to sample extremes and we applied it to our time series. We used a threshold $th = 2.5m$ and a declustering time $dl = 48h$, obtaining the figure below. However, no justification was given for those parameters. In this section, we will start giving insight on how to select $th$ and $dl$, which is inherently a verification and validation process for our chosen distribution.

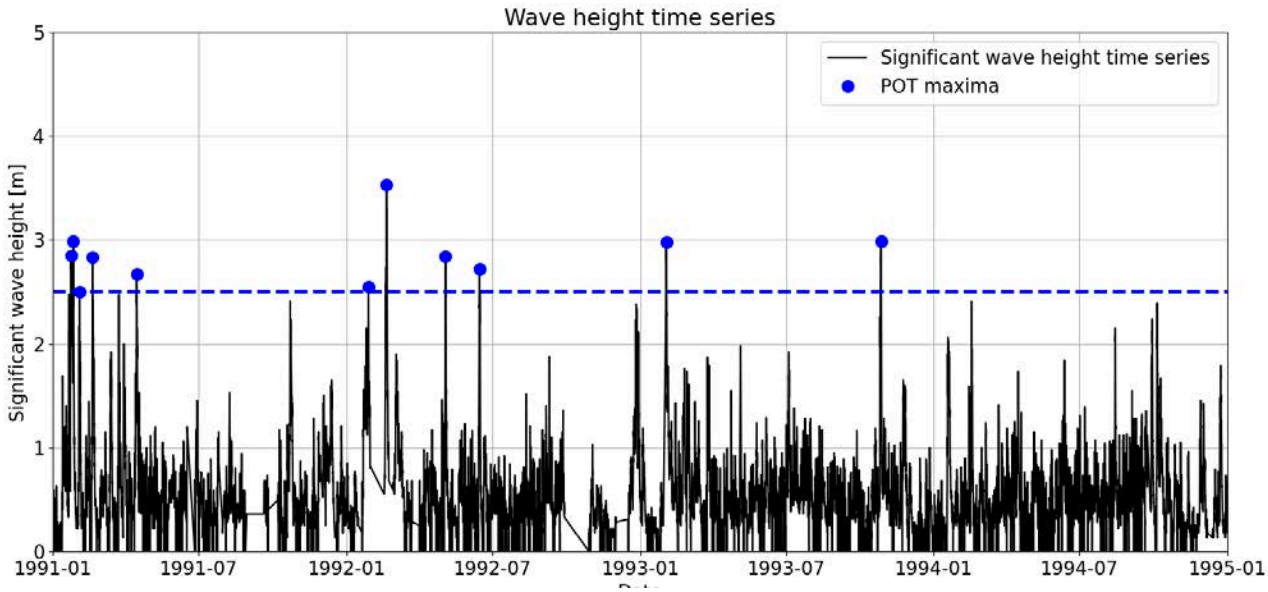



*Fig. 7.16* Application of POT to $H_s$ time series with $th = 2.5m$ and $dl = 48h$.

We also discussed previously that extreme observations tend to cluster in time and, thus, we need to ensure that only one extreme observation is sampled within each cluster to guarantee our assumption of *iid* observations. Thus, threshold and declustering time should be selected accounting for it. In addition, the concept of Poisson process and its relationship with EVA was introduced. It was concluded that by ensuring that the number of excesses per year followed a Poisson distribution, the sampled extremes were *iid*.

There are several techniques in the literature to support the decision-making process of selecting $th$ and $dl$ for POT. Here, we will start with the basic one (application of properties of Poisson distribution and hypothesis testing to check whether the number of excesses per

year follows a Poisson distribution) which is the underlying assumption below the more complex techniques that we will see in subsequent sections.

We already applied POT with $th = 2.5m$ and $dl = 48h$ to our example dataset. Let's check whether those parameters are appropriate or we should change them. To do so, we are going to check if the number of excesses per year follows a Poisson distribution.

First step is to calculate the number of excesses we have per year and, with it, the empirical pmf and the empirical cdf. Remember that the pmf gives us $P[X = x]$ and the cdf, $P[X \leq x]$.

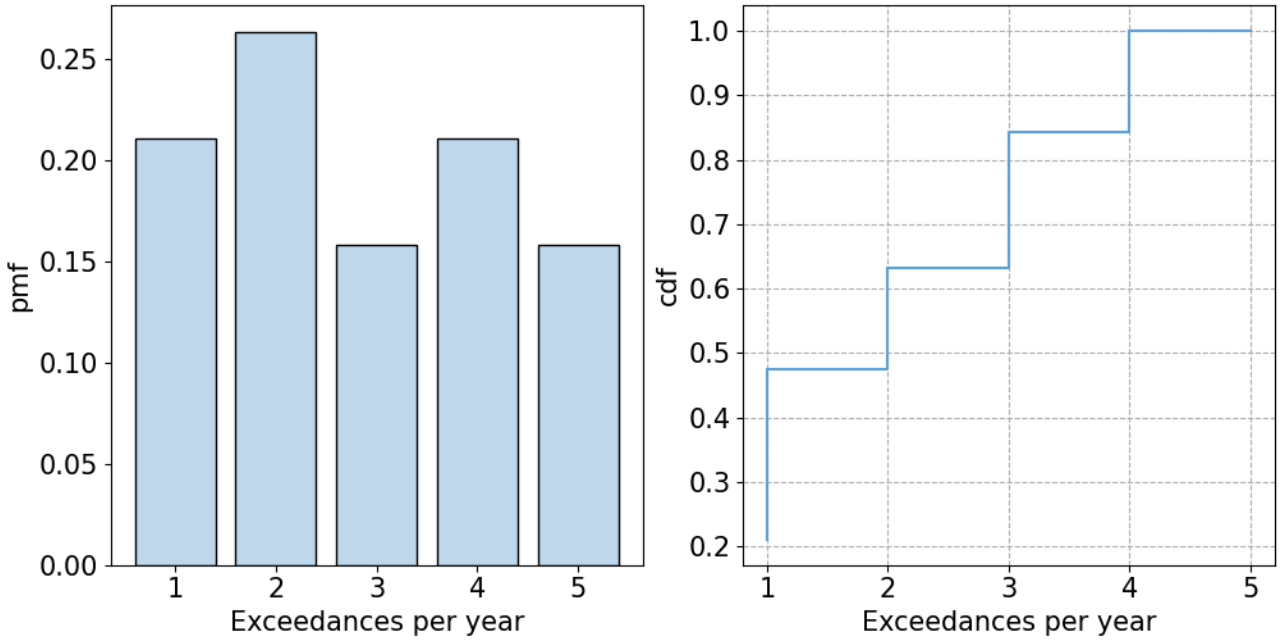



*Fig. 7.17* Empirical pmf and cdf for number of excesses per year.

Now, we can fit a Poisson distribution to that empirical distribution and check whether the Poisson distribution is a reasonable model for our number of excesses per year.

We can fit such distribution using Moments method, which consists of estimating the parameters of the distribution based on the moments calculated from the data (mean, standard deviation…). Applying the properties of the Poisson distribution, $E[X] = Var[X] = \lambda$, being $\lambda$ the distribution parameter. Based on that, we can conclude:

- Using my observations, $E[X] = 2.84$ and $Var[X] = 1.92$. Thus, $E[X] = Var[X]$ is not true for our observations, but we can assume $E[X] \approx Var[X]$.
- We can assume a fitted Poisson with $\lambda = E[X] = 2.84$ for the subsequent analysis.

Once we have fitted the Poisson distribution, we can visually check the fit, as shown below. The figure on the left shows the supperposition of the empirical and fitted cdf. The figure on

the right shows the empirical and estimated probabilities for number of excesses = 1, 2, … 5, namely the PP-plot. The fitting seems reasonable.

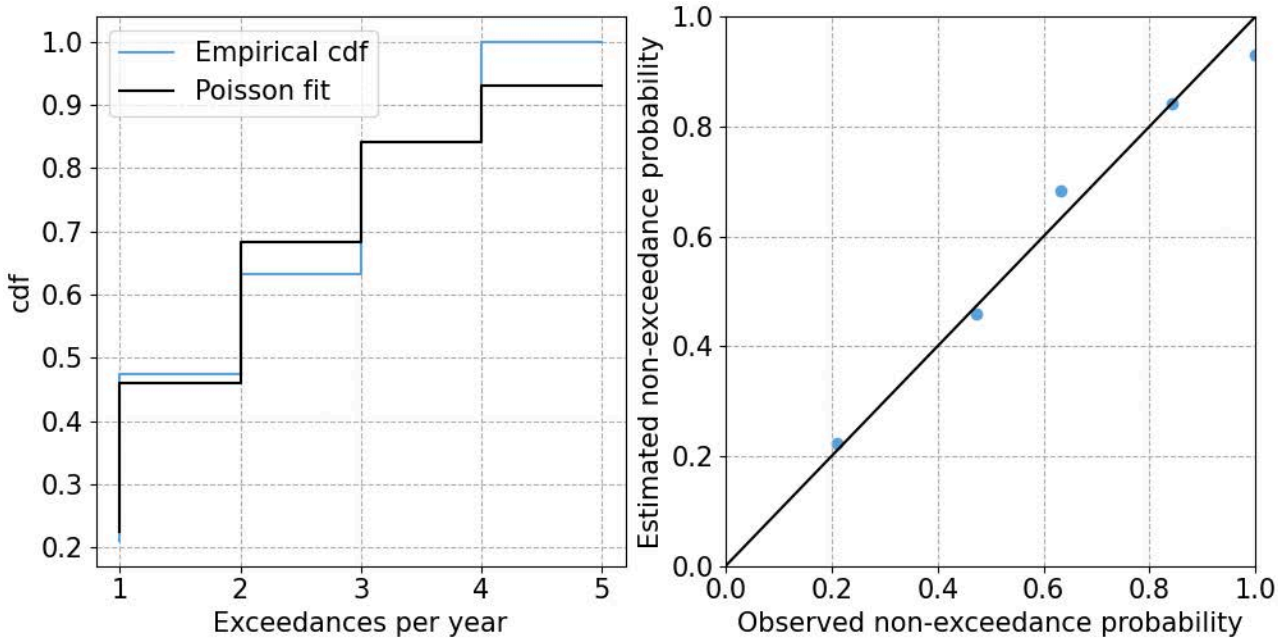



*Fig. 7.18* Empirical pmf and cdf for number of excesses per year.

In order to further support our decision, we can perform a goodness-of-fit hypothesis test for discrete distributions, such as $\chi^2$ test.

# Let's practice!

A scientist is analyzing the extreme precipitation rate events in a city and decided to apply POT to sample the extreme events. The scientist has chosen a threshold $th = 100\, mm/h$ and declustering time $dl = 2$ min. The following figure shows part of the timeseries together with the sampled extremes and the threshold.

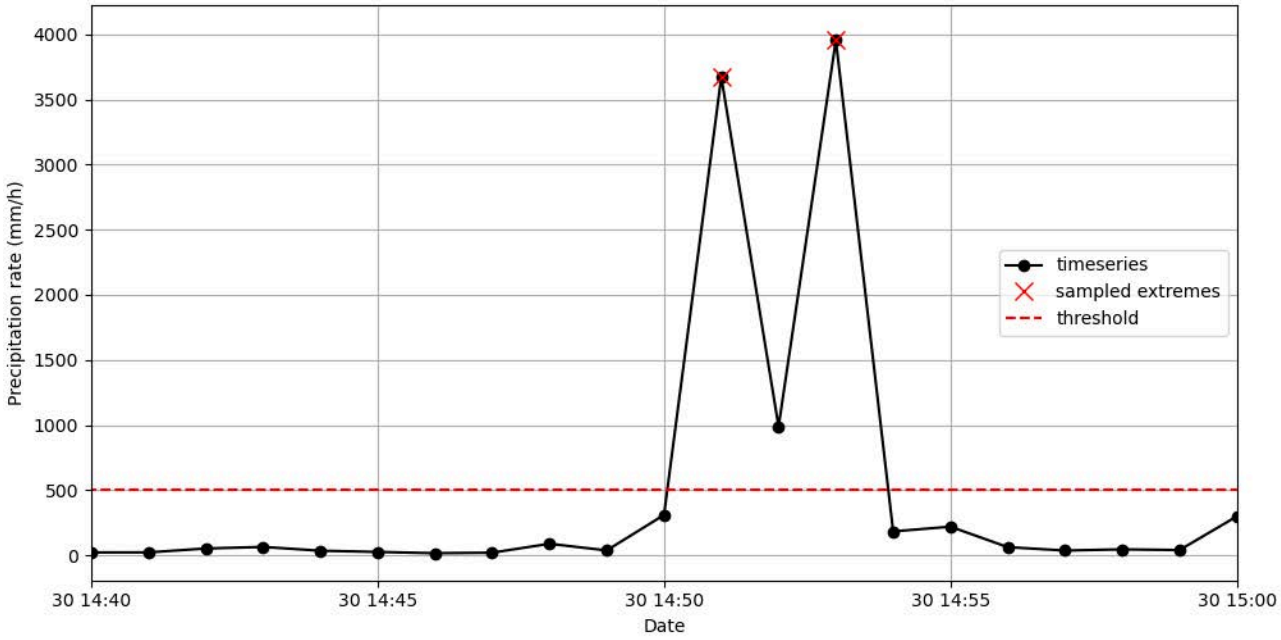



*Fig. 7.19* Sampled extremes in the precipitation rate timeseries.

Do you think that the selected POT parameters ($th$ and $dl$) are adequate?

**Answer**

The declustering time is not long enough to prevent POT to sample extremes belonging to the same storm, according to the above picture. Two extremes from the same storm event are sampled and, since they are generated by the same drivers, cannot be considered independent. Moreover, if we apply the knowledge on the physical phenomenon that the scientist is studying (a storm), we know that extreme storms are longer than a few minutes.

The scientist performs again the analysis with a different set of parameters and obtains a mean value of events per month of $E[X] = 3.33$ and a variance of $Var[X] = 3.24$. Without further information, do you think that the select POT parameters are adequate?

**Answer**

Applying the properties of the Poisson distribution, $E[X] = Var[X] = \lambda$, being $\lambda$ the distribution parameter. Here $E[X] = 3.33 \approx Var[X] = 3.24$. Therefore, the number of sampled events per time block (here, one month) seems to come from a Poisson distribution. This indicates that the sampled events are independent and, thus, the parameters of the POT analysis are adequate.

# GPD: Introduction

## Contents


- Definition and parameters of GPD distribution


After sampling the extreme observations in our time series using POT, we have a sequence of independent and identically distributed excesses. Now, we want to model their statistical behavior.

Let $F_{th}$ be the distribution function of the excesses over a threshold, $th$,

$$F_{th} = P[X - th \leq x | X > th], \quad x \geq 0$$

Let the excesses be $Y = X - th$ for $X > th$ for a sequence of $n$ independent random variables $X = X_1, \ldots, X_n$.

We already saw that given:

- A sequence of independent random variables $X = X_1, \ldots, X_n$,
- The maxima of the sequence $M_n = max(X_1, \ldots, X_n)$, where $n$ is the number of observations in a given block,
- $M_n$ are independent and identically distributed.

For large $n$ the cumulative distribution function of those maxima tend to the Generalized Extreme Value ($G(x)$, GEV) family of distributions, regardless the distribution of $X$. If that is true, **the distribution of the excesses can be approximated by a Generalized Pareto distribution** [1].

# Definition and parameters of GPD distribution

The Generalized Pareto distribution is defined as

$$H(y) = \begin{cases} 1 - \left(1 + \frac{\xi y}{\sigma_{th}}\right)^{-1/\xi} & for\ \xi \neq 0 \\ 1 - exp\left(-\frac{y}{\sigma_{th}}\right) & for\ \xi = 0 \end{cases}$$

where $y \geq 0$ if $\xi \geq 0$, and $0 \leq y \leq -\frac{\sigma_{th}}{\xi}$ if $\xi < 0$.

When applied to model the excesses over a threshold, we are calculating conditional probabilities to $X > th$. This distribution function can be computed as function of the random variable $X$ and the threshold $th$ as

$$P[X < x | X > th] = \begin{cases} 1 - \left(1 + \frac{\xi(x-th)}{\sigma_{th}}\right)^{-1/\xi} & for\ \xi \neq 0 \\ 1 - exp\left(-\frac{x-th}{\sigma_{th}}\right) & for\ \xi = 0 \end{cases}$$

If our previous assumption holds, the parameters of the GPD can be determined based on those of the GEV: the shape parameter ($\xi$) is the same for both distributions, and $\sigma_{th}$ is function of the location and scale parameters of the GEV as

$$\sigma_{th} = \sigma + \xi(th - \mu)$$

Note that $th$ acts like a location parameter for the GPD distribution. In the plot below, this effect is illustrated.

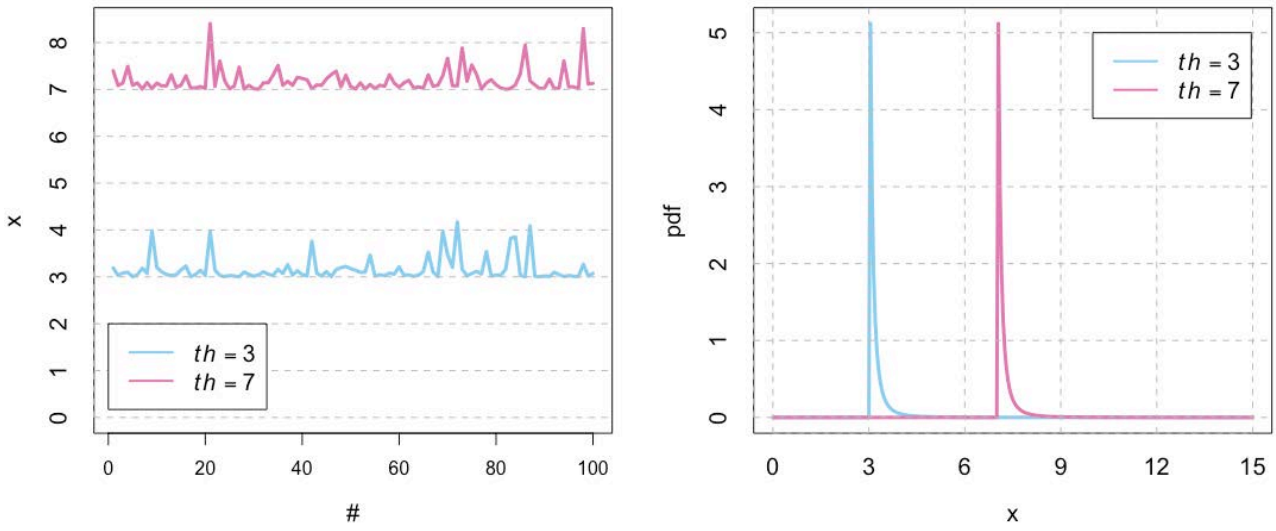



*Fig. 7.20* Influence of shape parameter on pareto distribution.

Similarly to the GEV, $\xi$ determines the behavior of the tail of the distribution. If $\xi < 0$, the distribution presents an upper bound at $th - \sigma_{th}/\xi$. If $\xi > 0$, the distribution has a heavy upper tail that behaves like a power function of the exponent $-1/\xi$. If both $\xi = 0$ and $th = 0$, GPD reduces to the Exponential distribution and if $\xi = -1$, to the uniform distribution. We can see it in the following plot.

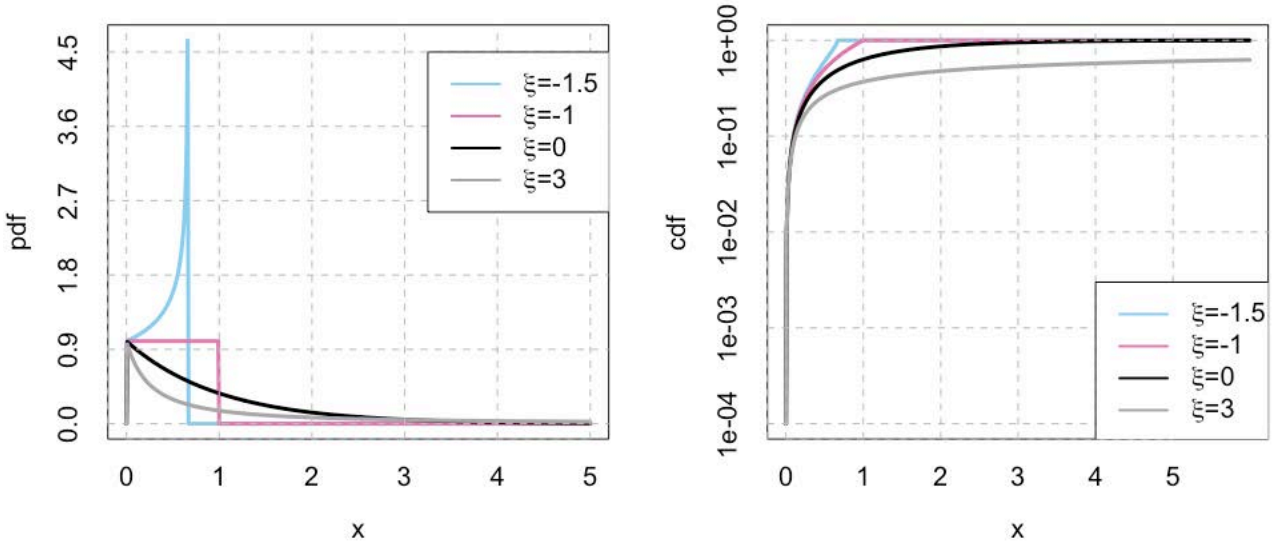



*Fig. 7.21* Influence of shape parameter on pareto distribution.

In the above plot, we have the cdf of four GPDs with a $th = 0$, $\sigma_{th} = 1$ and $\xi = -1.5,\ -1,\ 0\ and\ 3$. We can see the cdfs of the GPD with $\xi = -1.5\ and\ -1$ have a clear upper bound in $x = 0.67\ and\ 1$, respectively. If we apply the previous formula $th - \sigma_{th}/\xi = 0 - 1/(-1.5) = 0.67$ and $0 - 1/(-1) = 1$, which fit our visual observation.

Regarding the scale parameter $\sigma_{th}$, it presents a behavior equivalent to that observed on $\sigma$ for the GEV distribution. The higher $\sigma_{th}$, the wider the pdf of the distribution.

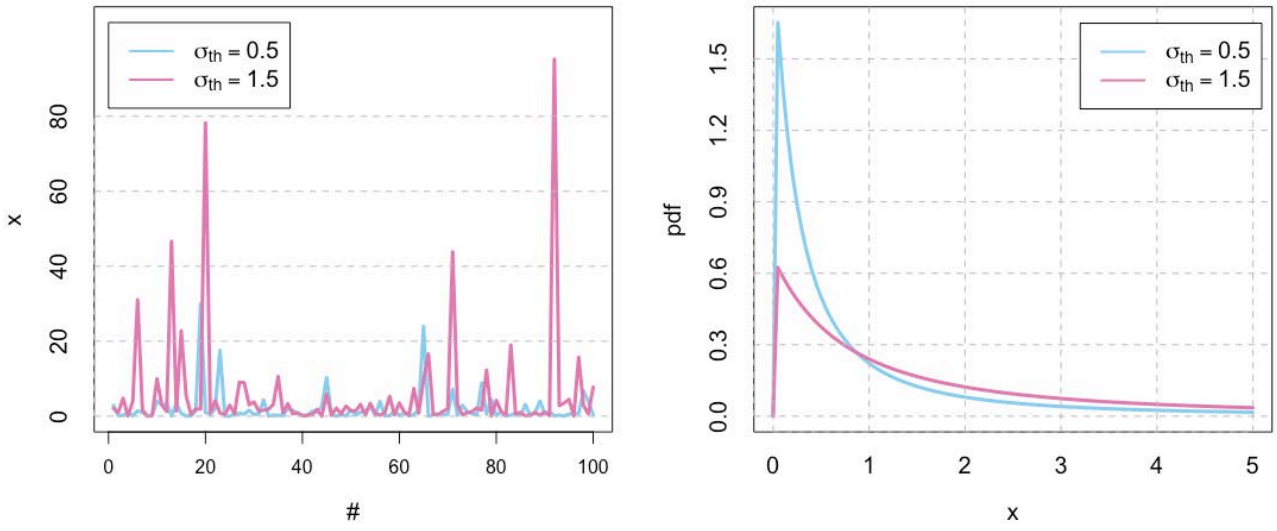



*Fig. 7.22* Influence of scale parameter on pareto distribution.

J. Pickands. Statistical inference using extreme order statistics. *the Annals of Statistics*, 119-131, 1975

# GPD: m Return Levels

## Contents


- From $m$-observations to $N$-years
- Computing $N$-years return levels
- Let's apply it!
- Let's practice!


Now you know a bit more about the GPD distribution. Here, we will see some considerations to keep in mind when applying it to model extremes.

> **MUDE exam information**
>
> In this section, the derivation to go from m-observations return levels to N-years return levels is provided based on the Poisson model for extremes. You are not required to know the derivation for the exam, it is shown as background information so you know where it is coming from. You need to know how to compute return levels from a Generalized Pareto distribution, as shown in the section "Let's apply it".

We already saw that we are interested in evaluating the return level of the studied variable: the $N$-year return level $x_N$, which is expected to be exceeded once every $N$ years. **How can we do that?**

Imagine that we have already fitted a GPD with $\xi \neq 0$, so the exceedance probabilities can be computed as

$$P[X > x | X > th] = \left(1 + \frac{\xi(x-th)}{\sigma_{th}}\right)^{-1/\xi}$$

GPD is a conditional distribution ($X > th$), so we need to account for the probability of being above the threshold and, thus, observing an excess, $P[X > th] = \zeta_{th}$. Therefore, writing it in terms of exceedance probabilities

$$P[X > x] = P[X > th]\, P[X > x | X > th] = \zeta_{th} \left(1 + \frac{\xi(x-th)}{\sigma_{th}}\right)^{-1/\xi}$$

Note that $\zeta_{th}$ depends on the selected threshold.

Then, the return level $x_m$ exceeded in average every $m$ observations is computed as

$$1/m = \zeta_{th}\left[1 + \xi\frac{(x_m - th)}{\sigma_{th}}\right]^{-1/\xi}$$

which leads to

$$x_m = th + \frac{\sigma_{th}}{\xi}\left[(m\zeta_{th})^{\xi} - 1\right]$$

which is valid for large values of $m$ to ensure $x_m > th$.

For $\xi = 0$ the same procedure can be performed, obtaining

$$x_m = th + \sigma_{th} log(m\zeta_{th})$$

**$x_m$ is called the $m$-observations return level.**

# From $m$-observations to $N$-years

In Engineering and Geosciences, we typically work with $N$-year return levels (what we defined as the Return Period). Then, we need to account for the number of observations per year, $n_y$, being $m = N \times n_y$. Applying this in the previously derived equations, we can compute the $N$-year return level as

$$x_N = \begin{cases} th + \frac{\sigma_{th}}{\xi}[(Nn_y\zeta_{th})^{\xi} - 1] & for\ \xi \neq 0 \\ th + \sigma_{th} log(Nn_y\zeta_{th}) & for\ \xi = 0 \end{cases}$$

Therefore, if we want to calculate the return level associated with a return period of 10 years, $m = 10 \times n_y$.

**But how can we compute the probability of being above the threshold and, thus, observing an excess, $P[X > th] = \zeta_{th}$?**

You have already seen how the Poisson distribution helps us modelling the number of excesses over a threshold when performing POT. Here, we will use this distribution to compute the probability of being above the threshold $P[X > th] = \zeta_{th}$ and, thus, the return levels based on a GPD distribution.

# Computing $N$-years return levels

In the previous sections, we reached the following expression to calculate the return levels

$$x_N = \begin{cases} th + \frac{\sigma_{th}}{\xi}[(N n_y \zeta_{th})^{\xi} - 1] & for\ \xi \neq 0 \\ th + \sigma_{th} log(N n_y \zeta_{th}) & for\ \xi = 0 \end{cases}$$

where $x_N$ is the $N$-years return level, $th$ the threshold, $\sigma_{th}$ and $\xi$ are the fitted parameters of the GPD, $n_y$ is the average number of observations per year and $\zeta_{th}$ is the probability of observing an exceedance.

We saw that we can model the number of exceedances per year using a Poisson distribution and that this distribution is characterized by a parameter $\lambda$ equal to the mean and standard deviation of the random variable. Thus, we can calculate $\zeta_{th}$ [1] as

$$\hat{\zeta}_{th} = \frac{\hat{\lambda}}{n_y}$$

where $\hat{\lambda}$ can be estimated as the average number of exceedances per year as

$$\hat{\lambda} = \frac{n_{th}}{M}$$

being $n_{th}$ the total number of sampled exceedances and $M$ the number of years in the database.

Updating the first equation using the Poisson distribution, we obtain

$$x_N = \begin{cases} th + \frac{\sigma_{th}}{\xi}[(\lambda N)^{\xi} - 1] & for\ \xi \neq 0 \\ th + \sigma_{th} log(\lambda N) & for\ \xi = 0 \end{cases}$$

or

$$x_N = \begin{cases} th + \frac{\sigma_{th}}{\xi}[(\frac{n_{th}}{M} N)^{\xi} - 1] & for\ \xi \neq 0 \\ th + \sigma_{th} log(\frac{n_{th}}{M} N) & for\ \xi = 0 \end{cases}$$

# Let's apply it!

Moving back to our example, we were interested in estimating $H_s$ with a $RT = 100\ years$. We had 20 years of hourly recordings in our buoy and, using $th = 2.5m$ and $dt = 48h$, we sampled 54 excesses.

First, we fit the parameters of our distribution using the sampled **excesses** (not the actual value of the random variable), $\sigma_{th}$ and $\xi$ using Maximum Loglikelihood Estimator. We obtain $\sigma_{th} = 0.69$ and $\xi = -0.27$.

Once we have fitted the parameters, we check the goodness of fit of such fitting. Below, the QQ-plot and the exceedance probability plot in logarithmic scale are presented.

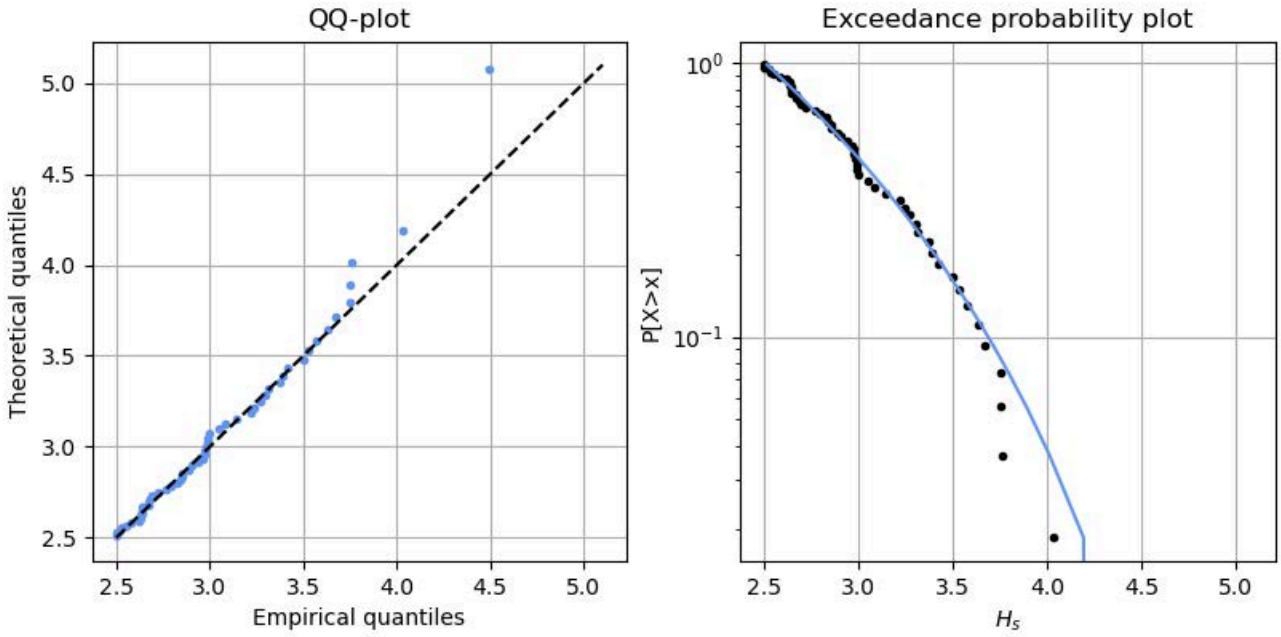



*Fig. 7.23* Goodness of fit for Poisson distribution.

The fitting is reasonable, so we can use the fitted distribution to determine the return level associated to a $RT = 100\,years$. We have $M = 20\,years$ of observations and $n_{th} = 54\,events$. Thus, $\hat{\lambda} = 54/20 = 2.7$. Applying the previous expression for $\xi \neq 0$

$$x_{100\ years} = th + \frac{\sigma_{th}}{\xi}[(\lambda N)^{\xi} - 1] = 2.5 + \frac{0.69}{-0.27}[(2.7 \times 100)^{-0.27} - 1] = 4.49\,m$$

Thus, the design wave height with a $RT = 100\,years$ is 4.49m based on our Extreme Value Analysis. **Congratulations! You have performed your first Extreme Value Analysis using POT and GPD!**

# Let's practice!

A colleague is performing EVA to determine the maximum design wind loading in a building. The engineer retrieved a timeseries of wind speeds of 17 years and used POT to sample the extremes within the timeseries. The engineer obtained 68 extreme excesses.

What is the average number of exceedances per year?

**Answer**

The number of years is $M = 17$ and the total number of exceedances is $n_{th} = 68$. Thus, we can compute the average number of exceedances per year as

$$\hat{\lambda} = \frac{n_{th}}{M} = 68/17 = 4$$

Our colleague has fitted a GPD on the obtained excesses using a threshold $th = 40$km/h. The fitted parameters are $\sigma_{th} = 4.1$ and $\xi = 0.3$. What is the 25 years-return level?

**Answer**

$$x_{25\ years} = th + \frac{\sigma_{th}}{\xi}[(\lambda N)^{\xi} - 1] = 40 + \frac{4.1}{0.3}[(4 \times 25)^{0.3} - 1] = 80.7\ km/h$$

Finally, our colleague has decided to apply monthly maxima and, thus, fit a GEV to the selected observations. Since 17 years of data are available, 204 extreme observations are obtained. The fitted parameters of the GEV are $\mu = 37$, $\sigma = 5$, and $\xi = 0.3$.

**Answer**

Considering that the GEV has been fitted using monthly extremes, this distribution will provide the monthly non-exceedance probabilities. Then, we need to calculate what is the monthly non-exceedance probability that corresponds to a return period of 25 years. Since we are working with the monthly extremes, we know that $\lambda = 12$, as we have 12 observations per year. Thus, the monthly exceedance probability ($p_{f,m}$) can be computed as

$$p_{f,y} = \lambda p_{f,m} \rightarrow p_{f,m} = \frac{p_{f,y}}{\lambda} = \frac{1/25}{12} = 0.0033$$

After that, we can apply the CDF of the GEV distribution as

$$x_{25\ years} = \mu - \frac{\sigma}{\xi}[1 - \{-log(1 - p_{f,m})\}^{-\xi}]$$

$$x_{25\ years} = 37 - \frac{5}{0.3}[1 - \{-log(1 - 0.0033)\}^{-0.3}] = 112.5\ km/h$$

---

Note that in mathematics, $\hat{\lambda}$ denotes the estimation of $\lambda$.

# Return Period & Design Life

## Contents


- Deriving the probability of failure along the design life
- Let's practice!


In the previous sections, we have covered the process of how to perform EVA using POT sampling technique and GPD distribution. Also, Poisson distribution function and its relationship with EVA was introduced. Here, $RT$ will be again extended to cover the concepts related to the design life in the context of POT sampling technique and Poisson process.

As previously introduced, there are four key concepts to parameterize safety in the design phase:

- Probability of failure in one year ($p_{f,y}$).
- Return period ($RT$), usually given in years. According to the previous definition, $RT = 1/p_{f,y}$.
- Design life ($DL$): lifetime of the infrastructure.
- Probability of failure in the design life of the structure ($p_{f,DL}$).

We saw that the number of excesses per time block (typically a year) follows a Poisson distribution and can be used together with the GPD distribution to determine the yearly exceedance probabilities. However, **how can we obtain the exceedance probability along the design life?**

**MUDE exam information**

In this section, the derivation of the formula which relates the above mentioned parameters is presented based on the Poisson model. You are not required to know the derivation, it is shown as background information so you know the assumptions which are done in the process. You need to understand the concepts of yearly probability of failure, return period, design life and the probability of failure in the design life. Also, you need to be able to apply the derived equation.

# Deriving the probability of failure along the design life

We already saw that extremes could be assimilated as a Bernoulli process: for each year (trial), we check if the observed value exceeds our design value (success) or not (failure). Thus, the number of exceedances over a design value (successes) in an infinite number of years (trials) will follow the Poisson distribution if each trial is independent and the probability of success (exceeding the threshold) is very small.

Let's calculate the probability of observing an event $z_0$ higher than the design value $z_q$ at least once in $DL$ years ($p_{f,DL}$).

First, we calculate the probability of not failing along $DL$ applying the Poisson distribution. Remember that $\lambda = np$, being $n$ the number of trials and $p$ the probability of success.

$$p_X(0) = P[X = 0|DL, p_{f,y}] = \frac{\lambda^0 \, e^{-\lambda}}{0!} = e^{-DL \times p_{f,y}}$$

The probability of failing at least once in $DL$ can be computed as $1 - p_X(0)$ (1 - no failure), so

$$p_{f,DL} = 1 - p_X(0) = 1 - e^{-DL \times p_{f,y}}$$

We defined $RT = 1/p_{f,y}$, so

$$p_{f,DL} = 1 - e^{-DL/RT} = 1 - e^{-DL/RT}$$

Rewriting it in terms of $RT$, we obtain

$$RT = \frac{-DL}{ln(1-p_{f,DL})}$$

And now you can compute the design $RT$ based on the $DL$ and $p_{f,DL}$ recommended in design guidelines!

# Let's practice!

An scientist is designing a *thingamajig*. According to the recommendations, any *thingamajig* should be designed for a lifespan of 25 years with a probability of failure along the design life of $p_{f,DL} = 0.05$.

What is the design return period using the Binomial model?

**Answer**

Applying the expression for the return period derived applying a Binomial distribution

$$RT = 1/p_{f,y} = \frac{1}{1-(1-p_{f,DL})^{1/DL}} = \frac{1}{1-(1-0.05)^{1/25}} \approx 487.7 \text{ years}$$

What is the design return period using the Poisson model?

**Answer**

Applying the expression for the return period derived applying a Poisson distribution

$$RT = 1/p_{f,y} = \frac{-DL}{ln(1-p_{f,DL})} = \frac{-25}{ln(1-0.05)} \approx 487.4 \text{ years}$$

Is there a significant difference between both approaches?

**Answer**

In terms of the computed return periods ($RT$), there is not a significant difference. Although Poisson model is formally more correct since it assumes an infinite number of trials, both models can be applied.

# Supplementary Material

In the following sections, a refresher on the concept of Bernoulli process and Binomial distribution is presented; this is prerequisite material that will aid in understanding extreme value analysis concepts. In addition, some explanatory videos from previous courses are given in case you would like to learn about the material in this chapter in a slightly different way.

# Bernoulli and Binomial

## Contents


- Bernoulli trials and Bernoulli process: do I need to study for my probability exam?
- Binomial distribution


> Bernoulli processes and the Binomial distribution: why is this relevant?

Extreme value analysis and the study of extremes has a foundational basis in the theory of Bernoulli processes and the Bernoulli and Binomial distributions.

**MUDE exam information**

In this section, a refresher about Bernoulli processes and Binomial distribution is provided. You will not have questions in the exam about them and you don't need to know the presented derivations. You need to know how Bernoulli processes and Binomial distribution are applied to model extremes and the assumptions which are made for that.

# Bernoulli trials and Bernoulli process: do I need to study for my probability exam?

One student wants to know if she will pass her favourite course (of course, probability) without studying. Thus, she has the simplest possible experiment: one trial with two possible outcomes, pass (success!) and not pass (usually called failure). Note that they are mutually exclusive (if you pass, you haven't failed) and collectively exhaustive (there's no third possible outcome from the experiment). The results from such a trial are a two-sided Bernoulli random variable and, thus, the distribution of probabilities of the two outcomes is a Bernoulli distribution. The **Bernoulli distribution** is a discrete distribution function whose probability mass function (pmf) is given by

$$p_X(x) = P[X = x|p] = p^x(1-p)^{1-x} \qquad for\, x = 0\, or\, 1\, and\, p \in [0,1]$$

$$p_X(x) = P[X = x|p] = 0 \qquad otherwise$$

where $p$ is the probability of success ($x = 1$), and $x = 0\ and\ 1$ are the possible outcomes. This is, $p(success) = p$.

Imagine that the student is not studying and wants to leave it to chance. She would have to take the course several times (several trials) and since she is not studying between trials, each trial would be independent (she is not learning from previous experiences). This will constitute what it's called a **Bernoulli process**.

> **Bernoulli process criteria**
>
> The governing criteria for a Bernoulli process are:
>
> 1. There are only two possible outcomes which are mutually exclusive and collectively exhaustive: success vs. failure.
> 2. There is a constant probability of success, $p$.
> 3. Each trial is independent. That is, the result of the next trial does not depend on the previous one.

# Binomial distribution

The student loved the probability course and she doesn't mind retaking it, but since she doesn't want to delay her graduation, she needs to pass the course in the current year. In the course, there are three exams and needs to pass at least two of them. Therefore, she is interested in determining the probability of passing two exams out of the three without studying.

The student knows that needs to know the probability of success, $p$, since she remembers from the probability lectures that it is a Bernoulli process. Thus, she performs a survey to other students from previous courses. 50 students admitted not studying for a probability exam and 2 of them passed it. Then, she determines the success probability as $p = 2/50 = 0.04$.

**And now?**

Let's go step by step. Considering that the exam's outcomes (failure/success) are independent, the probability of having two successes and one failure is $p^2(1-p)$ (remember that events are independent!). However, there are three different sequences which may lead to that situation: passing the first two exams, passing the first and the third

exam or passing the last two exams. Hence, the probability of two successes and one failure in three trials can be obtained from the addition rule[1] as follows:

$$p_Y(2) = P[Y = 2|3, p] = \binom{3}{2} p^2(1-p)^{3-2} = \frac{3!}{2!(3-2)!} p^2(1-p) = 3p^2(1-p)$$

which is the pmf of the **Binomial distribution**. Thus, we went from the **Bernoulli** distribution to the **Binomial** distribution. The pmf of the Binomial distribution is then defined as

$$p_Y(y) = P[Y = y|n, p] = \binom{n}{x} p^x(1-p)^{n-x} \qquad for\ y = 0, 1, \ldots, n;\ p \in [0, 1]$$

$$p_Y(y) = P[Y = y|n, p] = 0 \qquad otherwise$$

where

$$\binom{n}{y} = \frac{n!}{x!(n-y)!}$$

which is the total number of possible combinations when selecting y successes from n trials. Thus, in our example, the random variable Y represents the number of successes in n trials and follows a Binomial distribution with parameters $p$ (probability of success) and $n$ (number of trials). The cumulative distribution function (cdf) of a Binomial distribution is given by

$$F_Y(y) = \sum_{k=0}^{x} \binom{n}{y} p^k(1-p)^{n-k}$$

Note that the maximum value of the cdf is reached for $Y = n$.

> **Binomial-distributed variable**
>
> The conditions for a random variable to follow a Binomial distribution are:
>
> 1. A series of Bernoulli trials is made with two possible outcomes: success or failure.
> 2. There is a constant probability of success, $p$.
> 3. There is a fixed number of trials, $n$.
> 4. The outcomes of the trials are independent.
> 5. The random variable X is the total number of successes in $n$ trials and the order of the successes is irrelevant.

Finally, **let's answer the student questions**. Applying the Binomial distribution:
$$p_Y(y) = P[Y = 2|3, 0.04] = \binom{3}{2} p^2(1-p)^{3-2} \approx 0.005$$

Sadly, there is a very low chance that the student can make it without studying! However, now she knows Bernoulli processes and Binomial distribution and that for sure increases the

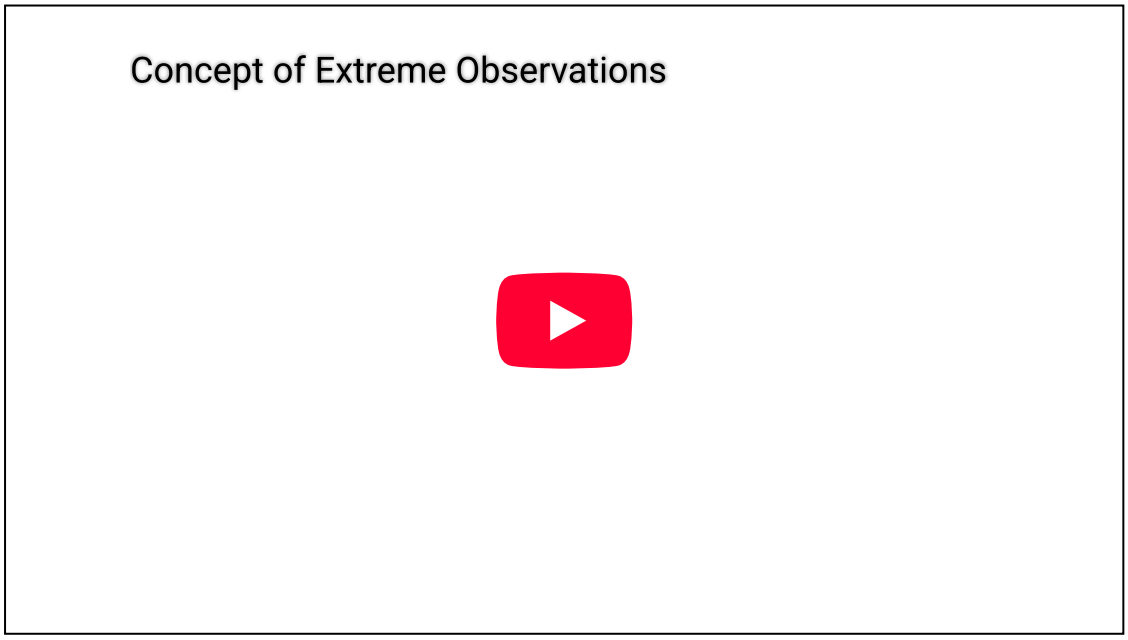
chances of passing!

---

> The addition rule is a basic probability axiom which states that for any sequence of k mutually exclusive events $A_1, A_2, \ldots, A_k$ is true that the probability of all of them together is the same as the sum of the individual probabilities.
> $P[A_1 + A_2 + \ldots + A_k] = P[A_1] + P[A_2] + \ldots + P[A_k]$

# Supplementary material: videos

On this page videos related to extreme value analysis are collected together. Note that there may not be a perfect overlap between the content in the chapters of the book and the videos; use the video titles to find relevant content. In case of doubt, always refer to the written contents of the book.



Return Period

*The following videos were used in another course and give more depth on some topics, but are included here because there is a lot of overlap with the EVA content of MUDE.*

**1. Introduction to EVA and overview**

**2. Design requirements**: Bernoulli trial and Binomial distribution

**3. Block Maxima and Generalized Extreme Value distribution**:

**4. Peak Over Threshold and Generalized Pareto distribution**: Introduction

**5. Peak Over Threshold and Generalized Pareto distribution**: Poisson process

**6. Peak Over Threshold and Generalized Pareto distribution**: Threshold and declustering time selection.

Only the first three minutes of this video are within the scope of this course.

# Risk Analysis

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

Throughout this book we have focused on a variety of deterministic and probabilistic topics, which probably appeared to be completely unrelated. However, in engineering practice we often need to combine deterministic and probabilistic approaches to design and assess the projects we work on and Risk Analysis concepts are an important way to facilitate this. In particular, concepts in this chapter are focused not so much on evaluating the behavior of a particular system, but rather *evaluating the risk associated with various outcomes* and, most importantly, providing *a framework with which decisions can be made to improve it.*

**Risk and Reliability in Practice**

Throughout this chapter of the book we will repeatedly draw on the field of flood management to illustrate risk and reliability concepts as this field requires expertise from all perspectives in civil and environmental engineering and geosciences, from climate and hydrologic processes to evacuation and recovery. In addition, this field has driven the development and use of risk and reliability techniques in the Netherlands since the flooding disaster of 1953, not to mention the experience gained during the previous millenium, albeit in a less rigorous mathematical fashion. And finally, new safety standards for primary flood defences in the Netherlands have been in place since 2017 that are formulated as a tolerable failure probability of dike[1] segments. As such, dike reinforcements are legally required to be designed according to these new standards, which requires one to show that the failure probability is less than an allowable maximum value.

Despite the focus on flood management, many applications exist in other fields, for example the discussion about the gas extraction in the North of the Netherlands which leads to increased earthquakes, building damage and potential injury to humans. A thorough analysis of the probability of earthquake occurrence, structural safety of various infrastructure (houses, levee, hospitals, pipelines), benefits associated with extracting gas and the resulting level of risk is required to make decisions about how to manage this industry. As

with the flood management application, advanced knowledge of probabilistic techniques is needed.

**MUDE exam information**

Concepts in this section to focus on for the Q2 MUDE exam are:

- **Risk Definition** and the **Risk Curve** (Section 8.1)
- Evaluation of risk using various **Safety Standards** (the last page of Section 8.2)
- In particular, it is essential that you can construct a risk curve and interpret its meaning, as well as assess it using the limit line approach

A few exercises are provided at the end of this chapter, but do not exhaustively illustrate the type of questions that you can expect on the exam. See exam questions from previous years, as well as in-class activities, for additional examples.

**MUDE not-on-the-exam information**

There are several pages included in this chapter that provides useful pre-requisite ot background information and *you are expected to read and understand them;* however, these pages do *not* need to be studied intensively for the exam. This includes:

- Steps of a Risk Analysis (second page in Section 8.1)
- Decision analysis, cost benefit analysis and economic optimization (first 3 pages in Section 8.2)
- The Paint System example (Section 8.3)

---

A dike is a structure, typically made of soil, that protects a specific region from flooding by physically holding back water. Usually associated with rivers, such structures are also widely used on the coast, especially in low elevation areas such as the Netherlands. The Dutch word for levee is *dijk,* but English word *dike* is used in this book. Outside of the Netherlands the words *embankment* and *levee* are used.

# Introduction to Risk Analysis

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

This book splits risk into two parts: risk *introduction* and risk *evaluation.* This chapter in particular introduces the *analysis* of risks from a broad perspective, beginning with definitions and key steps of a risk analysis, illustrated quantitatively through an FN curve. The next chapter covers risk *evaluation* concepts decision theory, cost benefit analysis and safety standards, where the principal focus is on making decisions and answering the question ‘how safe is safe enough?’

> **Note**
>
> The most important part of this chapter is the definition of risk and risk curves.

# Definition of Risk

## Contents


- Hazard, Vulnerability and Exposure
- Other Risk Definitions


Almost all activities in life are characterized by some level of risk. Especially in the design and management of engineering systems, risk and safety are key concepts and need to be taken into account explicitly. In social discussions no unambiguous meaning is assigned to the concept of risk. Two definitions given in the Oxford dictionary are: 1) a situation involving exposure to danger; 2) the possibility that something unpleasant or unwelcome will happen.

The first definition focuses on the consequences, the second on the possibility or probability. Quantifying and evaluating risks based on either probabilities or consequences (not both) is less realistic. For example, the risk of losing € 100 with a probability of 50% is different than the risk of losing € 1000 with the same probability. Also, the risk associated with losing a given sum of money will depend on the probability of the event. An often-used definition considers risk as an expected value:

*Risk is the probability of an undesired event multiplied by the consequences.*

The unit of risk now depends on the units of probability and consequences, where probability of an event is generally expressed as the probability per unit time, for example per year. The consequences of an undesired event often measure a diverse array of damages, such as material, ecological, injuries and fatalities. In many applications engineering consequences are expressed by means of a monetary value, in which case the unit of the risk (or expected value, $E(d)$) then becomes € per year. For a case with one event scenario $i$ with probability $p_i$ it yields:

$$E(d_i) = p_i \cdot d_i$$

A more general definition of risk has been given by Kaplan and Garrick (1981):

> Risk is a set of scenarios, $s_i$, each of which has a probability, $p_i$, and a consequence, $d_i$.

This definition allows the use of various so-called risk metrics (or risk measures) to quantify or depict risk. The expected value of the damage for a set of multiple discrete scenarios $i = 1, \ldots, n$ , can be expressed as:

$$E(d) = \sum_{S_{i=1}}^{n} p_i \cdot d_i \tag{8.1}$$

For the set of scenarios considered, the expected value quantifies risk precisely; however, it does not give insight in the magnitude of probability and consequences and the contribution of individual scenarios. For example, a single value does not indicate whether the risk is governed by a large number of scenarios with small consequences, or a few scenarios with large consequences and low probabilities. Therefore, an often-used companion risk-based tool is the risk curve, which shows the probability of exceedance and consequence for all scenarios. A well-known example of such a risk curve is the *FN curve*, which displays the probability of exceedance associated with $N$ (human) fatalities, schematized in Fig. 8.1. It is easy to compare the probability of exceeding a relatively low or high number of fatalities.

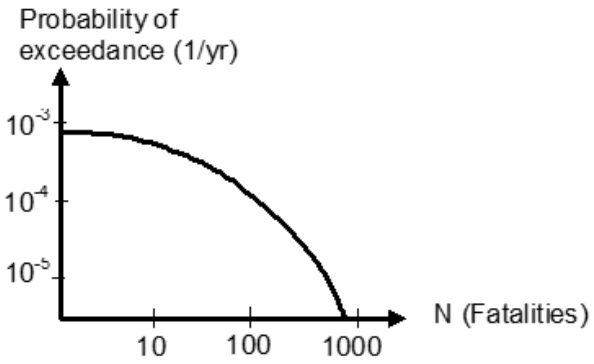



*Fig. 8.1* FN curve, showing the probability of exceedance of a certain number of fatalities N on Log-Log scale.

The FN curve was originally introduced in the 1960's for the assessment of risks in the nuclear industry (Farmer, 1967, Kendall *et al.*, 1977) and is now used to display and limit risks in a wide variety of industries around the world. It is an extremely useful way to quantitatively compare risk associated with a broad range of scenarios, and to make decisions. A famous example of this is shown in Fig. 8.2, which compares the risk estimated for a wide variety of engineering infrastructure. This figure is described further in the Risk Curve Section, and also illustrates the concept of acceptable risk, which is discussed in the Section on Safety Standards. In short, it allows one to begin answering the question 'how safe is safe enough?'

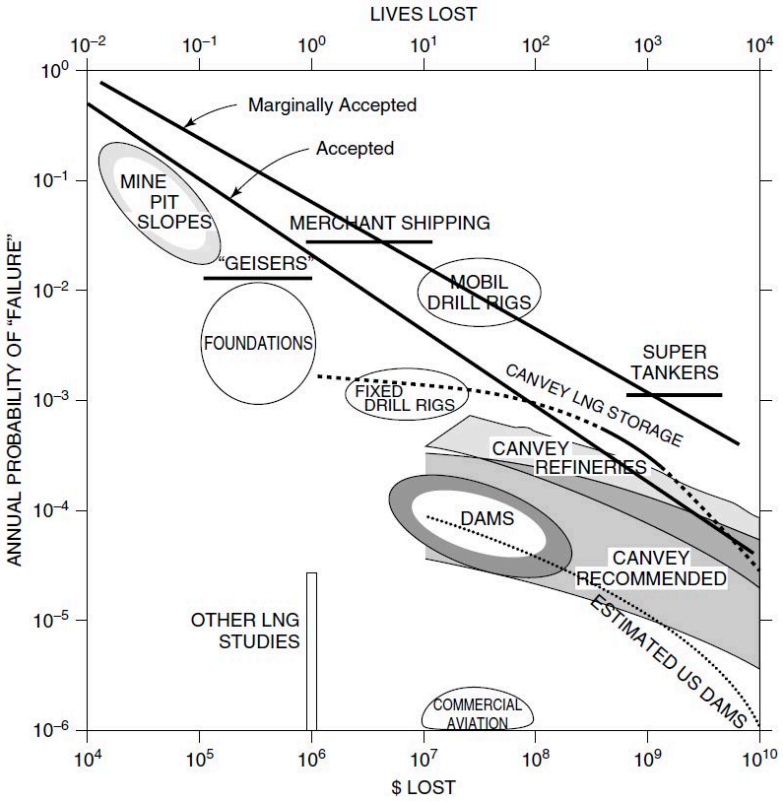



*Fig. 8.2* Risk curve for various engineering projects, showing probability of exceeding a specific number of fatalities, $N$, and damages, $, on a Log-Log scale, making it both a FN and FD curve; see Risk Curve Section. Source: Baecher and Christian (2003), based on Baecher (1982) and described in Baecher (2021).

# Hazard, Vulnerability and Exposure

Engineering disciplines that are focused on decision making where natural hazards play a significant role often define risk as the combination of hazard, exposure and vulnerability, as shown in Fig. 8.3.

- Hazard: A dangerous phenomenon, substance, human activity or condition that may cause loss of life, injury or other health impacts, property damage, loss of livelihoods and services, social and economic disruption, or environmental damage
- Vulnerability: The characteristics and circumstances of a community, system or asset that make it susceptible to the damaging effects of a hazard
- Exposure: People, property, systems, or other elements present in hazard zones that are thereby subject to potential losses

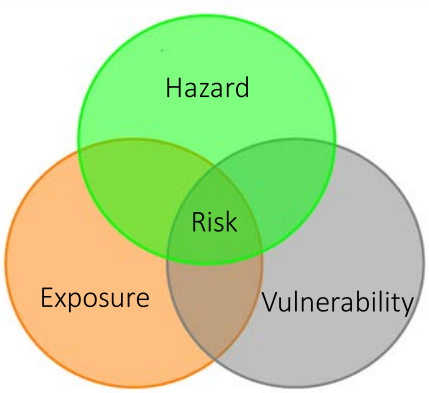



*Fig. 8.3* Alternative definition of risk as intersection of hazard, exposure and vulnerability. Risk is the small triangular-shaped area where all three circles overlap.

Often these terms are useful for framing qualitative discussions around complex topics where a quantitative analysis may be out of reach, for example, as done in IPCC reports on climate change (Cardona *et al.*, 2012). However, they can also be used to simplify the risk analysis by subdividing it into parts that are (computationally) independent. An analytic expression can easily be formulated by expanding the probability term of Equation (8.1):

$$p_i = P(N_i, E_j, H_k) = P(N_i|E_j, H_k) \cdot P(E_j|H_k) \cdot P(H_k) \tag{8.2}$$

where $N_i$, $E_j$ and $H_k$ are the vulnerability (fatalities), exposure and hazard of scenario $i$, respectively. Combining with Equation (8.1) and taking the sum over all combinations of variables ($i$, $j$, $k$) would give the total risk. In the case of our flood management case, an example of each term could be:

- Hazard, $H_k$: a high water level.
- Exposure, $E_j|H_k$: breach (failure) of a dike due to the high water level, $H_k$.
- Vulnerability, $N_i|E_j, H_k$: fatalities given the broken dike and high water level. $N$ is usually modeled as a random variable as it is dependent on factors such as evacuation effectiveness and flood warning systems.

# Other Risk Definitions

This book uses Equation (8.1) as the primary definition of risk. However, it is useful to highlight some risk concepts used in other domains.

Within economics, risk is generally associated with a deviation from the expected return or the probability of loss. In social sciences risk is often considered as a contextual notion or social construct. Vlek (1996) has summarized 11 formal definitions used in social sciences,

see Table Table 8.1. In some of these definitions (e.g. numbers 2 and 4) the perceived seriousness of the undesired consequences plays an important role. Examples of other, more informal risk definitions used in psychology are “the lack of perceived controllability”, “set of possible negative consequences” and “fear of loss” (Vlek, 1996).

Substantial research has also focused on factors that determine the perception of risk (e.g. Slovic (1987), Vlek (1996)) including: degree of damage, controllability of and familiarity with hazards, extent of benefits from an activity, and voluntariness of exposure.

*Table 8.1* Formal definitions of risk used in social sciences (Vlek, 1996)

| No. | Definition |
|---|---|
| 1 | Probability of undesired consequence |
| 2 | Seriousness of (maximum) possible undesired consequence |
| 3 | Multi-attribute weighted sum of components of possible undesired consequences |
| 4 | Probability · seriousness of undesired consequence (‘expected loss’) |
| 5 | Probability-weighted sum of all possible undesired consequences (‘average expected loss’) |
| 6 | Fitted function through graph of points relating probability to extent of undesired consequences |
| 7 | Semi variance of possible undesired consequences about their average |
| 8 | Variance of all possible consequences about mean expected consequence |
| 9 | Weighted sum of expected value and variance of all possible consequences |
| 10 | Weighted combination of various parameters of the probability distribution of all possible consequences |
| 11 | Weight of possible undesired consequences (‘loss’) relative to comparable possible desired consequences |

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Steps in a Risk Analysis

## Contents


- 1. System definition
- 2. Qualitative analysis
- 3. Quantitative Analysis
- 4. Risk Evaluation
- 5. Risk Reduction and Risk Control


> **Note**
>
> It is important to read and understand the content on this page, however, it does *not* need to be studied for the exam.

The previous section made it clear that risk is a function of probabilities and consequences. Risk analysis, therefore, consists of an analysis of probabilities and consequences associated with undesired events in a given system. Alternative terms used in literature are risk assessment and quantitative risk analysis (QRA).

A risk analysis is carried out because involved parties (e.g. designers, managers, decision makers) want to identify and evaluate the risks and decide on their acceptability. Outcomes of risk analysis can be used in the design process to decide on the required safety levels of new systems (e.g. a new tunnel) or to support decisions on the acceptability of safety levels and the need for measures in existing systems (e.g. a flood defence system). A quantitative measure of some form is needed to transfer decisions on acceptable safety into a technical domain (Voortman and Vrijling, 2004), such as the height of a flood defence or the strength of a building. In addition, risk analysis can be used to analyse the effectiveness of risk reduction measures, including management and maintenance strategies. Whereas the primary goal of a risk analysis is often to support rational decision-making regarding risk-bearing activities (Apostolakis, 2004), it also provides insight in the way a system may fail. As such, it can also serve as a tool of communication and management. Insights drawn from a risk analysis can be used to optimize system design and management and often there is a direct link to quality assurance.

In general, the following elements can be identified within a risk analysis, (based on: CUR (1997), Workgroup 32 (2001), Faber and Stewart (2003) and Jongejan (2008)):

1. System definition and setting the scope and objectives of the analysis;
2. Qualitative analysis of undesired events;
3. Quantitative analysis of the risk;
4. Risk evaluation (of the acceptability of the risk)

Each of these steps is illustrated in Fig. 8.4 and described in more detail below, providing context for the following chapters in this book. In addition to the risk analysis of a given system, *risk management* also includes the element 'risk reduction and control'. Depending on the outcome of the first four steps of the analysis, measures can be taken to reduce the risk, which will lead to changes in the system configuration and risk level. Steps 1–4 are often repeated several times with adjusted system specifications to obtain an optimal design.

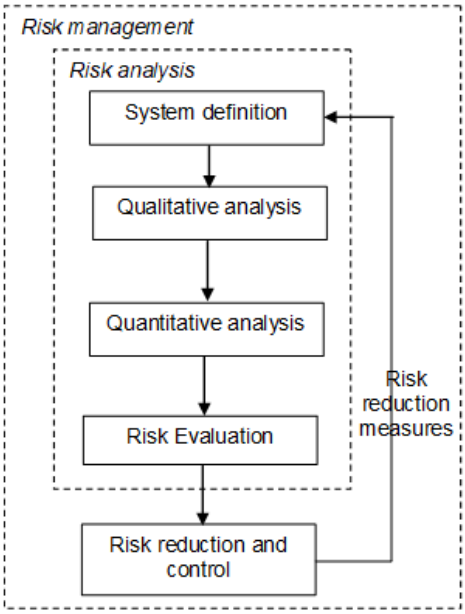



*Fig. 8.4* Schematic view of steps in risk analysis (risk assessment) and risk management.

# 1. System definition

This step entails the definition and description of the system as well as the scope and objectives of the analysis. Usually a system is divided into components and subsystems, which can be schematised graphically. Further information on the decomposition and analysis of system is provided in the rel_sys Chapter of this book, and illustrated through the example exercises.

A system can be represented in terms of physical components, organizations and users, and an external environment. In order to analyse failure and risks, not only physical components, but also organizations and operators and users need to be considered (see e.g. Bea (1998)), as different groups of organizations and persons will be involved in different roles. Consider the aviation industry, where professionals may be responsible for operation and management of the system (e.g. the pilots and crew in an aircraft), but important roles are also played by the users of a system (passengers) and external parties (people living near the airport exposed to risk and noise). Each group has a different relationship and attitude towards the risk, which could affect its acceptability. For example, a higher risk might be acceptable for pilots and crews in an aircraft (who derive a direct benefit from flying) than for regular passengers. Finally, the external environment (e.g. wind or runway conditions) will determine the loads on the system and affect potential failure mechanisms.

# 2. Qualitative analysis

In this step, potential hazards, undesired events, failure mechanisms and scenarios are identified and described. An important goal of this phase is to gain insight, as complete as possible, into all possible undesired events and their consequences. For most systems, multiple undesired events can be distinguished. For example, two events with different impacts that can both lead to flooding of a polder are 1) the inflow of large amounts of water due to a dike failure; 2) the inflow of smaller amounts of water when a sluice gate is not closed.

Failure occurs when a system no longer fulfils one or more desired functions, where the failed state can be reached through different failure mechanisms (or failure modes). For example, a dike can fail due to overtopping, but also due to geotechnical failure mechanisms such as instability or piping. A **limit state** is a condition of a structure beyond which it no longer fulfils the relevant design criteria (EN 1990:2002 E, 2001), of which two types are distinguished in practice: serviceability and ultimate limit states (SLS and ULS, respectively).

- **Ultimate limit state (ULS)**: failure or collapse of a system or structure occurs, for example, the breakwaters of a harbour entrance are destroyed as a result of extreme conditions. An example from structural engineering is the collapse of a roof of a building.
- **Serviceability limit state (SLS)**: exceedance leads to temporary and/or partial failure. For example, inability to use the harbour due to waves that are (temporarily) too high. Another example could be too much vibration of a structure, such that users experience discomfort.

Further information on the use of the SLS and ULS concepts in civil engineering is included in the rel_comp.

In a risk analysis, it is very important to get an overview of the various undesired events and failures before proceeding with a quantitative analysis, as many accidents in practice are caused by failing to identify failure modes. Unfortunately, creating a complete list of threats and failure modes is not an easy task. Support for this process can be found through are failure databases, literature studies, interviews, experiences with comparable systems, brainstorm sessions, et cetera. In addition, formal techniques for systematically identifying undesired events exist, such as Failure Modes and Effects Analysis.

# 3. Quantitative Analysis

The probabilities and consequences of the undesired events identified in step 2 are quantified in this step, and risk is quantified in as expected damage(s) or with a risk graph as described in the next Section.

## Probability

In simple cases the probability can be computed directly, but generally the undesired event must broken down into multiple steps or processes to facilitate the computation of probability, which are then recombined using integration or the total probability theorem. Such an approach results in systems of systems, each of which is composed of individual *components* (or elements), each of which is typically evaluated using the limit state concepts defined above. Often this the probability of a limit state being exceeded is referred to as the probability of failure, which of course reflects the component, not necessarily the system. Techniques for computing the probability of failure are covered in the rel_comp and rel_sys Chapters.

Whereas the most critical aspect of the previous step is identifying the most important failure modes, in the quantitative analysis it is critical to precisely define the quantity of interest, as ambiguity can lead to misunderstandings and incorrect assessment of risk. Often such ambiguities have a direct relationship with the conditional terms used in the probability computations, which imply specific statements about dependence (or independence). Consider the previous example of river level exceeding the dike height $P(h_w > h_{dike})$: the probability can be significantly different if a design lifetime of one year or fifty years is considered. In addition, the time of year may also play a role as different types of floods may occur. If the seasonal probability of flooding is binary and mutually exclusive with wet and

dry season each lasting 6 months of the year, the following expression holds (all values computed an a *per year* basis):

$$P(h_w > h_{dike}) = 0.5 \cdot P(h_w > h_{dike}|\text{wet}) + 0.5 \cdot P(h_w > h_{dike}|\text{dry})$$

Thus, if the risk analysis were only focusing on floods during the wet season, the probability of flooding would be overestimated by a factor on the order of 2 (assuming the probability is roughly equal in dry and wet seasons).

# Consequences

After failure has been defined, consequences of the events are quantified. First, physical effects associated with an undesired event are considered, such as heat and/or smoke from a fire, or inflow of water due to dike breach. Depending on the exposure of people or objects to the physical effects, damages, life loss or other impacts may occur. As an example, a specific consequence, $D$, of dike failure during a flood is considered as a sequence of three discrete events:

- The probability that a dike fails, $P(E_1)$
- The conditional probability that water flows into the polder given a dike breach $P(E_2|E_1)$
- The probability of damage given dike breach and inflow into the polder $P(D|E_1 \cap E_2)$

The probability of damage can now be computed by combining these terms:

$$P(D) = P(E_1)P(E_2|E_1)P(D|E_1 \cap E_2)$$

which is analagous to the hazard, vulnerability and exposure formulation of risk introduced in the previous Section.

Multiple types of consequences can be caused by one disaster, which are illustrated in Table 8.2 for the failure of a flood protection system. The damage is divided into tangible and intangible damage, depending on whether or not the losses can be assessed in monetary values. Another distinction is made between the direct damage, caused by physical effects of the event, and damages occurring outside the directly exposed area. The latter occurs when companies outside an impacted area experience damages, due to loss of demand from customers in the flooded area. In a risk analysis it is desirable to take into account a complete set of impacts; unfortunately many of the items from the table cannot be easily

quantified, if at all, thus the quantitative analysis and risk evaluation are often focused on economic damages and life loss.

*Table 8.2* General classification of damages, based on (Vrouwenvelder and Vrijling, 1996)

| | **Tangible** | **Intangible** |
|---|---|---|
| Direct | | |
| | Residences | Fatalities |
| | Structure inventory | Injuries |
| | Vehicles | Animals |
| | Agriculture | Utilities and communication |
| | Infrastructure and other public facilities | Historical and cultural losses |
| | Business interruption | Environmental losses |
| | Evacuation and rescue operations | |
| | Reconstruction of flood defences | |
| | Clean up costs | |
| Indirect | | |
| | Damage for companies outside the exposed area | Societal disruption |
| | Substitution of production outside the area | Damage to government |
| | Temporary housing of evacuees | |

# 4. Risk Evaluation

In the risk evaluation phase a decision is made whether the risk is acceptable or not and whether risk reduction measures should be implemented. In other words, a direct answer is sought to the question “how safe is safe enough?” (Starr, 1967). Results of the quantitative analysis provide input for risk evaluation and decision making, and different quantitative

approaches can be used to support risk evaluation, which are outlined here and introduced formally in the Risk Evaluation Chapter.

**Decision Analysis**: Also called 'decision making under uncertainty,' this approach aids the decision-making process by recording different possible outcomes along with the associated risks, costs and benefits, leading to the optimal selection from a number of alternatives. Often visual tools are used such as a risk matrix or decision tree.

**Cost Benefit Analysis**: costs and benefits of risk reduction measures are considered. When a large number of design choices are possible, an **economic optimization** can be applied to select an optimal system design, based on the costs and benefits of risk reduction.

**Safety Standards**: risk is compared with predetermined safety standards to directly determine acceptability. Such standards are typically imposed by government organizations or standards of practice and often focus on loss of life as the primary risk metric.

Given the nature of the key question "how safe is safe enough?", several political, psychological and social processes play a role in the evaluation of risk—in other words: risk evaluation is not purely a technical process, but involves many subjective elements and decisions. One difficulty facing regulators is that preferences and risk attitudes within society may diverge and that costs and benefits may not be distributed evenly, and that a single, collective decision has to be based on strongly divergent individual preferences. In practice, this implies that the establishment of collective decision making procedures is inevitably a political process. This ambiguity can also be found in the numerous interpretations of "the" precautionary principle, which is interpreted by some as a decision making criterion that requires proof of harmlessness (a scientific impossibility), whereas it is seen by others as a decision making procedure that puts emphasis on dialogue and stakeholder involvement, (e.g. Jongejan (2008)).

# 5. Risk Reduction and Risk Control

If the risks evaluated in first four steps of a risk analysis are considered unacceptable, several forms of risk reduction can be implemented, such as changes to the engineered system, or changes to its organization and management. It can be helpful to determine how risk can be controlled, for example by monitoring, inspection or maintenance.

Post-accident analysis indicates that human and organizational errors are still a major cause of failure in civil engineering systems, and it seems that the only suitable way to reduce human errors is by the incorporation of sufficient control in different phases of the construction process (Taerwe, 1986), and by a thorough education of all personnel involved.

Therefore, an extensive interaction between the safety methodology and quality management is a necessity in order to guarantee the safety of infrastructure.

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Risk Curve

## Contents


- A Famous Risk Curve
- Construction of an FN Curve


As introduced in the Definition of Risk Section, the risk curve is useful for evaluating the expected damages for a wide range of scenarios. In contrast to the total expected damages, it allows the comparison of relative magnitude of probability of occurrence and associated consequences. An FN curve tends to be the most widely used, where the 'N' refers to number of fatalities and the 'F' in comes from the CDF of the probability distribution of $N$, $F_N(n)$, although the curve itself actually represents the complementary CDF, $1 - F_N(n)$. It is often plotted on a log-log scale to allow comparison over a wide range of scenarios. While in theory a risk curve can be derived from the probability density function (or probability mass function for discrete cases), the distribution of damages is rarely known and must be approximated by considering a finite and suitable set of discrete events. Thus, in practice, the curve is constructed using scenarios as defined previously in our risk definition (Equation (8.1)).

Although the FN curve is widely used, the risk curve is not exclusively applied to evaluate expected number of fatalities, $N$. In principle any measurable quantity can be used to evaluate risk, referred to as a *metric,* although it can be difficult to apply this to some situations. For example, assessing the value of environmental characteristics can be challenging because the benefit is not easily related to a measurable quantity. The area of impacted wetland habitat during a flood (e.g., $km^2$) can be used as a risk metric, but this may not properly represent actual damage, which may recover quickly or slowly depending on the frequency of inundation. Financial damage (e.g., €) is often considered as a risk metric, in which case the FN curve is called an *FD curve.*

**Fatalities and Damages**

In some cases it is useful for planning and design purposes to equate $N$ and $D$, but this places a specific financial value on human life, which raises difficult moral and ethical concerns. This topic is discussed further in the Valuation of Human Life Section.

# A Famous Risk Curve

A risk curve is often useful because it can convey a large amount of complex information in an easily recognizable format. University lectures and conference presentations around the world which focus on risk, especially for civil engineering applications, will often begin with a casual reminder that 'risk is probability times consequences' and show some version of a risk curve. Fig. 8.5 is the most famous risk curve, having been shared widely over the last decades, perhaps because it so nicely compares a wide range of engineering projects.

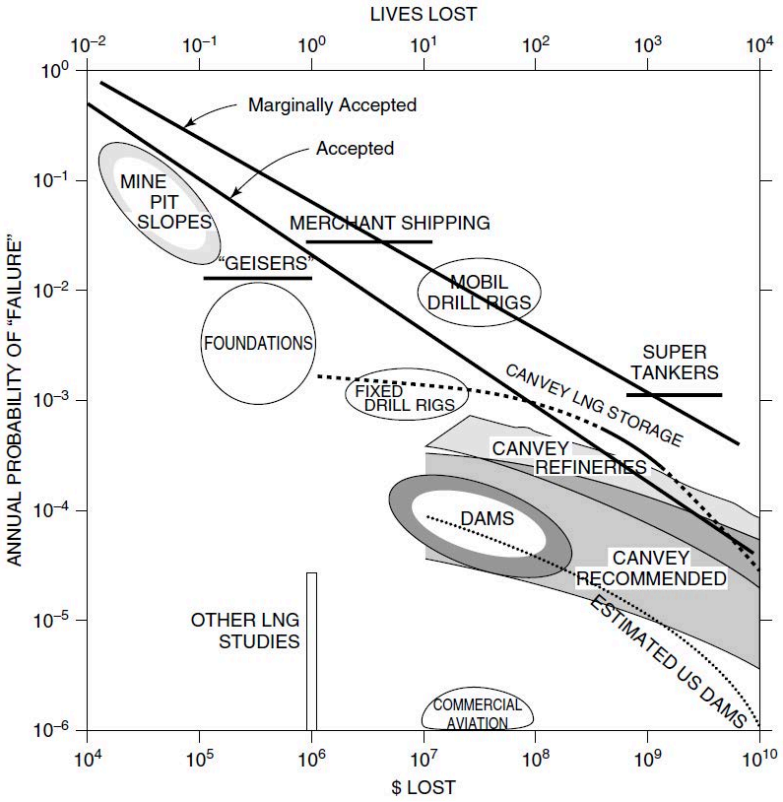



*Fig. 8.5* Risk curve for various engineering projects, showing probability of exceeding a specific number of fatalities, $N$, and damages, $, on a Log-Log scale, making it both a FN and FD curve; this figure is also included in the Definition of Risk Section. Source: Baecher and Christian (2003), based on Baecher (1982) and described in Baecher (2021).

Fig. 8.5 was created as part of a risk analysis for a petrochemical storage facility in Tokyo Bay, Japan, where large tanks, constructed on soft soils (hydraulic fill), are vulnerable to damage during an earthquake. Of primary concern was to prevent the petrochemicals entering bay waters, thus not only failure of the storage tanks, but also the containments systems in place to collect spilled material (a complicated system reliability problem). The client needed to convince the regulator that the final design was sufficient for keeping the risk of loss of containment to a sufficiently small level, which the project team was able to do by comparing the risk calculated for the Tokyo site to other engineering works around the world. The figure clearly illustrates what the estimated risk is for each project, and by making the assumption that these already-constructed projects provide a level of safety that is

acceptable to society (primarily via regulatory agencies), the 'acceptable' line can be used to approve other projects. This concept of acceptable standards based on risk curves is covered in depth in the Section on Safety Standards.

The origin of this figure was described a Terzaghi Lecture by Professor Gregory Baecher, which can be viewed on YouTube: *IFCEE 2021: Karl Terzaghi Lecture: Greg Baecher: Geotechnical Systems, Uncertainty, and Risk* (Baecher, 2021)}. The introduction to the Tokyo Project begins at 16:16, and the explanation of the risk curve is given at 27:00; however, the entire lecture is well worth listening to and closely related the subject of this book.

# Construction of an FN Curve

The following three figures illustrate the computation of an FN curve for a system with two mutually exclusive event scenarios:

- Accident 1 with $N_1 = 10$ fatalities and a probability of $P_1 = 10^{-2}$ per year
- Accident 2 with $N_2 = 100$ fatalities and a probability of $P_2 = 10^{-3}$ per year

First, the probability mass function is created $f_N(n) = P(N = n)$, taking care that the probability of no fatalities is included:

$$P(0) = 1 - P_1 - P_2 = 1 - 0.01 - 0.001 = 0.989$$

The cumulative distribution function, $F_N(n) = P(N \leq n)$, can thus be easily computed, as well as the probability of exceedance, $1 - F_N(n)$, also known as the *FN curve.* Note that if a risk metric such as damage in Euros was being used, we would call this the *FD curve* and use $D$ to denote the random variable, as in $f_D(d)$ and $F_D(d)$

While the curves below illustrate probability and consequences associated with specific scenarios, the expected value of the distribution can be computed, which is equivalent to the area under the FN curve. This is useful can be used to assess the entire system. For this example, the expected value of fatalities per year for all scenarios can be computed as follows:

$$E(N) = P_1 N_1 + P_2 N_2 = 0.2 \quad \text{(fatalities per year)}$$

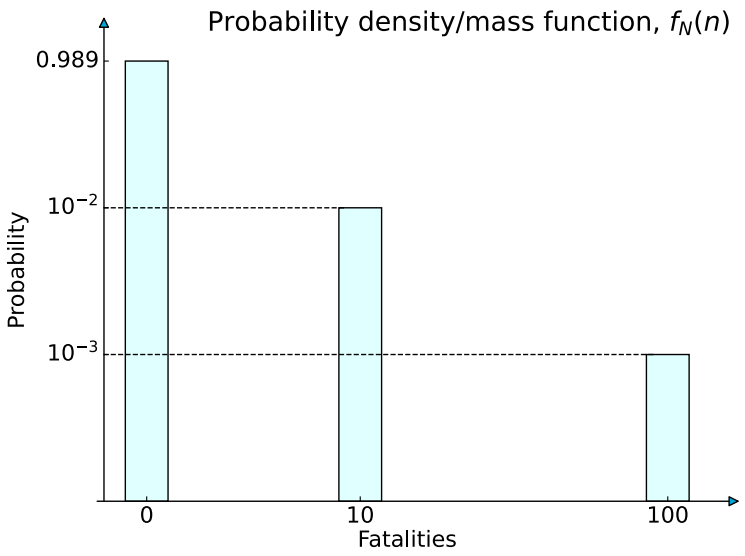



*Fig. 8.6* Probability mass function (PMF), $f_N(N)$.

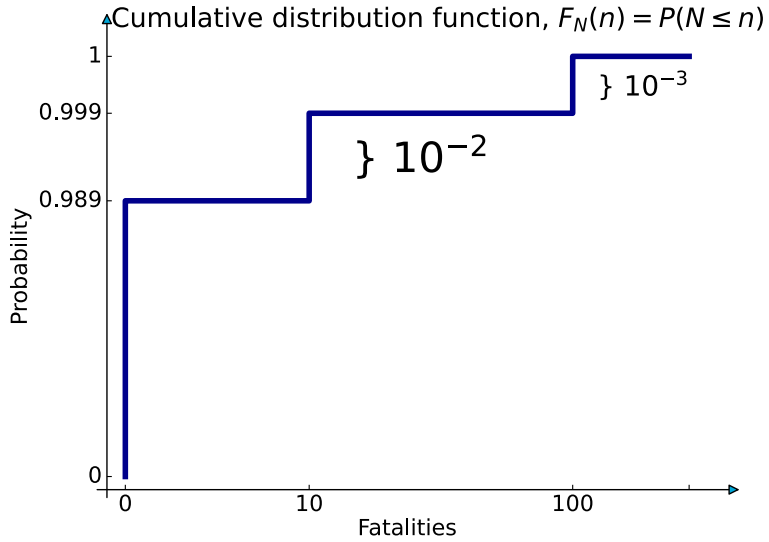



*Fig. 8.7* Cumulative distribution function (CDF), $F_N(n)$.

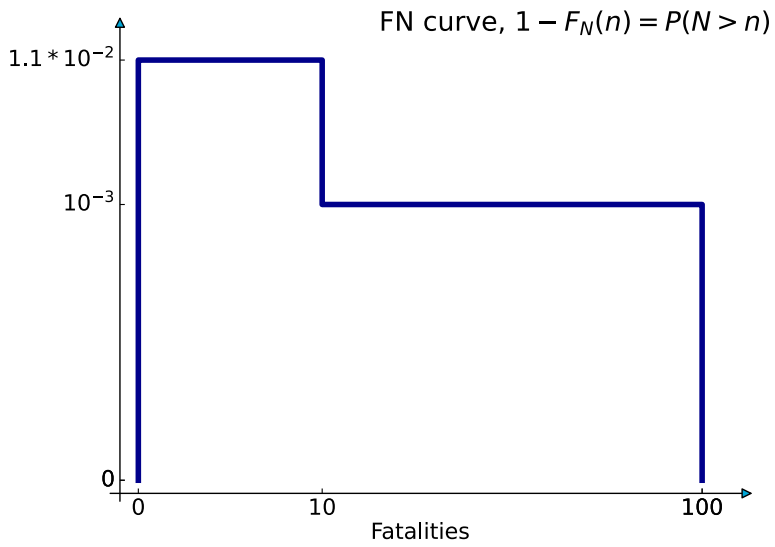



*Fig. 8.8* Exceedance probability (FN curve), $1 - F_N(n)$.

**Practice Exercise**

You can practice constructing a simple FN curve in FN Curve, although some of the questions require knowledge of limit lines, introduced in the Safety Standards Section.

**Attribution**

This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Risk Evaluation

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman. Find out more here.

This chapter focuses on the evaluation of risk (Step 4) by applying several different techniques. While the Sections may seem long, the methods are relatively simple and a lot of examples and explanations are provided to illustrate how they can be applied in practice. After this chapter, you will be ready to try all of the practice problems in the Exercise Chapter.

> **Note**
>
> The most important part of this chapter is the last page, which introduces several techniques for assessing risk. The other sections provide background information and examples that are useful for understanding the concepts, but are not essential for the exam.

# Decision Analysis

## Contents


- Introduction
- Decision rules


> **Note**
>
> It is important to read and understand the content on this page, however, it does *not* need to be studied for the exam.

Decision analysis, or decision-making under uncertain conditions is part of everyday life: when choosing to buy a lottery ticket or choosing to take an umbrella during cloudy weather. In contrast to the rather intuitive decision making in everyday matters, a structured analysis of different alternatives with associated risks, costs and benefits is very useful for decisions in (civil) engineering. This chapter offers a very basic introduction into the decision theory with applications to decision problems in the civil engineering domain. Further reference is made to the work by other scholars for more rigorous and detailed treatment of this topic. See, for example Raiffa and Schlaifer (1961); Benjamin and Cornell (1970).

## Introduction

Making a decision is in fact choosing from alternatives. The decision theory is based on the classic “Homo Economicus” model. It assumes that the decision-maker:

- has complete information about the decision situation;
- knows all the alternatives;
- knows the existing situation;
- knows which advantages and disadvantages each alternative provides, be it in the form of random variables;
- strives to maximise that advantage (formally called utility).

The decision-making concept discussed in these lecture notes assumes this model, although decision-making in practice is often different since the above conditions are often not fulfilled. For example, there can be multiple decision makers and multiple objectives, and in reality the decision maker does not know all the alternatives or their outcomes. For many practical cases this has led to an extension of the decision model, but not to a fundamental adjustment of the classical model described here.

Within a decision problem the following characteristics can be distinguished:

- the set of all possible actions or decisions $a$, from which the decision maker can choose
- the set of all (natural) circumstances, $\theta$ that influence the outcomes*
- the set of the set of all possible results, $\omega$, which are functions of the actions and circumstances: $\omega = f(a, \theta)$.

The actions, natural circumstances and the outcomes can be shown in a so-called decision tree (Figure Fig. 8.9).

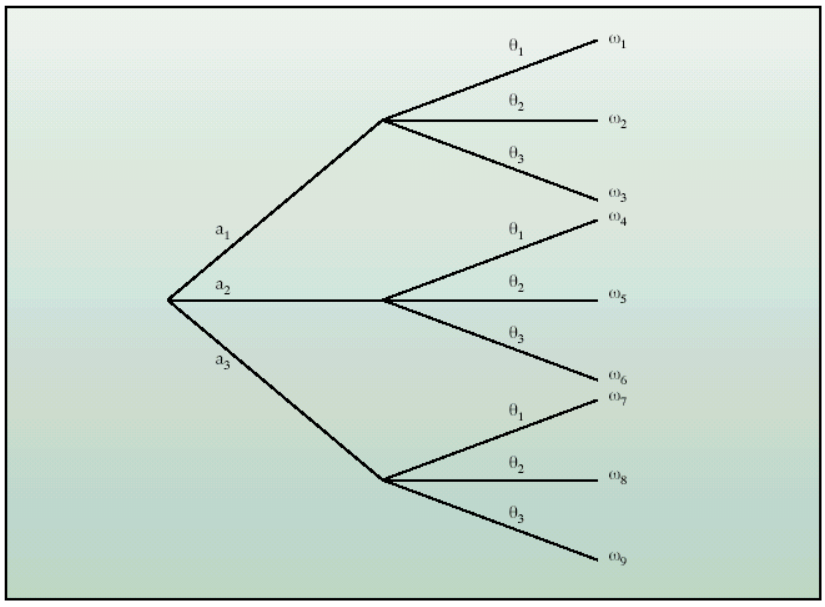



*Fig. 8.9* Decision tree.

Based on the possible results a choice is made for an action. To be able to assess the different results, a numerical value is assigned to each outcome, $\omega$, which can be used to establish the benefit of each outcome. This number can be a monetary value, a number on an arbitrary scale or utility–as long as the decision maker(s) can establish a consistent ranking of the outcomes with it. In the last two cases the benefit has no absolute value, but only gives the relative value of the different outcomes.

Utility, $u$, is a concept used to rank the possible outcomes according to the preferences of the decision maker, with possible values $0 \leq u \leq 1$ . A utility function can be used to

characterize the relative utility of various outcomes. The elaborations below are based on the monetary values as a measure for the outcomes and assume a risk neutral decision maker. This is a decision maker who is indifferent between choices with equal expected outcomes, even if one choice is riskier than the other. For example, a risk neutral decision maker would have the same preference for a € 400 pay out, or a 50/50 bet with a coin toss with outcomes of € 0 (head) or € 800 (tail). Utility and risk aversion are further discussed in later sections.

# Decision rules

Once a set of actions, circumstances and outcomes is known, various approaches can be used to come to a preferred decision. Various deterministic decisions rules are available which do not take into account the probabilities of the possible circumstances and outcomes. One example of such a decision rule is the minimax criterion: a decision maker wants to minimize maximum losses. This is in fact a risk-averse criterion. Another example is the maximax criterion: a decision maker chooses the option with the maximum income and is risk seeking.

Although these decision rules are helpful in some cases, the probability of occurrence of certain circumstances is a key feature of the decision problem. Information on the probability of outcomes is needed for an optimal choice of action(s). For example, when making a decision to start a business in soup or ice cream, the decision maker would want to know what the probabilities of rainy or sunny weather are. Selling ice cream in Dutch winter will probably not make a good (expected) profit, but it would be a profitable business in a Mediterranean summer.

Therefore it is necessary to include information on the probabilities of circumstances and outcomes, in order to determine a rational action with the highest expected value of the benefit. This theory is known as the Bayesian decision theory. In a probabilistic or Bayesian decision framework the optimal action $a*$ is defined as the one maximizing the expected utility. The following formula is found for the case with discrete outcomes.

$$a^* : \max_a E(u(a,\theta)) = \max_a \sum_\theta u(a,\theta_i)P(\theta_i)$$

In which $u(a,\theta)$ is the utility of action a under circumstance $\theta$ and $P(\theta)$ is the probability that circumstance $\theta_i$ occurs.

A rational decision is choosing the action with the highest expected (utility) value or highest benefit if outcomes are expressed in monetary values. This is illustrated in the example

below. Note that other examples in these lecture notes will also be based on monetary values.

**Example: buying shares or bonds?**

Suppose a person has €1000 at his or her disposal and is given the choice to invest this money in bonds or in shares of a given company. It is known that, on a yearly basis, 3 % of the current market value is distributed as interest on the bonds. The dividend of the shares depends on the company's profit. Suppose that the board of directors have made the following agreements:

- for a profit smaller than 5% of the shareholders capital, no dividend is paid; for a profit larger than 5% of the shareholders capital, dividend is paid, the percentage of which corresponds to 3% of the current market value of the shares;
- for a profit larger than 10% of the shareholders capital, the dividend corresponds to 6% of the current market value of the shares.

The set of actions A has two elements: $a_1$= investing in shares AND $a_2$= investing in bonds The set or market circumstances N has three elements, namely:
$\theta_1$ = company profit ≤5 %
$\theta_2$= 5 % < company profit ≤ 10 %
$\theta_3$= company profit > 10 %

Assume the inflation amounts to 2 %. The set of outcomes ω contains three possible outcomes for the shares: $\omega_1$= return (0 % -2 %) = -2 % per annum $\omega_2$= return (3 % -2 %) = 1 % per annum $\omega_3$= return (6 % -2 %) = 4 % per annum

Note that for the bonds the net outcome always yields $\omega_2$ =1% (i.e. 3% interest – 2% inflation). The outcomes can be illustrated using a decision tree (see Fig. 8.10) or a table (see Table 8.3).

*Table 8.3* Outcomes given the decisions ($a_1$, $a_2$) and market conditions ($\theta_1$, $\theta_2$, $\theta_3$ ).

| | **Market circumstances** | | |
|---|---|---|---|
| Actions | $\theta_1$ | $\theta_2$ | $\theta_3$ |
| $a_1$: buy shares | $-2$ % | 1 % | 4 % |
| $a_2$: buy bonds | 1 % | 1 % | 1 % |

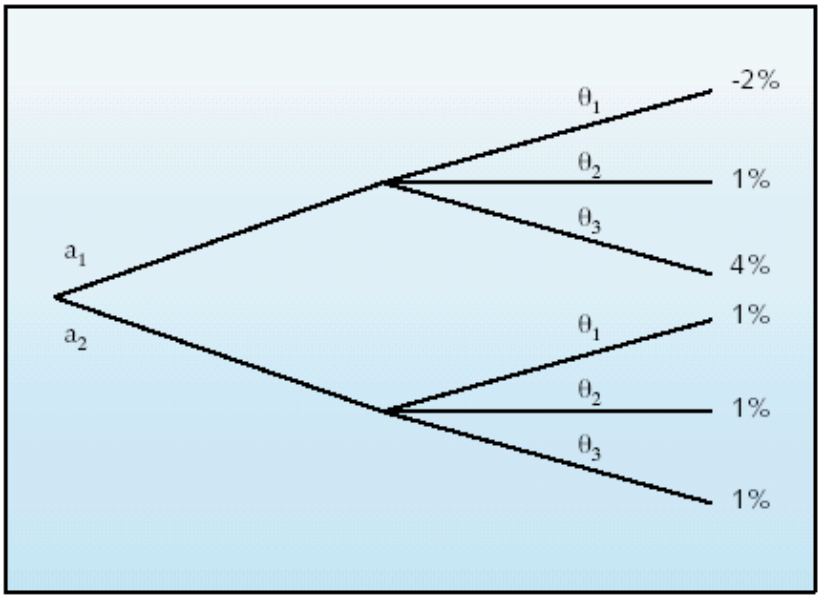



*Fig. 8.10* Decision tree for the example of buying shares ($a_1$) or bonds ($a_2$).

The deterministic decision rules can be applied to this example. Minimax would result in investing in bonds ($a_2$), maximax would result in buying shares. The optimal decision can be found by taking into account the probabilities of the market circumstances. These three circumstances are assumed to be exhaustive and mutually exclusive (i.e. outcomes cannot overlap and the sum of probabilities equals 1). The probabilities are estimated at $P(\theta_1)$ = 0.2; $P(\theta_2)$ = 0.3; $P(\theta_3)$ = 0.5. These probabilities can now be included in the decision tree. The expected value of the return of the actions is as follows:

Buying shares: 0.2(-2 %) + 0.3 (1 %) + 0.5 (4 %) = 1.9 %.
Buying bonds: 1%

In this case the expected outcome is larger for buying shares than for buying bonds. So for a risk neutral decision maker buying shares (a1) would be the preferred action. Note that this action also includes a probability of 0.2 of a loss. This is also expressed by a higher standard deviation of the expected outcomes for buying shares. The above example can also be extended by applying different utility functions for various outcomes.

In the previous example, the number of circumstances is limited and the probability distribution of the circumstances is discrete. For many decision problems this is not the case. The state of nature, for instance, can assume many values that cannot be made discrete. This, for example, would have been the case if the dividend in example 0 had been a percentage of the profit. In such cases a probability density function can be used to

characterize the spectrum of outcomes. Using a continuous form of formula (3.5), the expected value of various actions, and the optimal action / decision can be identified.

In taking decisions with uncertainties, it appears that probabilistic calculation techniques are a valuable aid to reach a rational choice. This is particularly the case if risks are dependent on the possible decisions. In such cases, Bayesian decision theory minimizes the total costs (i.e. investment costs plus risk in terms of potential losses). This can best be illustrated by means of an example from the civil engineering domain.

**Example: drainage of a construction site – decision tree**

In a river polder a basement has to be built in an excavated construction site. The construction site is made of sheet piling and the bottom is sealed off with a clay layer with a thickness of approximately 2.5 m. The thickness is not known exactly but measurements indicate that a normal distribution can be used with mean $\mu_d$= 2.5 m and a standard deviation of $\sigma_d$ = 0.2 m.

The river cuts through the clay layer and the underlying sand layer is fed by the river, which causes high uplift pressures on the clay which is exposed in the excavation (see Figure Fig. 8.11). If the pressure becomes too high the excavation base will push upward and water will enter the excavation in an uncontrolled way, damaging property and perhaps risking human life.

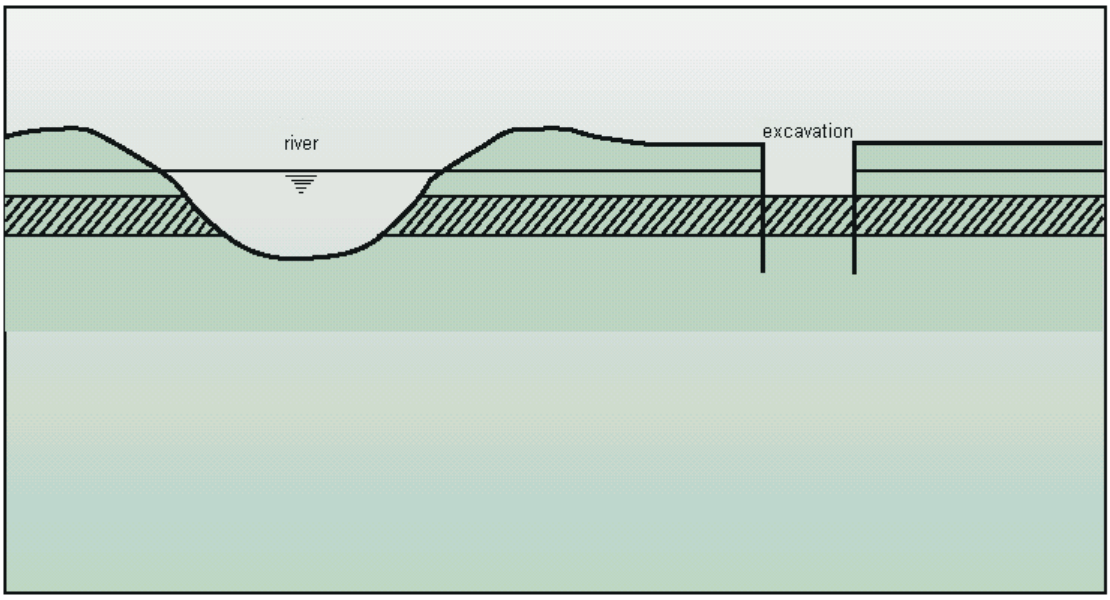



*Fig. 8.11* Cross-section of excavation and river.

Measurements of the groundwater levels indicate the maximum upward pressure, $h$, under the sealing layer in the construction period has an a normal distribution with an expected value of $\mu_h$ = 4 m water column and a standard deviation of $\sigma_h = 0.75\mathrm{m}$ (see Figures Fig. 8.12). Since $h$ is a hydraulic head (units of $[\mathrm{L}]$), pressure is calculated by multiplying the unit weight of water ($10\ \mathrm{kN/m^3}$).

Failure occurs when the uplift pressure exceeds the weight of the clay layer, which can be described by a limit state function $Z = R - S$ with resistance $R$ and load $S$:

$$Z = \rho_c \cdot d - \rho_w \cdot h$$

Where $\rho_c$ is the density of clay ($20\ \mathrm{kN/m^3}$), and is assumed here to be a deterministic value. The probability of failure is thus $P(Z<0)$, which for this situation can be found by calculating the mean and standard deviation of $Z$:

$$\mu_Z = \rho_c \mu_d - \rho_w \mu_h = 20(2.5) - 10(4) = 10\ \mathrm{kN/m^2}$$

$$\sigma_z = \sqrt{\rho_c^2 \sigma_c^2 + \rho_w^2 \sigma_{\mathrm{h}}^2} = 8.5\ \mathrm{kN/m^2}$$

Since $Z$ also has the normal distribution, we find the failure probability:

$$p_f = P(Z<0) = \Phi\left(-\mu_z/\sigma_z\right) = \Phi(-1.17) = 0.12$$

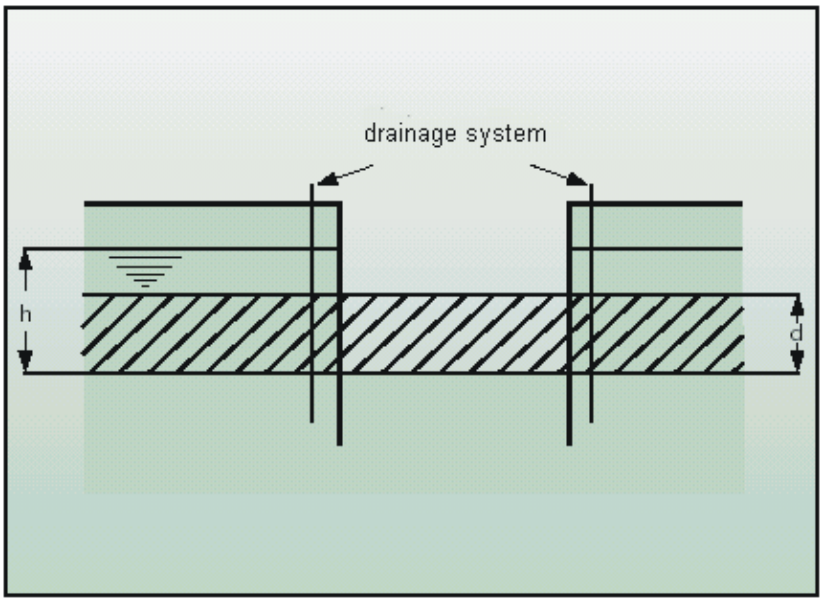



*Fig. 8.12* Cross-section of excavation near a river. Hashed area indicates the clay layer, which is underlain by sand. The water pressure is given by $h$ and is directly related to the river level (Figure Fig. 8.11).

The effect of a drainage system in the construction site (see Fig. 8.11) on the groundwater levels has been reviewed using groundwater flow calculations. It appears that it reduces the mean value of the maximum water levels to $\mu_h$ = 3.52m and the standard deviation remains the same. In this case the failure probability is reduced to 0.04. Such a drainage system costs €150,000.

The flooding of the construction site will result in estimated damages of €5,000,000, and the designer of the construction site is faced with the choice whether drainage facility is a worthwhile investment. To provide insight, the

decision problem is illustrated with a decision tree, but first the set of actions, circumstances and outcomes are defined:

The set of actions A consists of:

- $a_1$= excavation without drainage
- $a_2$= excavation with drainage

The set of circumstances N is formed by:

- $\theta_1$= the sealing layer remains intact
- $\theta_2$= the water pressure exceeds the weight of the sealing layer

The set of outcomes $\Omega$ consists of:

- $\omega_1$ = nothing happens; loss = € 0
- $\omega_2$ = the construction excavation is flooded: loss = €5,000,000

The previous analysis has shown that the probability of flooding of the excavation equals $p_f$ = 0.12 for a situation without drainage and $p_{fd}$ = 0.04 with drainage.

Without drainage, the risk, defined as the expected value of the loss, is

$$\text{risk, without drainage} = 0.12 \cdot € \, 5,000,000 = € \, 600,000$$

With drainage the risk is:

$$\text{risk, with drainage} = 0.04 \cdot € \, 5,000,000 = € \, 200,000$$

Costs and probabilities can also be shown in the decision tree (see Figure Fig. 8.13). The expected values of the costs can be calculated for the different actions by adding the present values of the cost of actions and risk:

- $a_1$ : expected value (additional) costs = risk =€ 600,000
- $a_2$ : expected value (additional) costs = extra costs + risk = € 150,000 + € 200,000 = € 350,000

This implies that the construction of the drainage system is rationally speaking the best decision for a risk neutral decision maker.

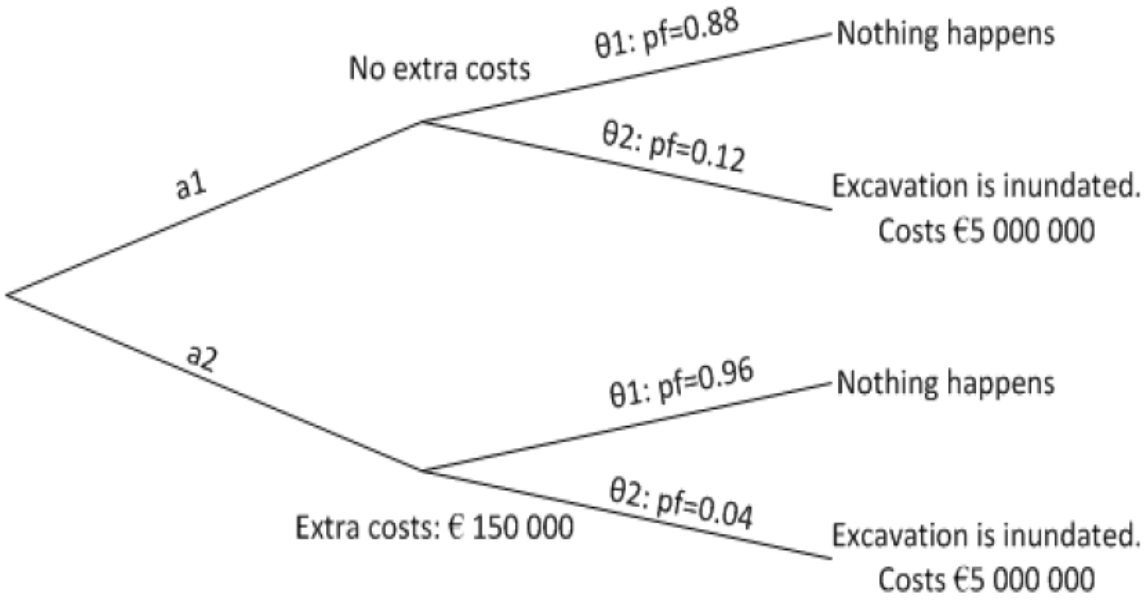



*Fig. 8.13* Decision tree with probabilities and costs.

**Attribution**

This chapter reuses content written by Bas Jonkman. Find out more here.

# Cost Benefit Analysis

## Contents


- Risk Reduction via Probability
- Risk Reduction via Damages
- Net Present Value
- Valuation of Human Life


> **Note**
>
> It is important to read and understand the content on this page, however, it does *not* need to be studied for the exam.

This section deals with simplified cost benefit analysis for risk reduction interventions in the engineering domain. An important question in evaluating (engineering) projects is whether the benefits outweigh the costs. Cost benefit analysis (CBA) is generally used for appraisal of a wide range of effects of projects or interventions in order to support decision making. The cost benefit analysis starts with defining the system and existing situation. Then, a broad range of effects of the proposed project and intervention can be identified.

Table 8.4 below shows an overview of effects of the Delta works that were built after the 1953 flood disaster in the Netherlands. The main aim of the delta works was to provide flood protection to the Southwest of the Netherlands. However, other effects included the agricultural benefits to the region (benefits) and the effects on environmental quality in the estuaries in which dams were built (costs or negative effects).

*Table 8.4* Costs and benefits of the delta works

| Costs | Benefits |
|---|---|
| Construction and maintenance costs | Reduction of flood risk (damage, loss of life) |
| Loss of environmental quality | Improvement of infrastructure |
| | Recreation |
| | Agricultural benefits |
| | Economic stimulus for the region |
| | Economic stimulus for the water engineering sector |
| | National Pride |

A choice has to be made for which effects in the CBA are evaluated in monetary terms. For some items, such as construction costs or economic risk reduction (see below) this is straightforward. For other items such as environmental effects or reduction of risk to life, approaches for monetary evaluation exist, but they are not standardized or undisputed.

# Risk Reduction via Probability

When considering investments that primarily focus on risk reduction (e.g. dike reinforcement or reinforcing of buildings for earthquakes), the main benefits will consist of the reduction of expected economic damages. For a measure to be cost effective, the investments should be smaller than the risk reduction.

$$I < \Delta E(D) \tag{8.3}$$

where:

- $I$ is investments
- $\Delta E(D)$ present value of risk reduction

This formula can also be used to calculated the benefit/cost ratio, i.e. $\Delta E(D)/I$, which should be greater than 1 for cost effective projects. Note that all costs and the risk reduction are given in terms of present value with a unit of €.

For investments that focus on prevention (i.e. reducing the failure probability of the system) (8.3) can be formulated as follows.

$$I < (P_{f,0} - P_{f,new})D$$

where:

- $P_{f,0}$ the initial failure probability
- $P_{f,new}$ the failure probability after risk reduction investment

**Example: Drainage of a construction site - costs and benefits**

The example from the previous section could also be formulated as a simplified cost-benefit analysis. The same values for the variables are used. The costs of the drainage are €150,000. The benefits are equal to the risk reduction, i.e.

$$\Delta E(D) = (P_{f,0} - P_{f,d}) = (0.12 - 0.04)€5,000,000 = €400,000$$

This shows that costs are smaller than the risk reduction benefits, i.e. $I < \Delta E(D)$. The benefit/cost ratio equals €400,000/€ 150,000 = 2.66

One can easily show that the investment would just be acceptable if the benefits and costs are equal, i.e. when the benefit / cost ratio equals 1. This would still be the case if the probability of failure with drainage equals $P_{f,d} = 0.09$.

# Risk Reduction via Damages

Other types of interventions do not affect the probability of an accident, but focus on reducing the damages. They are often indicated as mitigation. An example in the field of flood management concerns raising buildings instead of reinforcing the dikes. In this case the criterion becomes:

$$I < P_{f,0}(D_0 - D_{new})$$

where:

- $D_0$ the initial damage [€]
- $D_{new}$ the damage after investments in reducing the consequences [€]

# Net Present Value

The foregoing assumes that both the costs and benefits are expressed in the same unit, generally in the form of a present value [€]. The net present value (NPV) represents the sum of the present values of the benefit and cost cash flows over a period of time. In engineering we often deal with situations with larger initial investments, whereas the risk reduction benefits are spread out over a longer time period. For example, when a government invests this year in flood protection, the costs are made in year 0, but the benefits will be spread over the coming decades.

For such situations the failure probability is generally expressed per unit of time, mostly per year. That means that the risk (reduction) is expressed in terms of € per year, whereas the initial investments have the unit of €. The net present value of cost or benefit values over a future range of years can be calculated with (8.4). To calculate the net present value NPV [€] a discount rate should be used. The discount rate represents a required return on an investment.

$$NPV = \sum_{t=1}^{T} \frac{C_t}{(1+r)^t} \tag{8.4}$$

where:

- $C_t$ the costs in year $t$ [€]
- $T$ the reference period [years]

**Interest Rates**

The discount rate, $r$, is analagous to opportunity costs and interest rates that you may be familiar with from the field of economics. It provides a way to evaluate the true cost of investing in a particular project over a long time period, for example: if the value of a stock market increases with an annual interest rate of 4%, how does that compare to the expected earnings on the alternative investment under consideration?

Although civil engineering projects are typically not associated with financial investments like stocks and bonds, the need to consider the time value of money is identical. In fact, many governments use the financial instruments like bonds to finance projects, which is easy to see the relationship to interest rates.

The contribution of costs in a certain year to the net present value depend on the discount rate and the reference year. Costs or benefits closest to the present will have the greatest contribution. Fig. 8.14 below shows the value of $1/(1+r)^t$ for a given year $t$ for different discount rates. The higher the discount rate, the smaller the contribution to the net present value of costs or benefits that are far away from the present. For an infinite time horizon it can be shown that $\sum 1/(1+r)^t \approx 1/r$. This approximation can be used in evaluating engineering projects with a long life time.

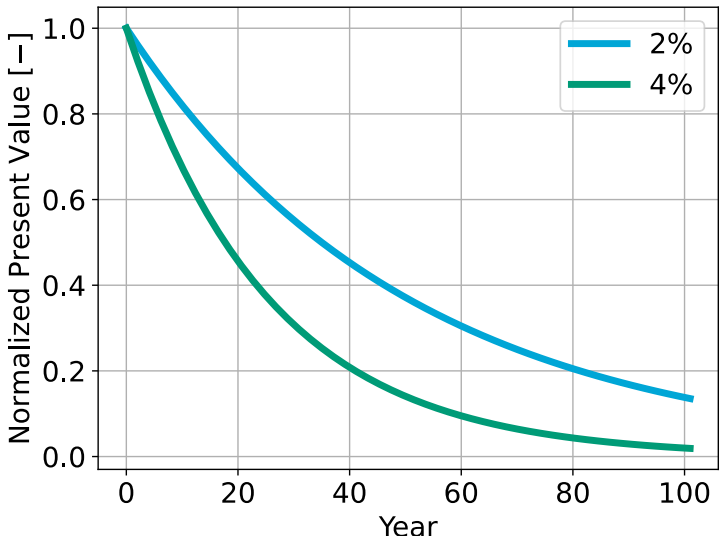



*Fig. 8.14* Normalized present value of a cost or benefit in year $t$ for different discount rates.

# Valuation of Human Life

The previous paragraphs have focussed on the evaluation of economic costs and benefits of risk reduction interventions. Many of these interventions also directly focus on reducing injuries and fatalities. Examples are regulations and investments in traffic safety which have introduced measures such as airbags and seat belts.

In literature on risk management the economic valuation of human life is often depicted as a difficult problem as it raises numerous moral questions. Some claim it is unethical to put a price on human life because life is priceless. The actual expenditures on risk reducing prospects show however that the investment in the reduction of risks to humans is always finite. Different approaches are available for evaluating the costs of interventions in relation to the reduction of risk to human life, see Vrijling and Gelder (2000); Jongejan *et al.* (2005) for a further discussion of the various approaches. One of the options is to add the economic value of human fatalities to the economic damages, i.e. $D + N \cdot d$. The value of the number of lives lost can be determined with different approaches.

Several approaches are based on so-called stated preferences. A survey can reveal how much people are willing to pay, e.g. for safety measures. In these cases the value of statistical life (VoSL) is obtained from the willingness to pay expressed by respondents in surveys. For example, in the cost benefit analysis for the flood defences in the Netherlands a value of a statistical life of €6.7 million per fatality is used (Jeuken *et al.*, 2013). The Value of a Statistical life lost in traffic accidents is estimated at €2.6 million (SWOV, 2012).

One alternative approach is based on so-called revealed preferences. The costs of saving and extra (statistical) life ($CSX$) for actual life-saving interventions that have been taken in the past can be determined.

$$CSX = \frac{I}{\Delta E(N)}$$

where:

- $CSX$ the costs of saving an extra life [€/life year]
- $\Delta E(N)$ the reduction of the expected number of fatalities per year

An extensive study on values in various sectors, see Tengs *et al.* (1995) showed that these vary widely across sectors and even within sectors. The highest $CSX$ values are found for risks for small probability – large consequence events, for example in the nuclear domain. For such cases the expected number of fatalities is already small and investments in incremental safety are large.

One other approach is to base the value of a human life on macro-economic indicators. Several metrics have been proposed that relate this value to a person's economic production. Given the difficulties associated with economic valuation of human life and the associated risk reduction, it is decided in some domains to develop separate criteria for considering the risk to life. This topic is further elaborated in ⚠️ section 3.5.

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman. Find out more here.

# Economic Optimization

> **Note**
>
> It is important to read and understand the content on this page, however, it does *not* need to be studied for the exam.

The previous sections have focussed on decisions for which the number of actions was limited, e.g. excavation with or without drainage and the associated costs and benefits. However, there are several situations in which the number of actions is unlimited. This occurs when the failure probability level has to be decided for a system that is yet to be designed, with an infinite number of design options. An example of this type of decision problem is the heightening of dikes, as in theory an unlimited amount of values can be chosen for the elevation, e.g. 2m, 5m, 5.1m, 5.11m, 5.1111m 6m etc.

For this situation an economic optimization that takes into account the costs of increasing the safety level and reducing the risks can be applied to derive an optimal level of safety (or the optimal “failure probability”). The economic optimization was developed and applied by van Dantzig (1956), to derive the optimal dike height for South Holland after the 1953 storm surge disaster, as will be further elaborated in the next section. In the economic optimization the total costs ($C_{tot}$[€]) are determined, consisting of the investments $I$[€] in a safer system and the present value of the risk $R$[€].

$$C_{tot} = I + R$$

The annual risk, or expected economic damage is found by:

$$E(D) = P_f D$$

where:

- $E(D)$ the expected value of the risk [€/year]
- $P_f$ the failure probability of the system per year [1/year]
- $D$ the damage in case of failure [€]

In this approach it is thus assumed that all damages are expressed in monetary terms. Additional criteria for separately considering the loss of human life are included in the next section.

The present value of the risk for an infinite time horizon can be found as follows:

$$R = \frac{P_f D}{r}$$

The risk can be reduced by constructing a safer system (a lower $P_f$), or limiting the damage (smaller $D$). In this case we assume prevention measures that focus on reducing the failure probability. The investments will become a function of the failure probability of the system, since increasing the safety will lead to an increase of costs.

$$I = I(P_f)$$

Fig. 8.15 shows the costs and risks as a function of the accepted failure probability of a system. The economic optimum is found when the total costs are minimal.

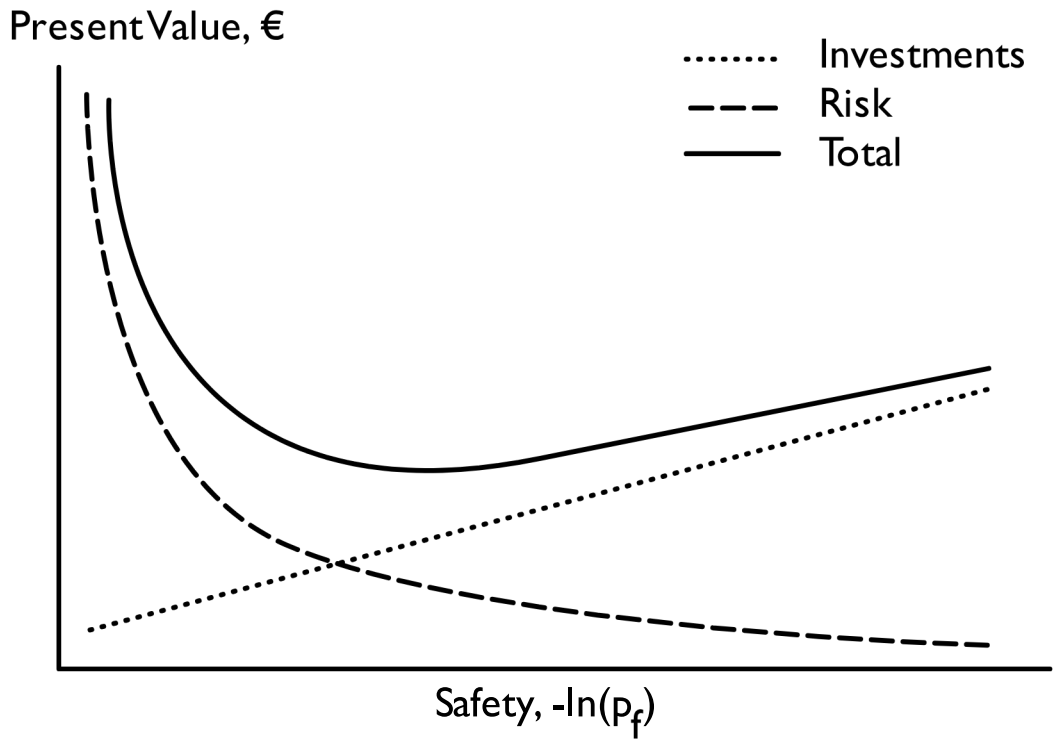



*Fig. 8.15* Economic optimization: costs, risks and total costs as a function of the failure probability of the system.

For this situation the following is valid:

$$\frac{dC_{tot}}{dP_f} = 0$$

This approach can be applied to various decisions problems, such as the optimal dike height (see next section) but also the dimensioning of other interventions such as sprinklers or ventilation in tunnels to reduce fire risks. In cases where no continuous functions are available to create a curve as in Fig. 8.15, the analyst can consider a limited number of design options (e.g. no, small, medium or large sprinklers in a tunnel) and determine investments, risks and total costs for these options.

In addition to the determination of the optimal failure probability, the cost benefit criterion should still be verified. (It is possible that we find an economic optimum with higher total costs than in the current situation without interventions). It is checked whether the benefits of risk reduction are larger than the costs of the dike heightening, i.e. $(P_{f,0} - P_{f,opt})D > I$.

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman. Find out more here.

# Economic Optimization Example

## Contents


- Probability and Risk
- Costs and Benefits
- Verification
- Discussion


Economic optimization is illustrated through the determination of an optimal dike height to protect against flooding.

Before the major floods of 1953, dikes in the Netherlands were not designed for a specified safety level but mainly strengthened based on practical experience. One of the main questions after the disaster was the optimal dike height and the “acceptable” probability of flooding. Van Dantzig was a professor in mathematics and a member of the first Delta committee. He developed an econometric approach to determine the optimal probability of flooding (or protection level) and the corresponding dike height (van Dantzig, 1956).

## Probability and Risk

The approach only considers failure of the dikes due to overtopping. The probability distribution of water levels along the Dutch coast can be approximated by an exponential distribution.

$$F(h) = 1 - e^{(-\frac{h-A}{B})} \tag{8.5}$$

In which:

- $h$ the water level [m]
- $A, B$ constants of the exponential distribution [m]

Neglecting wave run-up, the probability of failure of the dikes - leading to flooding - can be approximated by the probability of exceedance of the dike height $h_d$, i.e.

$$P_f = P(h > h_d) = e^{(-\frac{h_d - A}{B})} \tag{8.6}$$

It is assumed that there is total damage to all objects and infrastructures in the flooded area if the dikes fail. For the discount rate Van Dantzig applied a reduced interest rate $r'$ = economic growth – inflation. The net present value of the expected damages (i.e., the risk) is found as follows, assuming an infinite time horizon:

$$R = \frac{P_f D}{r'} = \frac{e^{-\frac{h_d - A}{B}} D}{r'} \tag{8.7}$$

# Costs and Benefits

As the risk is dependent on height, the dikes can be heightened further to limit the failure probability and thus risks. Investments in dike heightening are determined by initial costs for mobilization and variable costs that are dependent on the level of dike heightening (see also Fig. 8.16):

$$I = I_0 + I(h_d - h0) \tag{8.8}$$

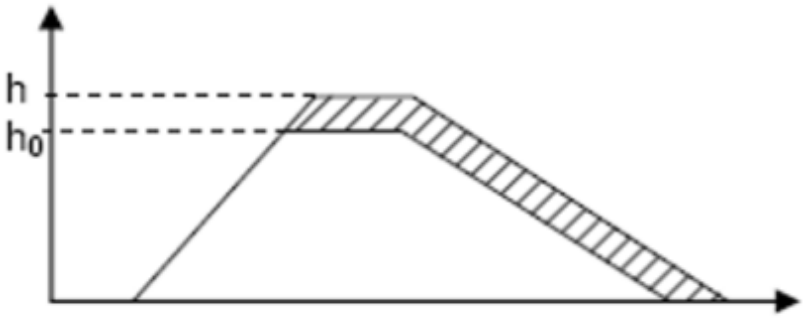



*Fig. 8.16* Schematic view of dike heightening

Now, the total costs $C_{tot}$ are the sum of the investment costs and the risk.

$$C_{tot} = I_0 + I(h_d - h_0) + \frac{e^{-\frac{h_d - A}{B}} D}{r'} \tag{8.9}$$

The optimal dike height is found when the total costs are minimal. This is the case when $dC_{tot}/dh_d = 0$.

$$\frac{dC_{tot}}{dh_d} = I - \frac{e^{-\frac{h_d - A}{B}} D}{Br'} = 0 \tag{8.10}$$

Since the $P_f = e^{-(h_d - A)/B}$, we can find the optimal flooding probability $P_{f,opt}$:

$$P_{f,opt} = \frac{IBr'}{D} \tag{8.11}$$

The larger the damage $D$, the smaller the optimal failure probability and thus the higher the level of protection. If incremental protection is expensive – i.e. for high values of $I$ – a higher optimal failure probability and thus a lower level of optimal safety will be found. The optimal failure probability is also dependent on the discount rate. Combination with (8.6) gives the optimal dike height $h_{d,opt}$:

$$h_{d,opt} = A - B \cdot \ln(P_{f,opt}) \tag{8.12}$$

# Verification

In addition to the determination of the optimal dike height the cost benefit criterion should still be verified by checking whether the benefits of risk reduction are larger than the costs of dike heightening.

$$I(h_{d,opt}) < \frac{(P_{f,0} - P_{f,opt})D}{r'} \tag{8.13}$$

Where $P_{f,0}$ the flooding probability in the initial situation. It can be easily shown that this is equivalent to a check of whether the total costs in the optimal situation are smaller than those in the original situation. The first Delta committee used the following values for South Holland:

- $D$ = fl 24.2·$10^9$ [unit is Dutch guilders]
- $r'$ = 0.015
- $I$ = 40.1·$10^6$ fl/m
- $A$ = 1.96 m
- $B$ = 0.33 m

- $h_0$ = 3.25 m
- $I_0$ = fl $110 \cdot 10^6$

Using these values, the following optimal dike height and optimal failure probability were derived:

- $h_{d,opt}$ = 5.83 m
- $P_{f,opt}$ = $8 \cdot 10^{-6}$ per year

# Discussion

Although the optimal safety level was determined at a failure probability of $P_{f,opt}$ = 1/125,000 per year, in later political discussions a value of 1/10,000 per year was selected for the probability of exceedance of design water levels. This implied that the dikes of South Holland would need to be designed for hydraulic conditions (water levels and waves) with a probability of exceedance of—on average—1/10,000 per year. In later decision-making, safety standards were derived for other dike rings (see Fig. 8.17), for example, flood defenses in the river system are designed for a safety standard of 1/1250 per year.

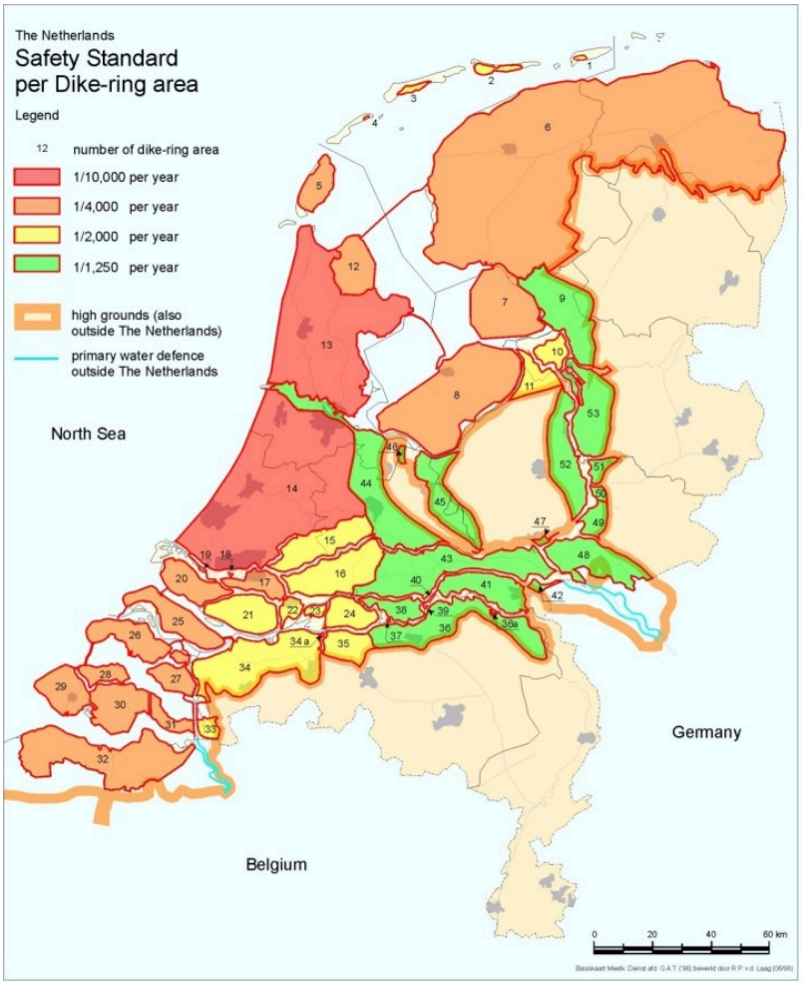


*Fig. 8.17* Dike Rings in the Netherlands, showing probability of exceedance of the design water levels per dike ring. Note that dike rings along the River Meuse are not shown on this map. Most of these dike rings have a safety standard of 1/250 per year. This map defined design levels until the safety standards changed in 2017.

It was expected that the actual failure probability for dikes designed for this design load, would be smaller than 1/10,000 per year. Recent risk analysis in the project VNK have shown that this is not the case due to the geotechnical failure mechanisms. For most dike rings the estimated failure probabilities are larger than the probability of exceeding the design loads. For example, for riverine dike rings that have been designed for design levels with a probability of exceedance of 1/1250 per year, failure probabilities are often in the order of magnitude of 1/100 per year (Veiligheid Nederland in Kaart, 2014). This is part of the reason for revising the safety standards and flood defence assessment criteria in the Netherlands, which since 2017 is based on an allowable probability of flooding rather than a load-based exceedance probability approach.

Historic discussions and developments in the Netherlands illustrate how the risk analaysis and risk management for a given system can change over time. Several extensions and additions to the van Dantzig model have been proposed over the years. For example, the inclusion of sea level rise (Vrijling and van Beurden, 1990), modelling of the damage as dependent on the water depth in the polder (van Dantzig, 1956), the inclusion of the economic value of loss of life as part of the damage, and inclusion of risk aversion by giving quadratic weight to the damages (Gelder *et al.*, 1997).

The model by van Dantzig was primarily focused on finding what the optimal safety at that time (when van Dantzig published his model in the 1960's) should have been. However, a single optimal level is not always the best solution. While considering longer timescales and changing conditions, such as economic growth and sea level rise, the model needs to address the possibility of multiple interventions during the period considered. This leads to questions regarding the timing of interventions (when? At which intervals?), as well as the optimal strengthening or raising of the dikes (how much?). Eijgenraam (2006) developed an economic decision model that takes into account both questions.

Fig. 8.18 below depicts the "saw-tooth" curve that shows the periodic interventions: both the extent of the intervention (vertical) as well as the timing between interventions (horizontal). In between interventions, the safety will gradually decrease due to sea level rise and / or subsidence of the dike. Over a longer period of time, the dike heightening (or strengthening) should follow sea level rise. Additional dike heightening (lower the failure probability even further) could be considered in case of economic growth, which will lead to an increase of damages and increase of protection standards. The length of the optimal interval between

interventions is largely dependent on the initial or mobilization costs. If these are high, for example in the case of structural interventions such as storm surge barriers, a long interval or life time of almost 100 years can be chosen. For regular dike reinforcements this interval will be more in the range of several decades (e.g. 50 years). For interventions with no or very small initial costs, such as nourishments along the coast, it is optimal to intervene more frequently.

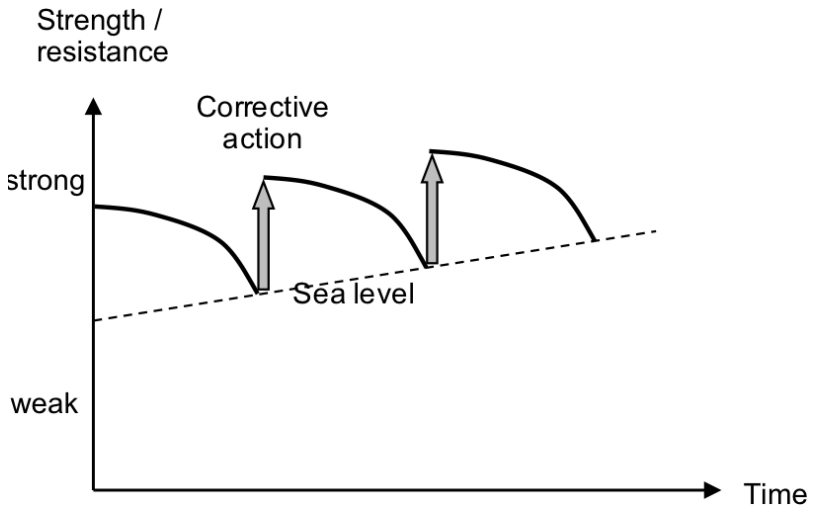



*Fig. 8.18* Periodic investments in dike reinforcement for a situation with sea level rise.

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman. Find out more here.

# Safety Standards

## Contents


- Three Types of Risk
- Limits for Individual and Societal Risk
- Case Study: Industrial Hazards
- Case Study: Flood Protection


> **Note**
>
> This note has been updated on Friday, January 17, 2024. It previously said that you did *not* need to know the content on this page, however, it was not true (and conflicted with the front page of this chapter). While you don't need to memorize everything, you are expected to understand the concept of a limit line and how to interpret it (i.e., section *Limits for Individual and Societal Risk*). If you are unsure of what this means, review WS 2.8, which will illustrate the scope that you are expected to understand.

When answering the question "how safe is safe enough" a merely economic treatment with cost benefit analysis or economic optimization is often not sufficient for activities with risks to people. Therefore, criteria have been developed that focus on risks to human life. This section focuses on safety standards and criteria for evaluating the risk to life.

Two aspects are typically considered when evaluating and regulating risks to the public: the total or population-wide effects, and the distribution of effects within the affected population. Table 8.5 summarizes these perspectives. The societal perspective is concerned with 'total effect' and the effects of large-scale accidents on the society, in terms of economic damages and life loss. The individual perspective is concerned with distributive justice ('equity'), i.e. the distribution of harm over the population.

As risk is often the by-product of an otherwise legitimate and advantageous activity, such as production or transportation, regulating risks is essentially a balancing act between economic and social activities on the hand and a sufficiently safe society on the other hand

(e.g. Jongejan (2008)): just as too lenient regulations are suboptimal, too stringent ones are too.

*Table 8.5* Overview of perspectives on risk evaluation

| Aspect | Relevant risk metric | Rationale |
|---|---|---|
| Equity ('distributive justice') | Individual risk | Regulation is to prevent that individuals are exposed to disproportionally large risks |
| Total effect | Societal and economic risk | Regulation is to limit large-scale accidents and optimize the costs and benefits of risk reduction |

# Three Types of Risk

Based on the general concepts described above, it has been proposed to evaluate risks based on three criteria ((TAW, 1985, Vrijling *et al.*, 1995, Vrijling *et al.*, 1998)):

- limit the **invidual risk** to prevent that certain people are exposed to disproportionally large risks;
- limit the **societal risk** to limit the risks of large scale accidents with many fatalities;
- **Economic optimization** to balance investments in risk reduction from an economic point of view.

## Societal and Economic Risk

Two of these criteria have already been discussed elsewhere in this book. **Societal risk** refers to the probability of an accident with multiple fatalities and is often graphically represented by an FN-curve (Risk Curve Section). **Economic optimization** has already been discussed in the Economic Optimization Section.

## Individual Risk

**Individual risk** concerns the probability of death of a person due to an accident. Various related definitions exist. The "average individual risk" for a certain activity can be calculated,

e.g. the individual risk due to smoking or the risk of an airplane crash for a frequent flyer. Table 8.6 compares the probability of various types of fatal accidents.

*Table 8.6* Average probability of death due to several activities. (statistics based on various sources for the Netherlands and Europe)

| **Activity** | **Probability (per year)** |
|---|---|
| Mountain climbing | $10^{-2}$ |
| Traffic (young men) | $10^{-4}$ |
| Accidents at home | $10^{-4}$ |
| Structural failures | $10^{-7}$ |

The individual risk due to an accident can be calculated with:

$$IR = P_f P_{d|f}$$

where:

- $IR$ the individual risk [1/year]
- $P_f$ the probability of an accident [1/year]
- $P_{d|f}$ the conditional probability of death given an accident

Both the individual risk and $P_{d|f}$ can be formulated as a function of a location near a risk source. If it is assumed that a person is permanently present at that location, the individual risk becomes a property of that location and it can be used for zoning and land use planning. This is applied in the industrial safety policy in the Netherlands (see next section).

**Example: Difference between individual and societal risk.**

To illustrate the difference between the individual and societal perspective, consider a fairly safe car that has a probability of $10^{-5}$ per year of causing a fatality due to technical failure. This may well be acceptable to an individual. Such a probability is in the same range as the average death rate in traffic (650 fatalities / 17 million people which is about $4.10^{-5}$ per year). If only 1000 cars are sold in the Netherlands, the number of fatal accidents with this car is rare, i.e. 0.01 fatalities per year. However, when the car becomes popular and 10 million cars are sold, the average number of fatalities due to technical failure becomes 100 per year. This may well lead to public concerns and indignation. From a societal point of view this may no longer be acceptable, whereas the individual risk has not changed.

# Limits for Individual and Societal Risk

Safety standards can be used to set limits to the individual and societal risk. A **limit** value can be set for **individual risks**. Such a limit value is to avoid disproportionate exposures by laying down a minimum safety level for all individuals. In various fields of applications limit values in the range of $10^{-4}$ to $10^{-6}$ per year are used (see below for more information). To put the stringency of the individual risk criteria into perspective, one could consider the effect of the probability of an accident on life expectancy. When an average person would be constantly exposed to a maximum allowable level of risk of 10-6 per year, the decrease of his or her life expectancy would be only 1 day as shown in Table 8.7.

*Table 8.7* Relationship between the additional risk and life expectancy

| Additional probability of death in a year | Life expectancy in years | Decrease of life expectancy in years | Decrease of life expectancy in days |
|---|---|---|---|
| $10^{-3}$ | 74.97 | 3.16 | 1153 |
| $10^{-4}$ | 77.81 | 0.32 | 117 |
| $10^{-5}$ | 78.10 | 0.03 | 11 |
| $10^{-6}$ | 78.13 | 0.00 | 1 |
| $10^{-7}$ | 78.13 | 0.00 | 0 |

Societal risk can be evaluated by means of an FN limit line. The calculated FN curve of the system should, in principle, not exceed the limit line. An FN-criterion is defined by three variables: (1) its base point (the exceedance probability of 1 fatality, i.e. C), (2) its slope ($\alpha$), and (3) its probability and/or consequence cut-off ($A$ and $B$). Fig. 8.19 shows the different constraints that could make up an FN limit line.

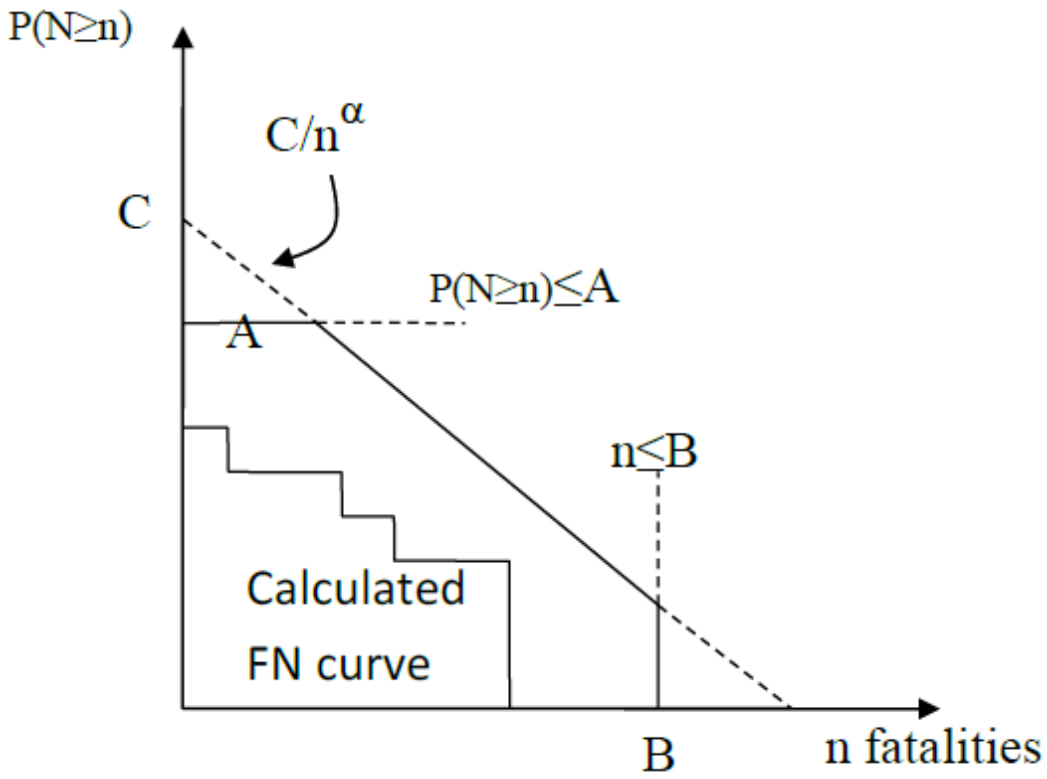



*Fig. 8.19* Schematic FN curve illustrating limit line formulation.

The general formulation for such a limit line without horizontal of vertical cut-off equals:

$$1 - F_N(n) \leq \frac{C}{n^\alpha} \tag{8.14}$$

where:

- $C$ a constant that determines the vertical position of the limit line
- $\alpha$ a coefficient that determines the steepness of the limit line

The limit line is called risk neutral[1] if $\alpha = 1$, since it places equal weight on exceedance probabilities and numbers of fatalities. If $\alpha = 2$, the limit is risk averse. This means that that the exceedance probability of 10 times as many fatalities should be 100 times lower. This has been motivated by public aversion to large numbers of fatalities. For example, the loss of 1000 people in one accident (e.g. a major explosion) could be valued differently than 1000 losses of 1 person in separate accidents (e.g. in traffic).

For different applications limit lines have been developed with varying constants and steepness. Examples of application areas include industrial risks in the Netherlands (next section), dams in the United States and Canada, and chemical risks in Hongkong and the UK (Jonkman *et al.*, 2003).

**Exaple: Risk Matrix**

Risk matrices are often used in various industries for risk evaluation and decision support, for example, to quickly prioritize actions, especially in time-sensitive situations. For a given undesired event the extent of probability and consequences are estimated on a qualitative or semi-quantitative scale, see Fig. 8.20 for an example. Ranges of failure probabilities or consequences can be assigned to the qualitative terms on the axes in the example. The combination of probability and consequence determines the level of risk, and depending on the application, whether it is acceptable or whether it requires more attention and risk reduction efforts. However, unlike the FN curve, the cumulative effects of multiple possible events are generally not considered in a risk matrix.

| E = Extremely High Risk<br>H = High Risk<br>M = Moderate Risk<br>L = Low Risk | | | PROBABILITY | | | | |
|---|---|---|---|---|---|---|---|
| | | | Frequent | Likely | Occasional | Seldom | Unlikely |
| | | | A | B | C | D | E |
| SEVERITY | CATASTROPHIC | I | E | E | H | H | M |
| | CRITICAL | II | E | H | H | M | L |
| | MARGINAL | III | H | M | M | L | L |
| | NEGLIGIBLE | IV | M | L | L | L | L |

*Fig. 8.20* Example of a risk matrix used by various engineering and military branches of the United States Army Corps of Engineers (Source: Corps Risk Management Center or Risk Analysis Gateway).

# Case Study: Industrial Hazards

The Dutch major hazards policy deals with the risks to those living in the vicinity of major industrial hazards such as chemical plants and LPG-fuelling stations. The development of the Dutch major hazards policy was strongly incident driven, as were European efforts aimed at the prevention of major industrial accidents. After a number of severe industrial accidents, including the Bhopal accident in 1984 which killed an estimated 3000 people and severely injured over 200.000, a European directive was drafted concerning the prevention of major accidents: the 1982 Seveso Directive. This was later replaced by the Seveso II Directive. The cornerstones of the Dutch major hazards policy are a) the use of quantitative risk analysis (QRA); b) comparison of QRA outcomes with limits to individual and societal risks (Bottelberghs, 2000).

Within the Dutch major hazards policy, individual risk is defined as the probability of death of an average, unprotected person that is constantly present at a certain location. It is thereby a property of location and iso-risk contours can be plotted on a map (see Fig. 8.21). Individual risk is therefore also named local risk (“plaatsgebonden risico”) in the Netherlands. The shape of the risk contours for other applications will look different. For airports the contours will follow the shape of the runway and flight paths, for polders and flooding the risk contours will be highest in the deepest part of the polder.

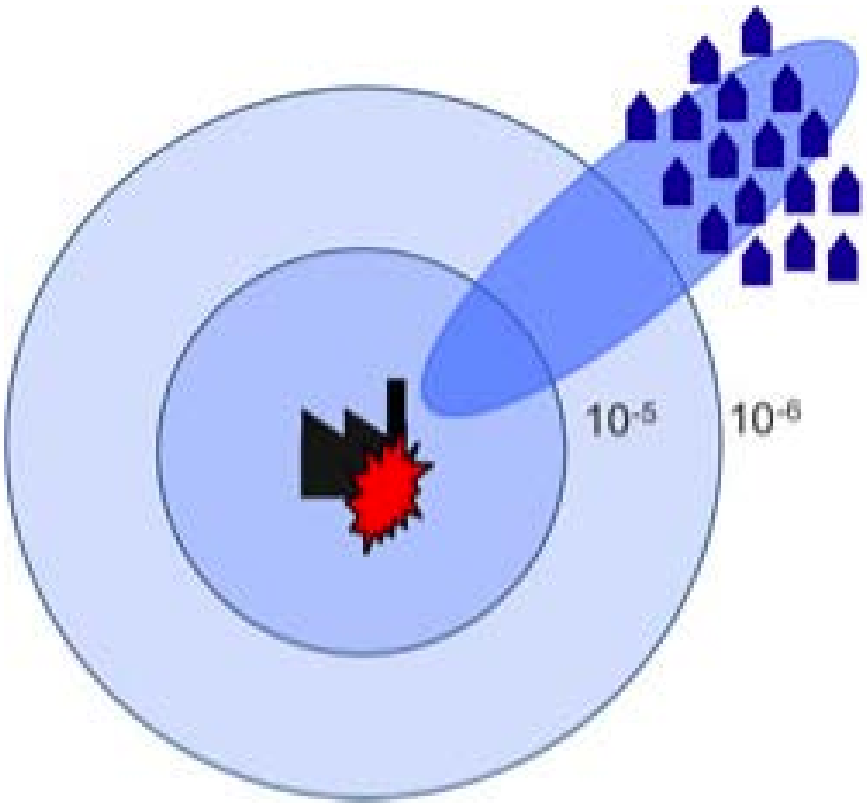



*Fig. 8.21* Example of a schematic individual risk contour (circles) for a chemical installation.

*Table 8.8* Individual risk criteria used in the Netherlands.

| | | **Individual (or local) risk criterion** |
|---|---|---|
| **Existing situations** | Vulnerable | $10^{-6}$ per year |
| | Limitedly vulnerable | $10^{-5}$ per year, strive for $10^{-6}$ per year |
| **New situations** | Vulnerable | $10^{-6}$ per year |
| | Limitedly Vulnerable | $10^{-6}$ per year |

A distinction is made between vulnerable objects such as schools and houses and limitedly vulnerable objects such as small offices. The following criteria apply. For new situations a limit of $10^{-6}$ per year applies. A comparison with Table 8.8 shows that this individual risk value has a negligible effect on life expectancy.

The criterion for societal risk that is used in the Netherlands for evaluating the third party risks posed by major industrial hazards is shown in Fig. 8.22 below. It serves as a reference in the broader assessment of third party risks by competent authorities. Exceedances of the criterion line also have to be motivated by competent authorities. When the criterion line is not exceeded, the acceptability of the third party risk still has to be motivated. The limit line is characterized by $C = 10^{-3}$ and a steepness of $\alpha = 2$, making it a risk averse criterion. The criterion is used to assess the acceptability of the risks of individual facilities.

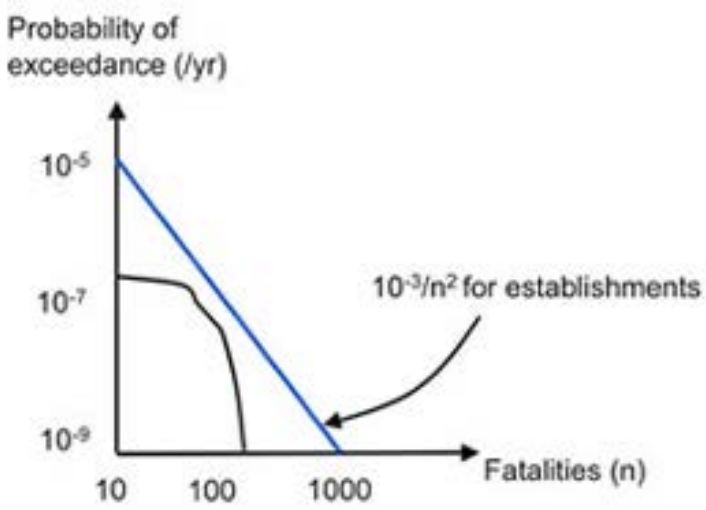



*Fig. 8.22* FN limit line used for installations in the Netherlands

# Case Study: Flood Protection

For systems for which no regulations are available the question ‘how safe is safe enough?’ can be difficult to resolve. A general model has been developed by the Technical Advice Committee for Water defences (TAW)[2] in the process of deriving safety standards for flood protection in the Netherlands, which, in a formal sense, has been an ongoing process since the 1950’s. This approach is derived on the basis of accident statistics (TAW, 1985, Vrijling *et al.*, 1995, Vrijling *et al.*, 1998), where underlying assumption of the model is that accident statistics are the result of a process of societal optimisation and that they thereby reflect what is apparently considered acceptable by society at large. Such an approach is commonly referred to as a ‘revealed preference’ approach, and was already introduced in a a general sense in the Chapter on Risk Analysis, Section A Famous Risk Curve (Fig. 8.2).

# Individual risk

Accident statistics reveal that the extent to which participation in the activity is voluntary strongly influences the level of risk that is accepted by individuals. Relatively high individual risks are accepted for activities that are voluntary and have a (personal) benefit, such as mountain climbing. Much smaller individual risk values are accepted for involuntary activities for which the risks are imposed by others, e.g. for chemical and nuclear industry. A policy factor ($\beta$) is therefore introduced to account for voluntariness of exposure. This factor is set at $\beta = 1$ for an individual risk value of $10^{-4}$ per year. This represents the “baseline” individual risk for the group young men[3] who are most at risk of dying in a traffic accident.

*Table 8.9* Accident statistics and proposed policy factor and characteristics of the activity (Vrijling, 2001, Vrijling *et al.*, 1998).

| **Prob. of death(per year)** | **Example/application** | $\beta$ | **Voluntariness** | **Benefit** |
|---|---|---|---|---|
| $10^{-2}$ | Mountain climbing | 100 | Voluntary | Direct benefit |
| $10^{-3}$ | Driving a motor cycle | 10 | Voluntary | Direct benefit |
| $10^{-4}$ | Driving a car | 1 | Neutral | Direct benefit |
| $10^{-5}$ | Flooding | 0.1 | Involuntary | Some benefit |
| $10^{-6}$ | Factory/LPG station | 0.01 | Involuntary | No benefit |

The proposed individual risk limit becomes:

$$IR < \beta \cdot 10^{-4} \tag{8.15}$$

An appropriate value for the policy factor $\beta$ can be chosen depending on the characteristics of the activity. If the conditional probability of death due to an accident $P(d|f)$ is known, the acceptable failure probability can be computed:

$$P_f \leq \frac{\beta \cdot 10^{-4}}{P_{d|f}} \tag{8.16}$$

Note that the Dutch individual risk criterion for hazardous installations would be obtained for $\beta = 0.01$ and a conditional probability of death of 1 ($P_{d|f} = 1$).

# Societal risk

The societal risk criterion proposed by the TAW is based on the thought that societal risk should be evaluated primarily at a national level as local developments may lead to a situation that is considered unacceptable by society as a whole (Vrijling *et al.*, 1995). The societal risk criterion at a national scale proposed by the TAW is:

$$E(N) + k\sigma(N) \leq \beta \cdot 100 \tag{8.17}$$

where:

- $E(N)$ the expected number of fatalities per year
- $k$ the risk aversion index (proposed value, $k = 3$)
- $\sigma(N)$ the standard deviation of the number of fatalities per year
- $\beta$ the policy factor

A risk aversion index $k$ has been introduced to account for risk aversion. For accidents with small probabilities and large consequences the standard deviation $\sigma(N)$ is large relative to $E(N)$, see example below. The total risk takes a risk aversion index $k$ [-] into account. For $k > 1$, the “cost of risk bearing” exceed expected loss, implying a risk averse attitude.

**Expected value and standard deviation for two systems**

We consider two systems

1. This system has a high failure probability of 0.01 per year and 1 fatality
2. The second system has a smaller failure probability of 0.0001 per year but a higher number of 100 fatalities.

For both systems a binomial distribution of the number of fatalities is applied meaning that the number of fatalities in case of failure is exactly known. The expected value and standard deviation of the number of fatalities are found as follows:

$$E(N) = P_f N \quad \sigma^2(N) = P_f(1 - P_f)N \tag{8.18}$$

The resulting expected value and standard deviation are shown in Table 8.10 below. Although both systems have the same expected value, the standard deviation for the "small probability – large consequence" event for system 2 is higher. Taking into account the standard deviation in the TAW criterion thus accounts for risk aversion against accidents with large numbers of fatalities.

*Table 8.10* Calculation of expected value and standard deviation for two activities.

| | $P_f$ | $N$ | $E(N)$ | $\sigma(N)$ |
|---|---|---|---|---|
| 1 | $10^{-2}$ | 1 | $10^{-2}$ | 0.0995 |
| 2 | $10^{-4}$ | 100 | $10^{-2}$ | 0.1000 |

The next step would be to distribute this maximum allowable level of societal risk over individual installations. After all, locally imposed societal risk criteria are necessary for achieving the desired national level of societal risk. The translation of the nationally acceptable level of risk to a criterion for a single local installation depends on the type of probability distribution of the number of fatalities. In Vrijling *et al.* (1998) a formulation of the risk acceptance at a local level is presented conform (8.14):

$$1 - F_n(n) \leq \frac{C}{n^\alpha}$$

For a binomial distribution this yields:

$$C = \left( \frac{\beta \cdot 100}{k\sqrt{N_a}} \right)^2$$

in which $N_a$ is the number of independent locations where the activity takes place. This requirement corresponds to the requirement set for the chemical installations if $\beta = 0.03$, $N_a = 1000$ and $k = 3$.

# Combination of Risk Types

According to the approach by TAW the three approaches (individual, societal and economic risk) lead to three acceptable failure probabilities. The most stringent of the three criteria can be chosen to determine the acceptable probability of failure of the system and to make sure that all three conditions are fulfilled. This can best be illustrated with an example (see below). The principles of this approach have been applied to derive the proposed new safety standards for flood defences in the Netherland by the Deltaprogramma (2014).

**Example: Combination of individual, societal and economic risk for a dike ring area**

We consider application of the three criteria to the case of dike rings in the Netherlands. There are about 100 dike rings in the Netherlands of different sizes. A dike ring is a flood prone area protected from flooding by flood defences and high grounds. The conditional probability of death given flooding depends on the depth of the polder and flood characteristics. Research on loss of life due to floods shows that a conservative estimate would lead to $P_{d|f} = 0.1$.

First, we consider the individual risk. A value of $\beta = 0.1$ is proposed as being exposed is considered as an involuntary activity with some benefit (i.e. living in a prosperous delta). This leads to an acceptable individual risk value of $10^{-5}$ per year. This limit has also been proposed by the Dutch government in the year 2014 ("basisveiligheid"). The acceptable flooding probability according to the individual risk becomes:

$$P_f \leq \frac{\beta \cdot 10^{-4}}{P_{d|f}} = \frac{0.1 \cdot 10^{-4}}{0.1} = 10^{-4} \text{ per year} \tag{8.19}$$

The societal risk criterion can be determined according to (8.14). Assuming a risk averse criterion $\alpha = 2$. We can determine the constant $C$ of the limit line for $N_a = 100$ installations and $\beta = 0.1$.

$$C = \left( \frac{\beta \cdot 100}{k\sqrt{N_a}} \right)^2 = \left( \frac{0.1 \cdot 1000}{3\sqrt{100}} \right)^2 = 0.11$$

The limit line for societal risk becomes $1 - F_N(n) \leq 0.11/n^2$. Both the individual and societal risk criteria are plotted below. As a third criterion the economic optimization can be added. The optimal or acceptable probability of failure depends on the damage and investment costs. A relationship with the graph below can be established by assuming that the number of fatalities is related to the economic damages. A dike ring with many inhabitants and potential fatalities will generally also represent a large economic value. For the sake of the example we assume that every fatality corresponds to an economic damage of €$5 \cdot 10^7$ (note: this is not equal to the value of a human life). To calculate the economic optimum for the example we assume arbitrary values of $r = 0.025$ and €$5 \cdot 10^6$ $B = 0.33$. Fig. 8.23 below shows the combination for the three criteria.

For a given number of fatalities in a dike ring the acceptable failure probability according to the three criteria can be derived. The individual risk criterion is independent on the number of fatalities. The economic criterion shows a linear relation between the failure probability and damage or number of fatalities. The societal criterion is risk averse so shows a decreasing quadratic relationship between acceptable failure probability and consequences.

For a given number of inhabitants and potential fatalities for a dike ring, the acceptable failure probability can be determined. For dike rings with between 1 and 10 fatalities – generally small areas-the individual risk criterion is the most stringent. For dike rings with large numbers of fatalities, the risk averse societal risk criterion becomes dominant.

Several extensions of this model are possible. One can consider to add the economic value of life loss or consider a different distribution of the nationally acceptable societal risk over dike rings with different sizes.

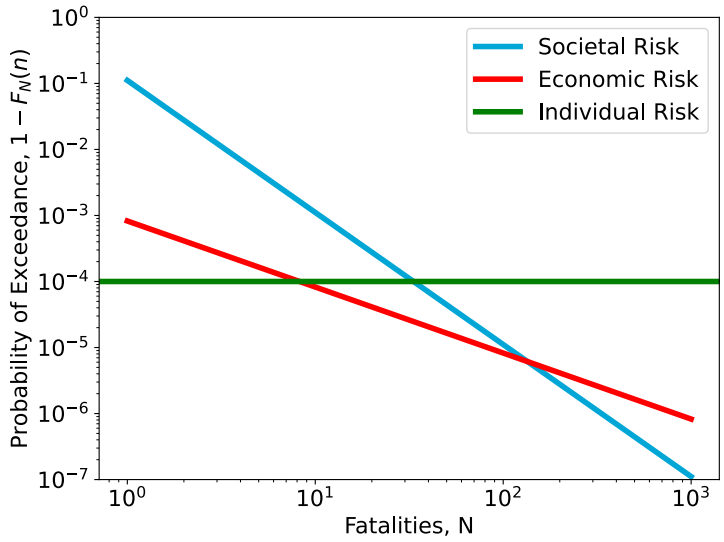



*Fig. 8.23* Combination of individual, societal and economic risk criteria for a hypothetical example.

One field of application where these concepts have been applied concerns the flood protection standards in the Netherlands. Since 2017 new safety standards have been in place for primary flood defences in the Netherlands that have been derived based on a risk assessment, with standards been formulated as a (tolerable) probability of failure of the flood defences.

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman. Find out more here.

---

It should be noted that the usage of the term “risk neutral” to describe the FN-curve limit line for $\alpha = 1$ is widespread but, strictly speaking, incorrect in the context of decision theory (see Decision Analysis Section). This is because the cost of risk bearing to a risk neutral decision maker equals expected loss (i.e. the product of probabilities and damages). The FN-curve shows cumulative probabilities, however. Also, an individual crossing of the limit line would not necessarily disturb a risk neutral decision maker, provided the other accident scenarios have relatively small probabilities.

TAW is nowadays called ENW: Extertise Network on Flood Protection.

The group ‘young men’ is used as a baseline because this was the only information available at the time the study was completed.

# Exercises

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

A number of exercises are included in this chapter that illustrate the previously discussed concepts of risk and reliability in a probabilistic design context. At the end a set of **sample exam** questions are included.

# FN Curve

A small town is located in a seismic region threatened by two faults. Fault A has a 10% probability of occurring each year, which would cause 10 fatalities, and Fault B has a 1% chance of occurring, which would cause 100 fatalities. Construct an FN curve for the town. Assume that the town is risk averse and that the tolerable risk limit is governed using a constant of C=1.

**Question 1:** Compute the expected annual fatalities for the town.

**Answer**

$$E(N) = P_1 \cdot N_1 + P_2 \cdot N_2 = 0.1 \cdot 10 + 0.01 \cdot 100 = 2$$

**Question 2:** If the town improved the safety of its buildings, what would be the maximum allowable number of fatalities that would be considered acceptable due to an earthquake on fault A?

**Answer**

Using the limit curve

$$1 - F_n(n) = C/n^{\alpha}$$

$$0.11 = 1/n^2$$

$$n = \sqrt{10} = 3.1 \approx 3$$

**Question 3:** What is the maximum value for the probability of an earthquake on Fault B that would be considered acceptable by the town?

**Answer**

Using the limit curve

$$1 - F_n(n) = C/n^{\alpha}$$

$$1 - F_n(100) = C/100^2 = 0.0001 = 10^{-4}$$

**Question 4:** Create an FN curve for this situation, including the limit line. Be sure to clearly label the axes and the values of key points on your diagram.

**Note**

If you are not sure what the limit line is, wait for this question until you read the Safety Standards Section (Risk Evaluation chapter).

**Answer**

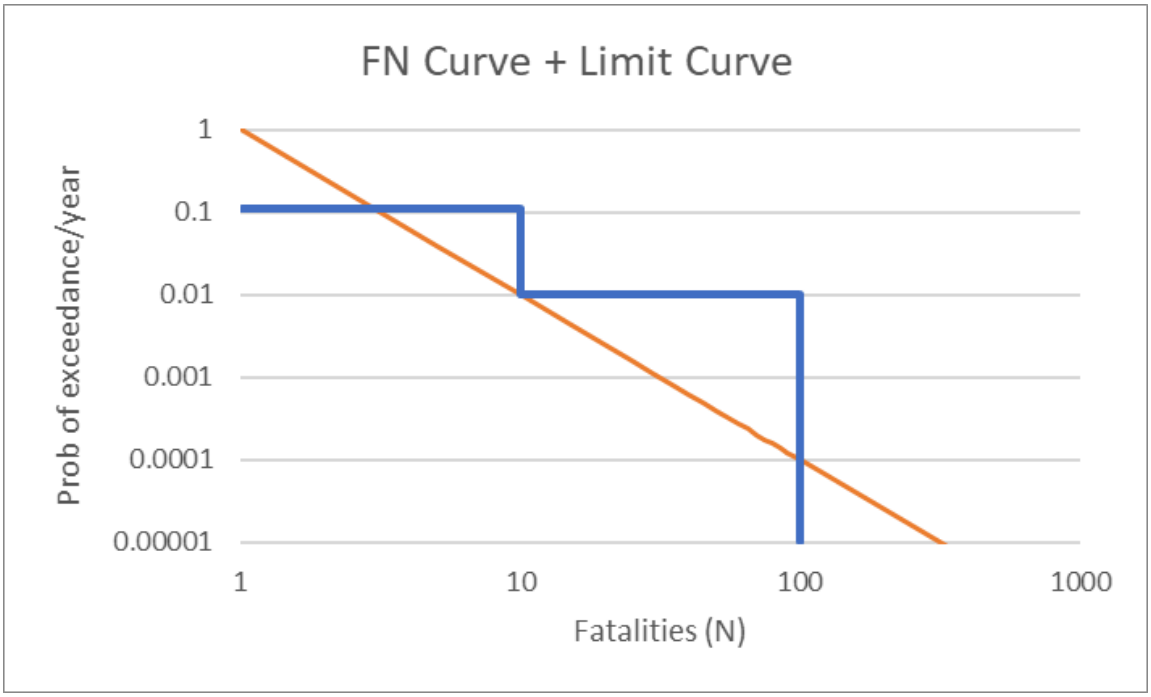



*Fig. 8.24* FN-curve for exercise 4

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Dam and River

## Contents


- Dam repair: probabilistic planning
- Acceptable risk


> **Note**
>
> Only Questions 3 and 4 of this exercise are relevant for the exam material in this chapter. Note, however, that Questions 1 and 2 should be review of material from previous chapters.

You are tasked to analyse the safety of a dam and the downstream river system. Downstream of the dam there is a dike ring protected by two main dike sections which must be able to contain water released from the reservoir. The dike sections are connected to each other and form a continuous boundary along the same side of the river. Let $P(F_1) = 0.01$ and $P(F_2) = 0.01$ denote failure probability of dike section 1 and section 2 respectively. There appears to be a correlation between failures of both sections, $\rho_{1,2} = 0.9$, and the figure below shows the effect of the correlation coefficient on the joint failure of the two sections.

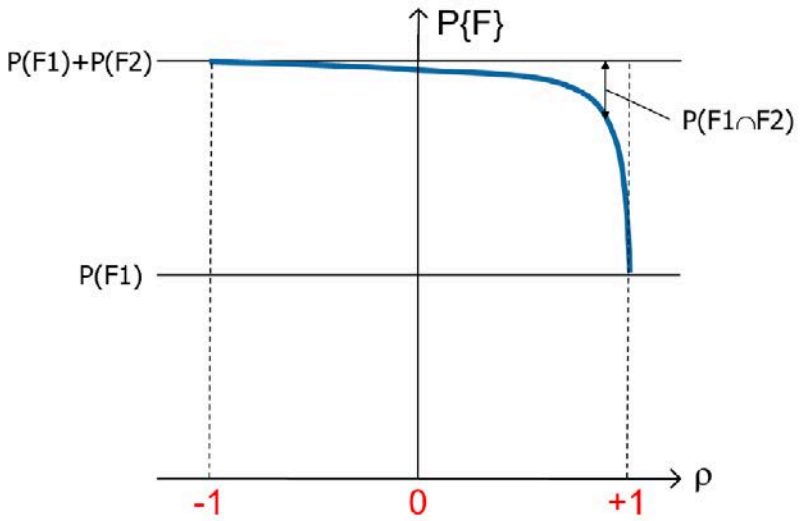



*Fig. 8.25* Effect of correlation coefficient joint failure, $P(A \cap B)$ (assume that $P(A \cap B) = 0.003$ at $\rho = +0.9$, where the arrow is located).

**Question 1:** Estimate the probability of flooding of the area protected by the two dike sections. Assume that the failure probability of each mechanism is 0.01, and they are *not* independent; however, you can use the figure above to quantify this.

**Answer**

Flooding of the protected area occurs when at least one of the two dike section fails. This is a series system. Thus, probability of flooding of the protected area can be determined by:

$$P(\text{flooding}) = P(\text{dike section 1 fails}) + P(\text{dike section 2 fails}) - P(\text{both dike sections fail})$$
$$P(F_1 \cup F_2) = P(F_1) + P(F_2) - P(F_1 \cap F_2)$$

Based on the figure and the correlation $\rho_{1,2} = 0.9$, the joint probability of failure $P(F_1 \cap F_2) = 0.003$ and the series failure probability is

$$P(\text{flooding}) = 0.01 + 0.01 - 0.003 = 0.017 \ /\text{year}$$

# Dam repair: probabilistic planning

The dam has to be repaired. The responsible minister wants to make sure the whole project is finished within 12 months, and only accepts a probability of 10% that actual construction works exceed this duration. The dam reinforcement consists of two activities. The second activity can only start when the first one has been finished. The durations of both activities are uncertain and normally distributed. The first activity has a mean duration of $\mu_1 = 4$ months and a standard deviation of $\sigma_1 = 1$ month; for the second activity $\sigma_2 = 2$ months.

**Question 2:** Compute the mean duration of the activity 2, so that the project duration would fall within the criterion given by the minister.

**Answer**

Let $T_1$ and $T_2$ denote the duration of activities 1 and 2; $T$ denotes the total duration; $T_1$ and $T_2$ are normally distributed:

$$T_1 \sim \mathcal{N}(\mu_1 = 4, \sigma_1 = 1)$$
$$T_2 \sim \mathcal{N}(\mu_2 = ?, \sigma_2 = 2)$$

$T = T1 + T2$ is also normally distributed, where

$$\mu_T = \mu_1 + \mu_2$$
$$\sigma_T = \sqrt{\sigma_1^2 + \sigma_2^2} = \sqrt{5} = 2.326$$

The probability of project duration greater than 12 months can be found with the complementary standard normal CDF, and should be a maximum of 0.1:

$$1 - \Phi\left[\frac{12 \text{ months} - \mu_T}{\sigma_T}\right] = 0.1$$

Applying the inverse CDF allows us to solve for $\mu_1$:

$$\frac{12 - \mu_T}{\sigma_T} = \Phi^{-1}[0.9]$$
$$\frac{12 - (\mu_1 + \mu_2)}{\sigma_T} = 1.2816$$

$$\mu_1 = 12 - 1.2816 \cdot \sigma_T - \mu_2$$
$$\mu_1 = 12 - 1.282(2.326) - 4 = 5.018 \text{ months}$$

# Acceptable risk

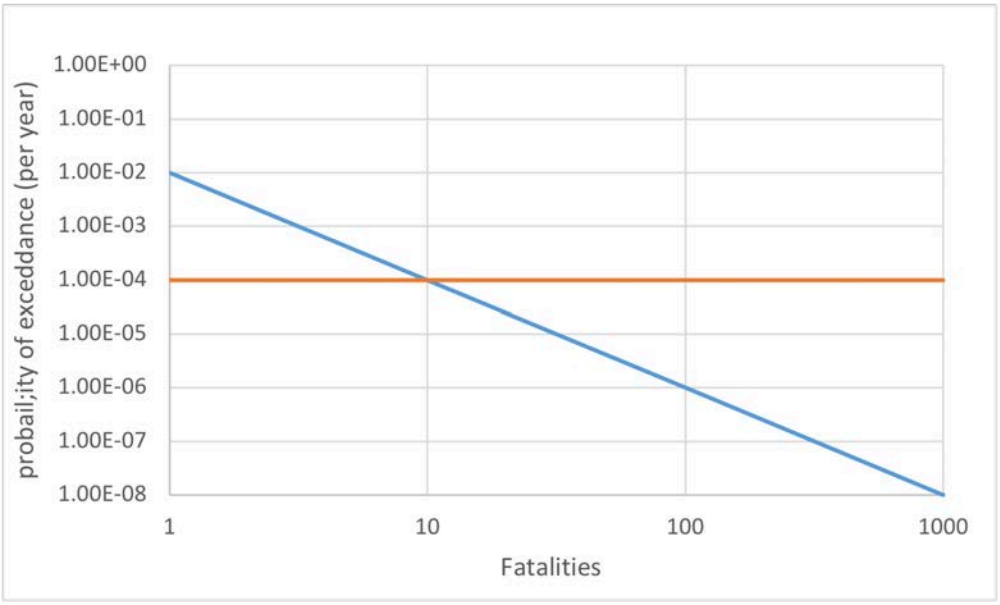
The government wants to study the acceptable risk and the acceptable probability of dam failure. Two criteria are used for acceptable risk:

- Societal risk (SR): an FN limit line: $P_f \leq 10^{-2}/n^2$.
- Individual risk (IR): $IR \leq 10^{-4}$ per year.

We know that mortality in case of dam failure is $F_{d|f} = 0.1$ and that the population living downstream of the dam is 1000 people.

**Question 3:** Plot the FN limit line and indicate the acceptable probability of failure according to both criteria (individual risk, societal risk).

**Answer**




**Question 4:** Provide an advice on the acceptable probability of failure for the dam. Give the proposed numerical value and a short motivation.

**Answer**

Based on individual risk criterion, we have:

$$IR = P_f P_{d|f} \leq IR_{acc}$$

where $P_{d|f} = 0.1$. Hence, the acceptable probability of failure is:

$$P_f \leq \frac{IR_{acc}}{P_{d|f}} = \frac{10^{-4}}{0.1} = 10^{-3} \ /\text{year}$$

Based on societal risk criterion (FN-Curve):

Acceptable failure probability can be determined by the FN-Curve:

$$P_f = 1 - F_N(n) \leq \frac{C}{n^{\alpha}} = \frac{10^{-2}}{n^2}$$

Based on given information above, expected number of casualties per dam failure event is $n = F_{d|f} \cdot 1000 = 100$.

$$P_f \leq \frac{10^{-2}}{100^2}$$
$$P_f = 10^{-6} \ /\text{year}$$

The most stringent of the two criteria applies, so the proposed safety standard would be $10^{-6}$ per year for the given inputs.

**Attribution**

This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Paint System

> **Note**
>
> This exercise is optional; it is provided to illustrate some of the concepts in the decision theory and economics-related pages of the Risk Evaluation chapter, which are not part of the exam material.

In this assignment we will consider a paint system that prevents rust and corrosion on a steel structure. A high quality paint system costs €40 per square meter and has a failure probability of 0.002, which assumes that the old paint system is completely stripped off and the steel surface is cleaned prior to installation, which has an additional cost of €20 per square meter.

Common cost cutting measures are to use a cheaper paint that costs half as much, but is five times more likely to fail, or to simply sand the old paint rather than stripping it off prior to applying a new paint, which increases the failure probability by 10, but only costs €5 per square meter.

During operation of the structure, if a rust spot is found it must be repaired immediately by removing the damaged paint and replacing it. The repair has an additional cost of €500 per square meter.

Construct a decision tree to evaluate the different options for steel preparation and paint, then use it to answer the following questions. Assume that the owner wants to minimize total costs, that the paint system only fails once in any reference period and that the probability of failure in each year is independent. All costs and probabilities are calculated on a square meter basis.

**Decision Tree**

This problem can be visualized using a decision tree as shown below. However, it results in a different interpretation of the problem than that shown in the solution. For example, consider the top branch at the far right, where 'No rust spot' is assigned a probability of 0.998. This is consistent with the formulation of a decision tree, but not consistent with the problem statement, which implies that the €60 cost associated with cleaning and painting is always done when replacing the paint system (thus a probability of 1.0). This is why the values in the decision tree are slightly different than those in Question 3, for example.

Although the decision tree is not part of the exam, this illustrates an important lesson: it is very easy to accidentally make probability calculations that are inconsistent with the situation you are trying to evaluate!

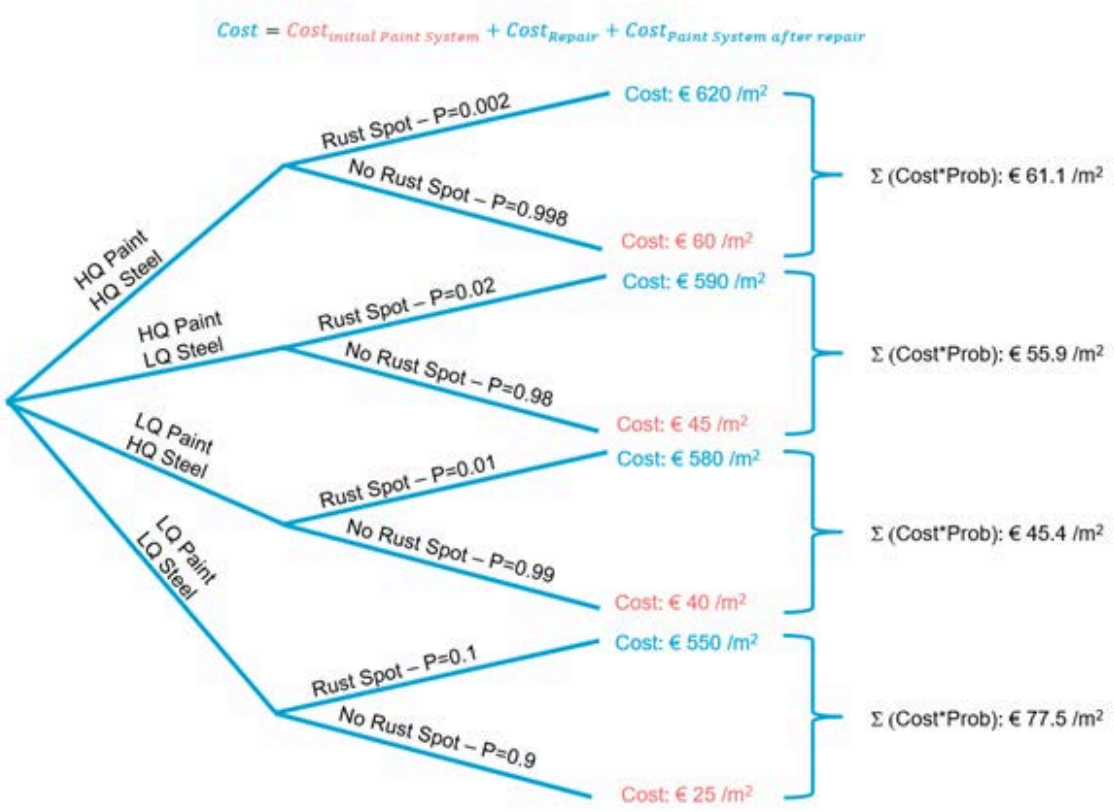



*Fig. 8.26* Decision tree.

**Question 1:** What is the probability of each paint system failure per square meter over one year?

**Answer**

| Paint | Cleaning | $p_f$ [1/yr] |
|---|---|---|
| High-Quality | Stripping | 0.002 |
| High-Quality | Sanding | 0.02 |
| Low-Quality | Stripping | 0.01 |
| Low-Quality | Sanding | 0.10 |

**Question 2:** What is the probability of each paint system failure per square meter over a 10 year period?

**Answer**

$$P_{f,10y} = 1 - (1 - p_{f,1y})^{10}$$

| Paint | Cleaning | $p_f$ [1/10-yr] |
|---|---|---|
| High-Quality | Stripping | 0.02 |
| High-Quality | Sanding | 0.18 |
| Low-Quality | Stripping | 0.10 |
| Low-Quality | Sanding | 0.65 |

**Question 3:** What is the expected cost per square meter over a one year period if the owner uses the high quality cleaning and high quality paint?

**Answer**

$$40 + 20 + 0.002 \cdot (500 + 40 + 20) = 61.12 \text{ Euro}/\text{m}^2$$

**Question 4:** What is the optimal paint system for a one year period?

**Answer**

$$C_{paint} + C_{cleaning} + P_{f,1y} \cdot (C_{paint} + C_{cleaning} + C_{repair})$$

| Paint | Cleaning | Expected cost [€/yr] |
|---|---|---|
| High-Quality | Stripping | 61.12 |
| High-Quality | Sanding | 55.90 |
| Low-Quality | Stripping | **45.40** |
| Low-Quality | Sanding | 77.50 |

Low Quality Paint + High Quality Cleaning

**Question 5:** What is the optimal paint system for a 10 year period?

**Answer**

$$C_{paint} + C_{cleaning} + P_{f,10y} \cdot (C_{paint} + C_{cleaning} + C_{repair})$$

| Paint | Cleaning | Expected cost [€/10 yr] |
|---|---|---|
| High-Quality | Stripping | **71.10** |
| High-Quality | Sanding | 144.70 |
| Low-Quality | Stripping | 91.63 |
| Low-Quality | Sanding | 366.94 |

High Quality Paint + High Quality Cleaning

**NOTE:** If you're getting slightly different results, don't panic! Most likely you have approximated the probabilities computed above in question 2.

Assume the owner has decided to use the cheap paint system, and is simply going to sand it off (i.e. the cheap cleaning method) and re-apply the same system every year (i.e., for these problems consider a 1-year reference period). You have been asked to assess whether it is worthwhile to use a quick sonic test system to prevent failures during the year, and if it is, the number of tests that should be performed per year. Each test costs about €3 per m2,

and if a weak spot is found, a new layer of paint can be easily applied, lowering the failure probability by a factor of 0.50.

**Question 6:** What is the annual expected cost due to paint failure if no sonic inspections are done?

**Answer**

$$E(D) = P_f \cdot D = 0.1 \cdot 525 = 52.50$$

**Question 7:** What is the annual expected total cost for the paint system if no sonic inspections are done?

**Answer**

Using the decision tree from above: €77.50/m2

**Question 8:** What is the change in risk for 3 sonic inspections?

**Answer**

$$\Delta E(D) = (P_{f,0} - P_{f,new}) \cdot D = 45.94$$

**Question 9:** What is the annual expected cost due to paint failure if 3 sonic inspections are done?

**Answer**

The expected damage is: $\$E(D) = P_f \cdot D = 0.5^3 \cdot 0.1 \cdot 525 = 6.56\$$ But the 3 inspections cost €9, thus the total expected cost is €15.56. It is incorrect to multiply the inspection cost by the failure probability. NB: this calculation implies the inspections occur regardles of when the failure happens.

**Question 10:** What is the maximum number of inspections per year that can be performed that are cost-effective? Note: Round your answer towards the smallest integer.

**Answer**

For the investments to be cost effective:

$$\frac{\Delta E(D)}{I} > 1$$

Which results in a maximum of 17 inspections per year.

**Question 11:** What is the optimum number of sonic inspections to perform each year? Plot a graph, with the Risk, Investment and Total cost curves and clearly indicate the optimum.

**Answer**

As shown in the figure, the optimum is 3 per year.

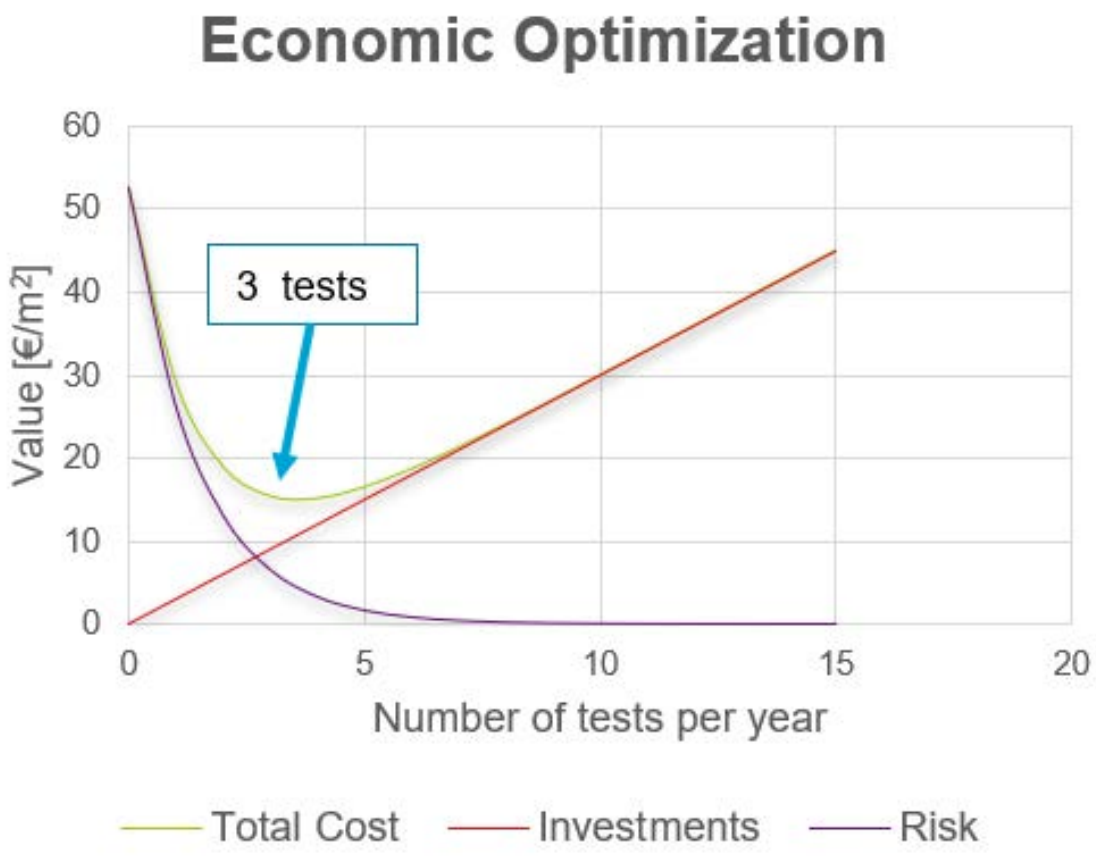



*Fig. 8.27* Optimization curve for exercise 12

> **Attribution**
>
> This chapter reuses content written by Bas Jonkman and Robert Lanzafame. Find out more here.

# Programming, Week 1.1: Getting Started!

> **Attribution**
>
> This chapter reuses material from *Learn Programming for Engineers*. Find out more here.

The purpose of this week is to get software installed and be prepared to work with Python code and Jupyter Notebooks. This includes some programming "theory" to help you understand what you are doing with the MUDE software stack, and why we use particular tools. A brief introduction to Computers and Computing Environments is provided before the software installation instructions. Once the software is ready, you are able to read and implement the information in: the File System and Python Warmup sections:

# Computers and Computing Environments

The two sections in this chapter have been prepared to introduce you briefly to what goes on “under the hood” in your computer. An understanding of computers and environments can make it much easier to understand why we use *environment managers* like `conda`.

# Computers

## Contents


- Summary of Application Software Goals


This is a brief overview of *computers,* with a focus on illustrating the role of application software and introducing the concept of *computing environments* (explained elsewhere in this book).

**What is a computer?**

A quick look on Wikipedia provides a simple definition: *a computer is a machine that can be programmed to automatically carry out sequences of arithmetic or logical operations (computation).*

This can take on a variety of meanings, and of course modern computers are everywhere around us, from smart watches and cell phones to the old desktop computer buried by old exams underneath Robert's desk. For our purposes in this book we will consider the following definition of computer

> **Tip**
>
> **Our Definition of Computer**: a machine that can be configured with software to execute programs to automatically carry out operations, in particular numerical analyses and visualization. The machine can be physically present (i.e., a desktop or laptop computer), or a virtual machine (i.e., a "cloud" computer; although this is accessible over the internet and may seem like it does not exist, in reality there is always a physical computer sitting somewhere in an air-conditioned server farm processing those 1's and 0's).

A computer can be broken down into a number of components, for example: hardware, software and peripherals, as illustrated in the figure below.

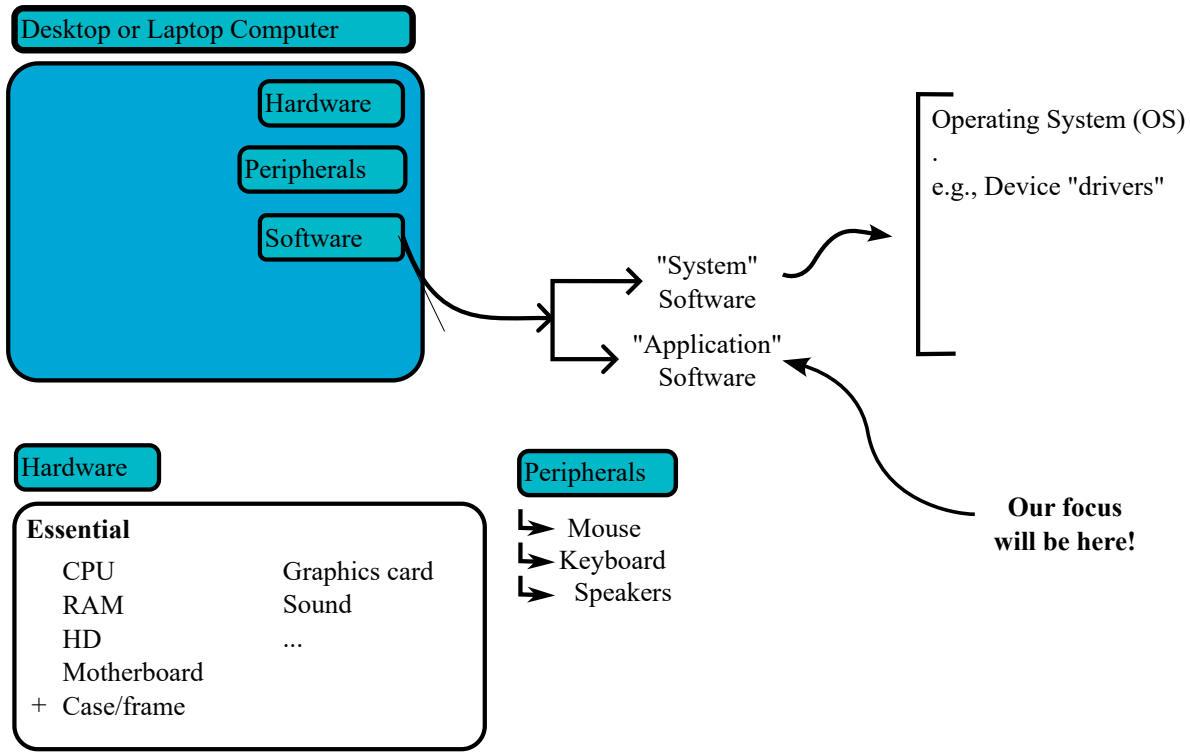



*Fig. 1* Essential components of a typical computer. In particular, we are interested in the Application Software.

Although it is not the focus of this book, it is important to recognize the complexity in hardware, software and components that we often take for granted in modern computers (similar diagrams can be imagined for everyday devices like your cell phone). Of all these components, our work in scientific computing generally focuses on the software part of a computer, which can be broken down into two parts: the *system* and *application software.*

The *system software* includes the *operating system* (OS) of a computer and the *drivers* that allow the OS to be run on a particular set of hardware. Common examples are the Windows and Mac (Apple) OS's. ALthough the OS on your computer is critical to consider, as we will see later, our goal is to install a few tools (software) that are compatible with our OS/computer and will eventually enable us to ignore the specific OS that we are running, especially when collaborating with colleagues.

As far as this book is concerned, the *application software* is where our true interest lies, as it is here where we will be writing programs to solve our problems of interest and collaborate with others. This is where our everyday software tools for scientific computing are categorized, for example, Python, VS Code, Matlab, our web browsers (e.g., Chrome) and much more software! As illustrated in the following figure, "we want to be" working at a relatively high level of application software (e.g., writing Python code to conduct analysis and make interesting visualizations) such that we don't have to worry about which OS we are working on. This is both to allow us to focus on our work, as well as collaborate others.

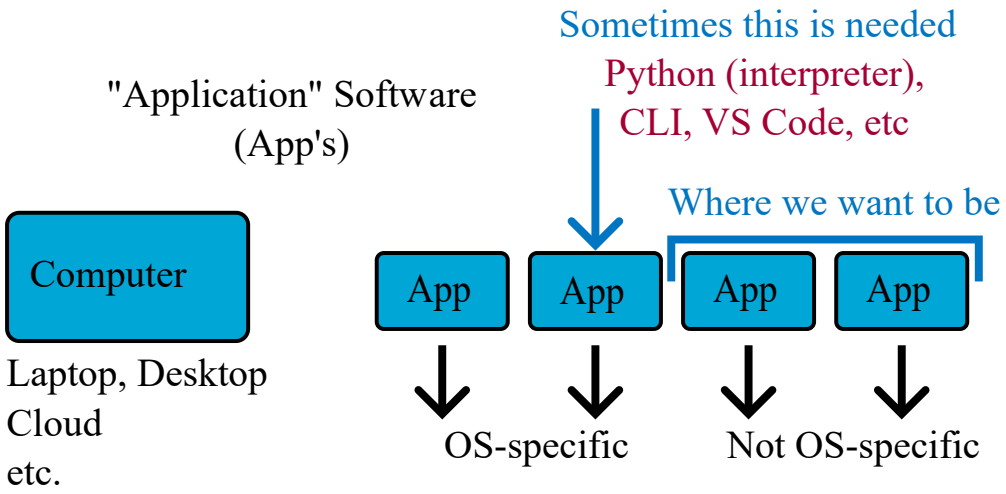



*Fig. 2* Relationship between the code we write, the OS-specific tools we need to install and our computer.

Of course, there are times when we still need to be conscious of which OS we are running, but this is less frequent, for example, when we are installing the proper version of `conda` or Python, as well as our IDE of choice (e.g., VS Code).

# Summary of Application Software Goals

Our goal is to write programs on our computer that can function on a wide variety of computers, primarily via relatively high level *application software*. In theory, this is simple: for example, we write code in Python and share it with others. Unfortunately the real world is more complex than that. There is huge variation amongst different computers, and it is not trivial to ensure that a piece of software produces the same results on all computers, let alone being able to run on various operating systems or intermediate application software.

Fortunately, there is a solution: *computing environments,* described in another section of this book.

# Environments and Environment Managers

## Contents


- What is an environment?
- Why is an Environment Manager Necessary?
- Examples of Environment Managers
- Environment Variables


This is a brief introduction to *computing environments,* with a focus on the Python programming language for scientific computing applications.

## What is an environment?

An *environment* is…*complicated.* Suffice to say it is related to operating system, programming languages and files on your computer, but also related to the hardware (i.e., the processor, etc). For our purposes, it mostly has to do with identifying a specific version of Python as well as the various Python *packages* (e.g., `numpy`) that are used for a given *project.* Often this concerns specifying *versions* that are compatible with each other; this is really what `conda` is doing when you ask it to build a new environment, or add a new package (e.g., with `conda install` or `pip install`).

Unfortunately, because computer hardware plays a role in determining compatibility, you may find that while something works perfectly on your own computer, it goes wrong on another. In summary, you can consider an environment as *a specification to define acceptable versions of various code to enable a project (code) to run successfully on your own computer, as well as the computers of others (e.g., fellow students, teachers and cloud-based computers).*

If the concept of an *environment* facilitates the running of code, an *environment manager* facilitates the creation, modification, specification, etc, of an *environment* itself.

The purpose of environments and environment managers is to allow an arbitrary software project to be used on an arbitrary computer, regardless of operating system (although one should note that some OS-specific software is required, for example the environment manager itself). This is illustrated schematically in the following figure.

*Fig. 3* An environment manager ensures that an environment can be set up and used for a particular project on an arbitrary computer.

In a more simple definition, an environment is simply a directory on your computer that contains the source code that has been selected by an environment manager to be compatible with the operating system and hardware on your computer, as well as the source code of the other packages in that same environment. The compatibility between these pieces of software is referred to as *dependency,* and tools that identify the proper versions of software are called *dependency solvers.* One commonly used example is `conda`.

# How to Create and Environment?

In many applications it is sufficient to determine a specific version of Python (e.g., version 3.12) and a list of the software you would like to use, then as an environment manager to download compatible versions and set up an environment. In a simplified view, this is roughly equivalent to downloading source code and putting it in a directory on your computer.

If one were using the `conda` environment manager, this could be done by listing the desired software in a `environment.yml` file (a simple text-based file) and running the following command:

```
conda env create -f environment.yml
```

Where `-f environment.yml` instructs `conda` which file contains the list of desired software, which contains, for example, Python 3.12 and a number of commonly used Python packages for scientific computing:

```yaml
name: my_environment_name
dependencies:
  - python=3.12
  - numpy
  - scipy
  - pandas
  - matplotlib
```

# Why is an Environment Manager Necessary?

Environments and environment managers make it very easy to share projects between collaborators without the need to make computer-specific versions that are compatible with a wide variety of hardware and operating systems. This is illustrated schematically in the following figure, where multiple environment managers can be used to run projects on different computers. Depending on the requirements of a project or computer, a wide number of environment managers could be considered by the user.

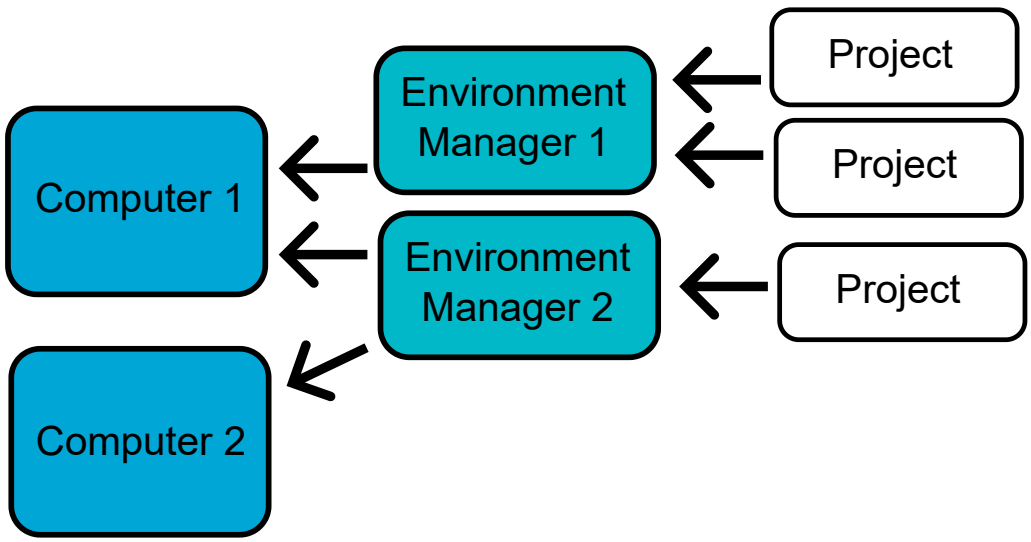



*Fig. 4* There are many types of environment managers that can be selected depending on a particular project, operating system or environment type.

# Examples of Environment Managers

One of the most common environment managers for scientific computing is `conda`; this popular command line tool is available many software distributions, for example, Anaconda or Miniconda.

Another popular environment manager is Python itself, via it's own virtual environment module (`venv`). Perhaps unsurprisingly, Python's `venv` is limited to the management of Python packages (primarily via `pip`), unlike `conda`, which can manage software packages and environments that use a wide variety of languages (for example, the R language, or LaTeX and Pandoc software).

## Choosing an Environment Manager

If you are a student, chances are the choice of environment manager has already been made for you. If not, you have a choice to make!

The use of Python virtual environments (`venv`) should be seen as an alternative to `conda` (especially if you installed Anaconda or Miniconda). If you are working primarily with Python packages available on `conda-forge` or PyPI, a `conda` environments will most likely work the same (and can also be easily selected from within VS Code). However, Python virtual environments (`venv`) can be created much more quickly than `conda` environments, and may enable you to manage your Python packages more efficiently in future projects.

However, note that `conda` comes with its own set of advantages, for example, in addition to support beyond Python packages, it has an improved ability to solve for package dependencies, which provides more reliability when distributing more complex software.

# Environment Variables

Modern computers have *environment variables*, which are used to define the behavior of your operating system (which is itself a type of computing environment). These variables are generally not important for most applications covered in this book, as we will focus on computing environments that are more distant from the operating system (for example, a Python environment; not the operating system directly). However, *in particular for users with a Windows operating system,* it is often needed to change the `PATH` environment variable to make sure various software can be used via a CLI.

Instructions for changing your `PATH` variable on Windows can be found via this link.

Feel free to visit the wikipedia page for additional explanation or examples.

# Software Installation

The software installation pages guide you through the process for installing several tools that we will use on a daily basis in MUDE. A brief overview is provided here:

- Miniconda: a software distribution that includes:
  - the Python programming language (we will use version 3.12)
  - Conda (or `conda`): an environment manager
  - a few other tools to make things go smoothly, for example the Python package manager `pip`
- Visual Studio Code (abbreviated VS Code or VSC):
  - this is our Integrated Development Environment (IDE) of choice becuase of its ease-of-use, flexibility and collaboration features
  - note that VSC is an alternative to Jupyter Notebook or Jupyter Lab IDE's, which you may have used in the past
- Jupyter: a Python-based software
  - Note that Jupyter Notebooks are a file format (extension `*.ipynb`), and Jupyter Notebook is also an IDE
  - We will install Jupyter in our computing environment using `conda` and interact with the Jupyter Notebook `*.ipynb` files using VSC as our IDE

> **Note**
>
> If you are not familiar with Jupyter notebooks you should spend some time getting comfortable; hopefully you have a patient group member that is willing to show you a few tricks. Even for those who have used them before, if you are unfamiliar with VS Code you will need a few minutes to get used to the slightly different interface.

# Miniconda Installation

## Contents


- Download and Installation
- Check Whether Installation Was Successful
- Using Anaconda Prompt


Miniconda is smaller version of Anaconda that is very quick and easy to install. Like Anaconda, it contains the essential tools to start working with Python: `conda`, Python and a few related packages. You can read more about it on the Miniconda website.

The screenshots in these instructions were made using the Command Prompt on a Windows Operating System (screenshots are from Windows 10). Where needed, instructions and notes for Mac users are provided in addition.

> **Tip**
>
> Unless you have installed Anaconda or Miniconda recently (e.g., within the last 2-3 years), we recommend you completely remove older versions of these software distributions from your computer and start from scratch. While it is possible to upgrade them, in our experience it is usually faster and less problematic to simply reinstall.

## Download and Installation

As described on the Miniconda website installer page, Miniconda can be installed by running the following in the Windows Command Prompt:

**Windows** | **Mac**

As described on the Miniconda site, *these three commands quickly and quietly download the latest 64-bit Windows installer, rename it to a shorter file name, silently install, and then delete the installer.*

Open the command prompt by searching for `cmd` in the start menu.

```
curl https://repo.anaconda.com/miniconda/Miniconda3-latest-Windows-x86_64.exe -o minic
start /wait "" miniconda.exe /S
del miniconda.exe
```

**Tip**

Note that depending on your Windows settings `CTRL+V` does not always work; you may need to use the menu bar of the terminal window or right-click and select paste. In addition, the paste feature typically works one line at a time, therefore after the first line is executed (download) you may need to hit enter once or twice to complete the last lines, as they may not execute automatically.

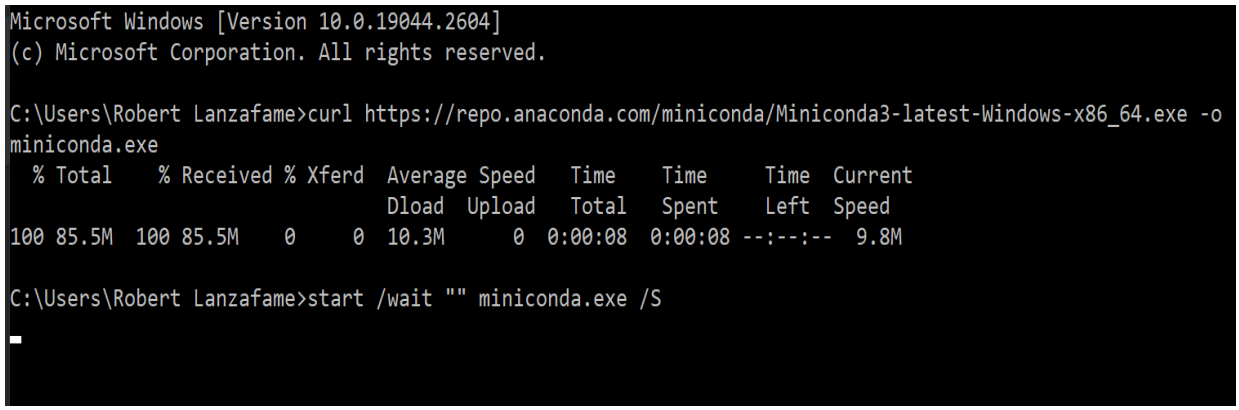



*Fig. 5* Command line installation of Miniconda (Windows).

**Make sure `conda.exe` is available on your system `PATH`**

On a Windows machine the Command Prompt doesn't know where to find the executable file `conda.exe` you just installed. You can resolve this in two simple steps: 1) find the location of `conda.exe` on your computer, and 2) add the location to the `PATH` environment variable.

These two steps are described below

1. Find the location of `conda.exe` on your computer.

The executable file `conda.exe` is what allows you to run the command `conda` from a CLI. The default location of this file is in your computer user settings: `C:/Users/<your user name>/miniconda3/`. Inside this folder you should see a number of files, one of which is a subdirectory `./Scripts/`; inside this folder are a number of executables, one of which is `conda.exe`. Therefore, the complete path to `conda` is usually this:

```
C:\Users\<your user name>\miniconda3\Scripts\conda.exe
```

> **Tip**
>
> Can't find the file and folders described above?
>
> Check out the Hidden Files and Folders page!

2. Tell your computer how to find `conda.exe`

This is described on a **separate page:** environment variable instructions for Windows OS.

# Check Whether Installation Was Successful

Before moving on, you should verify that the installation worked properly. A "successful" installation implies that the `conda` application is accessible on your CLI; this is required for some IDE's to work properly (e.g., VS Code).

The easiest way to check if everything worked properly is to make sure that the `conda` command line tool can be found and executed on your system, which can be done by opening the command prompt and typing `conda --version` (note the two dashes, `--`, not one, `-`!). Depending on your operating system and CLI; the following code snippet illustrates the command and response in Windows Command Prompt:

```
C:\Users\YOUR_USERNAME>conda --version
conda <YOUR_VERSION_NUMBER>
```

> **Note to Windows Users**
>
> There is a difference between the Command Prompt and the Anaconda version of the Command Prompt (Anaconda Prompt). Make sure `conda` is accessible on the "normal" Command Prompt. As it is made accessible in Anaconda Prompt by default during the installation process, it does not confirm whether or not `conda` is also added to the `PATH` variable for your computer.

# Using Anaconda Prompt

Now that Miniconda is installed on your system, from now on we will use the Anaconda Prompt instead of the windows command prompt; this way instructions will be identical regardless of whether you are using Windows or Mac OS!

**Windows OS** | **Mac OS**

Open the Windows tool bar and begin typing “Anaconda”. You should soon see two options appear: “Anaconda Prompt” and “Anaconda PowerShell Prompt.” There two versions are based on the two primary CLI’s on Windows: Command Prompt and PowerShell. Unless you are an experienced PowerShell user, we recommend using the Command Prompt version: Anaconda Prompt (see figure below):

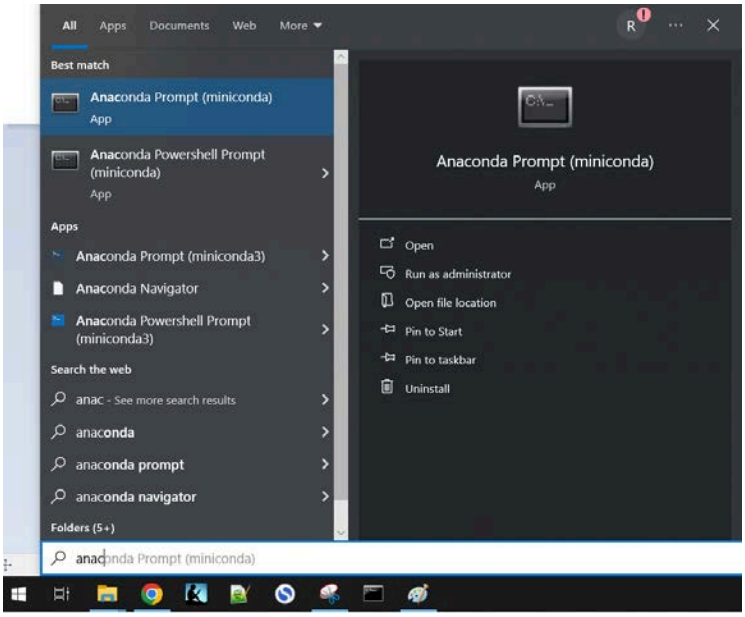



*Fig. 6* Open Anaconda Prompt from the Windows menu.

You can now use conda and all the packages within it!

# Hidden Files

## Contents


- Change Settings via a GUI
- Use the Command Line Interface (CLI)


By default some of the folders (directories) and files are hidden on your computer to protect you against making changes that causes problems. You will need to see some of these files and folders to properly set up and use the software described in this book.

## Change Settings via a GUI

### Mac OS

This is very easy to do on a Mac, simply use the following keyboard combination (i.e., holding three buttons simultaneously):

```
Command + Shift + .
```

If this doesn't work, try a Google search for "Finding hidden files on a Mac" and look for an explanation that includes a computer and OS similar to your personal setup.

### Windows OS

Settings for hidden files and folders can be found directly via the Windows File Explorer. First find the "options" or "settings" configuration window (sometimes pressing the `ALT` button toggles the menubar on and off):

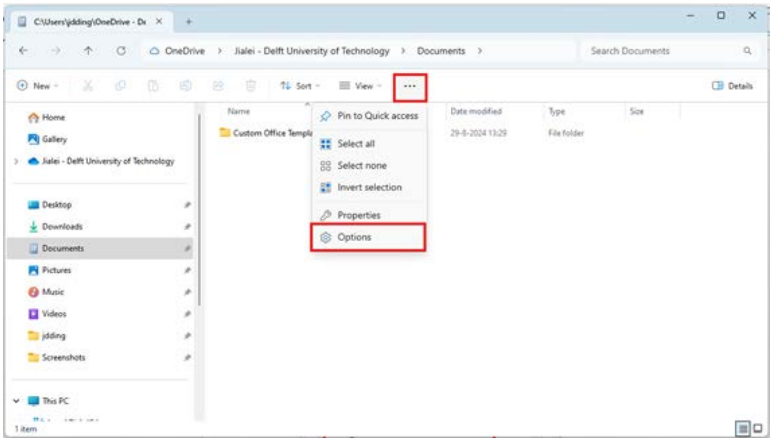



*Fig. 7* Find the options window for your directory (Windows).

Then navigate to the proper location in the “View” tab of the Folder Options window:

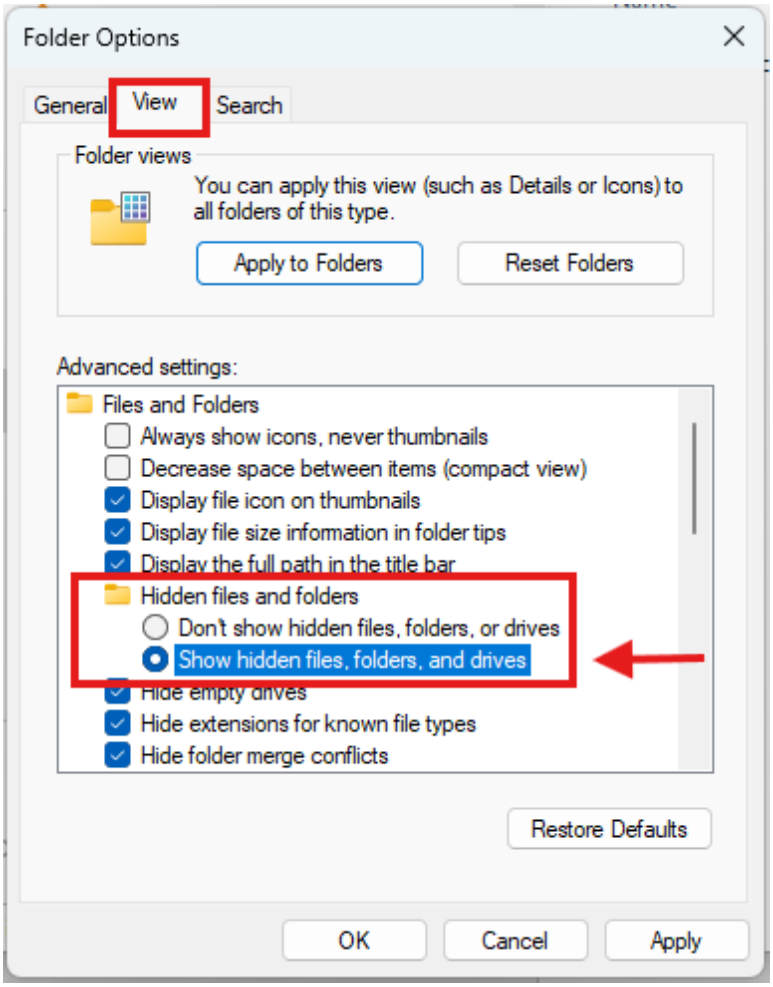



*Fig. 8* Show hidden files and folders for your directory (Windows).

# Use the Command Line Interface (CLI)

If you would like to view hidden files and folders via the command line, this is illustrated with the simple commands here (review the CLI page for an overview of the different systems):

Unix-type CLI | Command Prompt (Windows) | PowerShell (Windows)

The option `a` lists all files (including hidden ones) and the `l` presents the output in a nicely formatted list (one item per row):

```
ls -la
```

# Command Line Interface

## Contents


- Definitions and Terms
- Examples of CLI
- Basic CLI Skills: Cheat Sheets
- Links to other Cheat Sheets


A *Command Line Interface* (CLI) is a method of interacting with a computer that involves individual lines of text. These so-called *command lines* can be entered manually by the user, as well as written in a text-based file and passed on to a computer or application for automatic entry and execution. The alternative to a CLI is a *Graphical User Interface* (GUI).

> **Why use a CLI?**
>
> A few of the main reasons for using a CLI in engineering work are:
>
> - rudimentary tasks are easily automated and repeated (e.g., copying, moving, renaming files and folders)
> - some software does not have a GUI
> - a CLI can easily be opened on any modern laptop and can be much faster than using a GUI
>
> For example, once you can use the CLI you will probably find it much easier than using a GUI to navigate to a working directory on your computer, create and activate a computing environment for your project and open and begin working in your chosen IDE.

There is a long history of CLI's and many different types; a brief overview and set of examples are provided on this page.

# Definitions and Terms

There are various terms to be aware of, but we will only give a brief explanation here (see this tutorial for a more detailed explanation):

- **Terminal**: a piece of hardware for entering data into a computer. How we did it in the "old days" (the Wikipedia page has a few examples).
- **Console**: similar to a terminal, but adds hardware, interaction and a few other things (if you want to understand the difference, try reading this)
- **Shell**: an application that allows a user to write commands and enter data that can be translated into commands and data that can be executed by the operating system and hardware on a given computer; in short, this is how your instructions can be interpreted by any computer, regardless of hardware or operating system. A shell generally has two methods for interaction, a *graphical user interface* (GUI) or a *command line interface* (CLI).
- **Command Line Interface**: a method of interacting with a computer that involves typing lines of text into an interface; modern CLI's are implemented in shell applications. Visit the CLI Wikipedia page for an exhaustive overview of CLI's.

Now that these have been defined, you might realize that most of your life you have probably used a GUI to interact with your computer (remember, your mobile phone is also a computer!). This page is about using the alternative shell interface: a CLI!

You will hear many people and see software that use the terms above interchangeably (for example, VS Code uses the term "terminal" for all CLI's), but in reality usage of the term *terminal* or *console* today generally refers to a *shell* with a *CLI.* We recommend sticking to the term **command line interface (CLI)**, or simple the **command line** as this is platform independent and captures what we are physically doing with regards to the content in this book: entering and executing commands via a CLI.

# Examples of CLI

There are many types of modern CLI's that you may come across, here is a short list to give you an idea of their names and a few observations about each one:

- **Terminal**: the standard CLI on a Mac OS
- **Command Prompt**: one of several CLI's commonly used on Windows OS
- **PowerShell**: one of several CLI's commonly used on Windows OS. Notorious for having a syntax and set of commands that is different from most other CLI's (we

recommend you avoid this one, if possible)

- **Anaconda Prompt**: a custom CLI that builds on other CLI's. For example, on Windows OS you can find a Command Prompt and PowerShell versions, depending on your preference
- **Bash**: widely used on Linux systems
- **Git Bash**: a CLI that comes with the Git software distribution that is the easiest way to get Unix style commands on a Windows OS

> **Tip**
>
> Many CLI's use *UNIX style syntax and commands.* UNIX is a family of operating systems that were developed in the 1970's and influenced many other computer OS's.
>
> Mac and Linux OS's use UNIX style commands, but unfortunately the CLI's'that are available by default on Windows OS do not. This can make it more difficult to use the CLI for Windows users, especially because the vast majority of online documentation and open source CLI tools available today use UNIX-like syntax and commands.
>
> It is easy to work around this issue, however, for example:
>
> - it is easy to find the equivalent commands between UNIX and Windows CLI's _once you are aware that there is a difference) (see examples below)
> - it is possible to get UNIX style CLI's on Windows OS (e.g., Git Bash)

# Basic CLI Skills: Cheat Sheets

The tables below will give you commonly used commands on Mac and Windows OS's

> **Note**
>
> We've only included the Powershell commands here. However, the Windows Command Prompt (cmd) is also very easy to use. You can use the descriptors in the tables below to help with a web search, for example: *list directory contents windows command prompt.*

| | Mac (UNIX Style commands) | Windows (Powershell) |
|---|---|---|
| Present Working Directory | `pwd` | `pwd` |
| List content of the present working directory | `ls` | `ls` |
| Change: Change Directory | `cd /path/to/directory` | `cd /path/to/directory` |
| Touch: Creates a new file or updates the timestamp | touch {filename} (`touch my_file.txt`) | touch {filename} (`touch my_file.txt`) |
| Create new directory | mkdir {directory_name} (`mkdir my_folder`) | mkdir {directory_name} (`mkdir my_folder`) |
| Copy: Copies file or directory | cp {source} {target} (`cp my_file.txt my_folder`) | cp {source} { target} (`cp my_file.txt my_folder`) |
| Moves a file or directory | mv {source} {target} (`mv my_file.txt my_folder`) | mv {source} {target} (`mv my_file.txt my_folder`) |
| Removes: Removes file | rm {filename} (`rm my_file.txt`) | rm {filename} (`rm my_file.txt`) |
| Removes: Removes directory | rmdir {directory_name} (`rmdir my_folder`) | rmdir {directory_name} (`rmdir my_folder`) |
| Clear up terminal | `clear` | `clear` |

**Python**

| | Mac (UNIX Style commands) | Windows (Powershell) |
|---|---|---|
| Verifying and version | `python3 --version` | `py -3 --version` |
| Opening Python shell | `python3` | `py3` |
| run python files | python3 {py_file_name} `python3 filename.py` | py3 {py_file_name} `py3 filename.py` |
| Open python file in VS Code | code {py_file_name} `code filename.py` | code {py_file_name} `code filename.py` |

> **Tip**
>
> Depending on your computer settings, you may need to use `python` or `python3` to call the interpreter. Usually one or the other words by default on Mac, as it comes pre-installed with Python. For Windows OS, we recommend you use the Anaconda Command Prompt, as it was installed along with Python, which should be findable as long as you added `conda.exe` to your `PATH` environment variable.

# Links to other Cheat Sheets

Here are links for more detailed command line cheat sheets:

Windows command line cheat sheet

Mac Terminal command line cheat sheet

# Environment Variables

## Contents


- The `PATH` Variable


> **Note**
>
> This is for Windows Operating Systems only. It is often required when installing command line tools, for example `conda`.

Your computer has *environment variables* which define the behavior of your operating system. For example, one commonly used variable is `PATH`, which helps your computer locate key files, especially when using a CLI.

**Tip**

Keep in mind that the Windows OS allows you to set environment variables for a specific *user* as well as for the *system* (all users). It is important to note which environment variable you should set for a given activity, as choosing the wrong one can have unintended consequences (or lead to much frustration!). The difference can be seen in the example here:

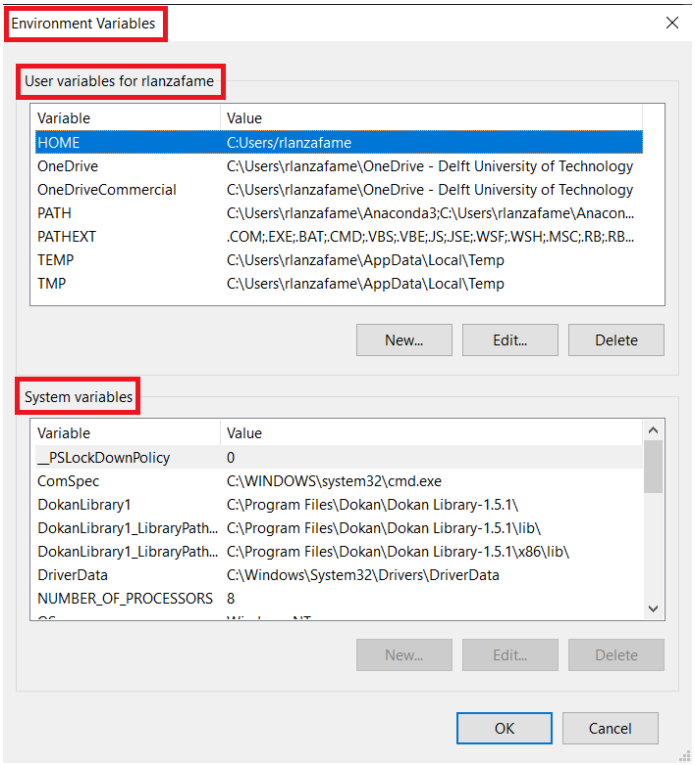



*Fig. 9* Environment variables, highlighting the difference between those for the user and system (all users).

# The `PATH` Variable

The `PATH` variable is like an address book for your computer: it lists various folders (directories) where the computer should look when executing commands. Updating the `PATH` variable is often required when installing software that is to be used with a CLI. In addition, updating this variable is often the solution to the problem of trying to execute a command and getting a “not found” message in response. For example, the following Windows Command Prompt illustrates what happens when (first) a command is executed that cannot be found by the CLI and (second) `conda` is called:

```
C:\Users\YOUR_USERNAME>software_that_is_not_on_path --version
'software_that_is_not_on_path' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\YOUR_USERNAME>conda --version
conda 23.11.0
```

**Tip**

Note the two dashes used with `--version`, not one, `-`!

The message that is returned with the `conda` version number indicates not only that `conda` is installed *and* that it is findable via the Windows `PATH` environment variable.

# Setting the `PATH` Variable

Setting the `PATH` variable is as simple as giving the path to the folder (directory) where the file or program of interest is located. All that is needed is to explain where the variable needs to be updated.

**Before you start, find your desired path!**

These instructions assume that you have *already* identified the path of the folder (directory) which needs to be added to the `PATH` variable. On windows, the easiest way to do this is to navigate to the folder using the File Explorer, then click in the address bar near the top and copy the text, which should look something like this:

```
C:\Users\<your user name>\miniconda3\Scripts
```

In this case, the folder (directory) contains the executable files `conda.exe`, which is what makes the command `conda` possible to use on the CLI. Thus, the full path to `conda.exe` is this:

```
C:\Users\<your user name>\miniconda3\Scripts\conda.exe
```

Note also that depending on your system settings, or if you are copying the path from a CLI, you may need to adjust the separators from a forward slash `/` to a backward slash `\`; the `PATH` variable on Windows requires a backward slash `\`.

The easiest way to find the appropriate setting window is by typing the first letters of “environment” in the Windows toolbar. The first three letters are usually enough to cause several relevant options to appear; select *Edit environment variables for your account* and see the figure below for an illustration:

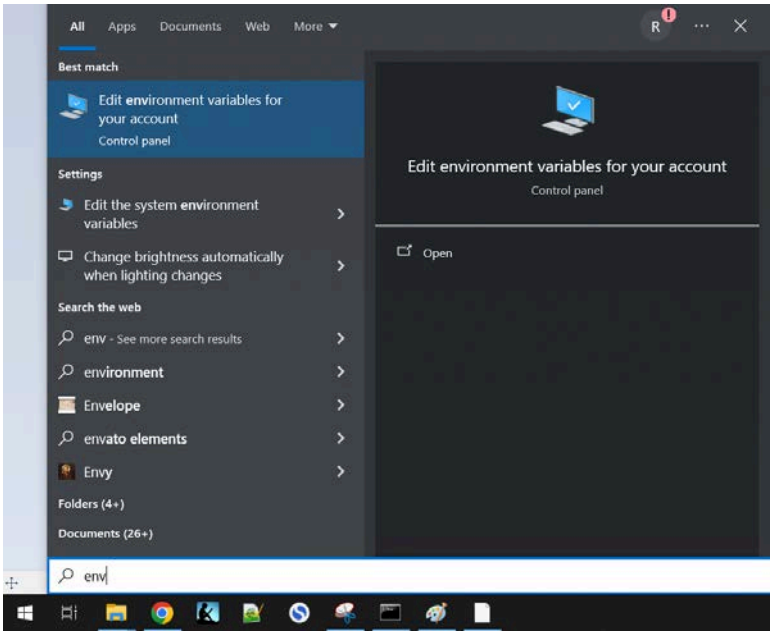



*Fig. 10* Search for environment variable settings; note the description “for your account”!

As a general rule you should only adjust the `PATH` variable for the *user*, **not** the *system.* Failing to do this is a common mistake and leads to frustration when installing applications like Anaconda and Miniconda.

To set the right variable, pay particular attention to the descriptions provided in the environment variable window, as it is easy to miss; the right location is illustrated clearly in the following figure:

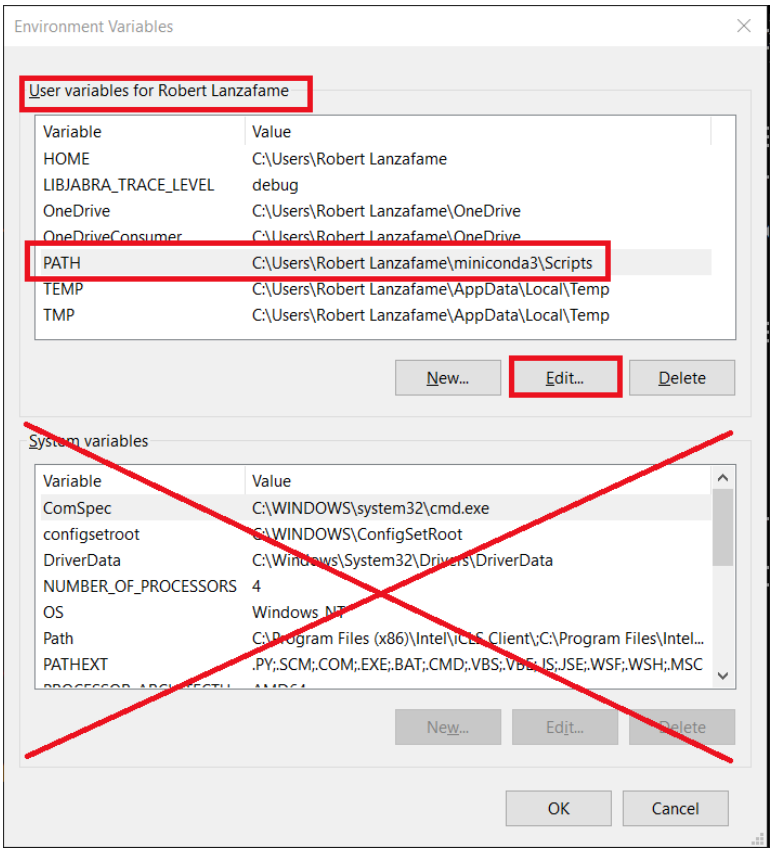



*Fig. 11* Unless specified otherwise, don't set an environment variable for the *system*, only your account!

Once you have identified the proper (user) window, look for the `PATH` variable, select it, then click the "Edit…" button. If the variable value is empty, you will probably see a window similar to that in the following figure; enter (paste) the path of the folder (directory) you wish to add to the path in the field "Variable value" then click "OK." Note that you can browse for the file and folder of interest via the settings window if you did not already copy the path via the File Explorer or a CLI.

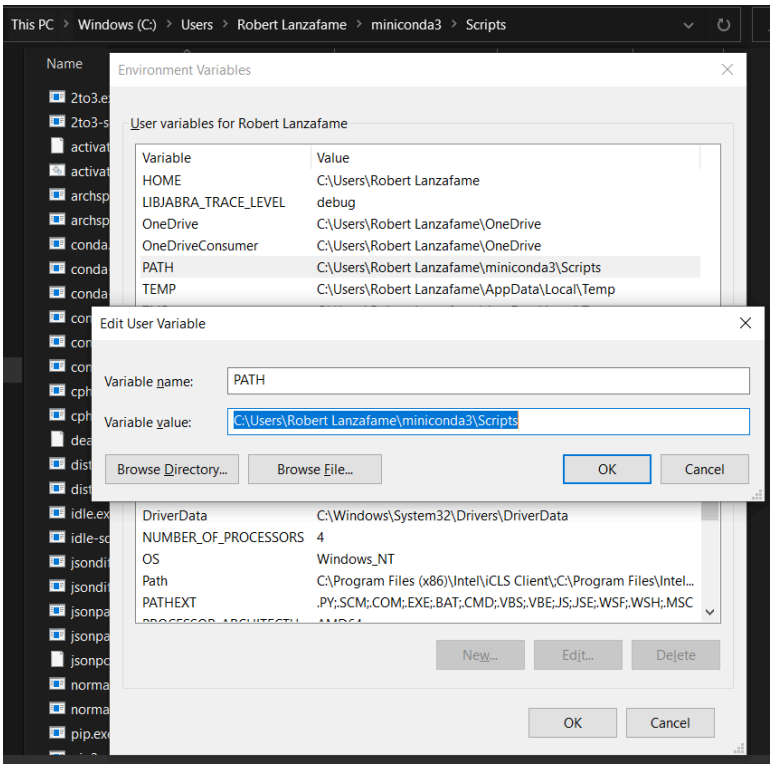



*Fig. 12* How to add a folder (directory) location to the `PATH` variable. The example here is for Miniconda.

If there are already values set for the `PATH` variable then the “Edit…” button will probably show a window like that in the following figure. In this case, you can add a new path via the “New” button; paste the path in an open row of the list. Note that you can browse for the file and folder of interest via the settings window if you did not already copy the path via the File Explorer or a CLI.

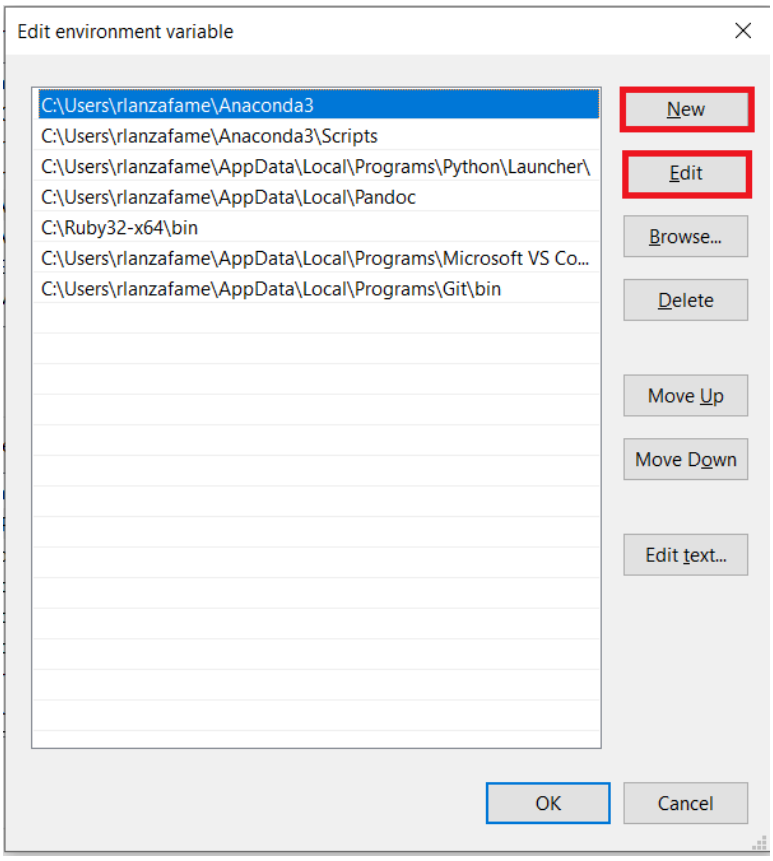



*Fig. 13* Example showing folders (directories) that have already been added to the `PATH` variable. Note the "New" and "Edit" buttons.

The following figure illustrates exactly where you should paste your desired folder (directory) path after clicking the "New" button:

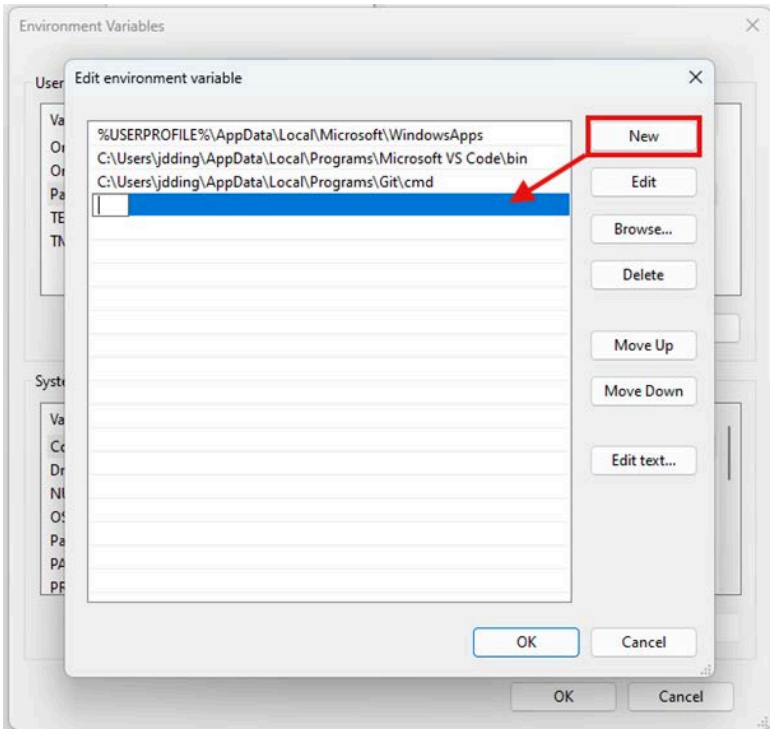



*Fig. 14* How to add a folder (directory) location to the `PATH` variable using the "New" button.

# Checking `PATH` Configuration

To check that your `PATH` variable was properly configured you should use a CLI to see whether your program of interest is now findable. For example, if you added the location of `conda.exe` to `PATH` you should now be able to execute `conda --version` (note the two dashes, `--`, not one, `-`!), which will return the following:

```
C:\Users\YOUR_USERNAME>conda --version
conda 23.11.0
```

**Tip**

You usually need to restart a CLI for changes to the variables to take effect. Simply closing and reopening the CLI is usually enough. If you are completely sure that the setup was done properly, it may also be prudent to restart your computer.

# Visual Studio Code (VSC)

## Contents


- VSC Installation
- Command Palette
- Using a CLI (a VSC “Terminal”)
- Activating a Python Environment


Visual Studio Code (VSC) is and *Integrated Development Environment* (IDE): a software that helps users program more effectively, as it combines different workflows in one place. For our purpose VSC is very useful because allows us to do things in one place, for example:

- manage files and sub-directories within our working directory
- execute Python code (and any other programming language!), including Jupyter notebooks
- manage Python environments
- use Git and version control
- many other things, especially with Extensions!

Although Jupyter Lab and Jupyter Notebook are also IDE’s, they don’t provide as many features as VSC, or maintenance and improvement of tools is slower because the Jupyter development community is smaller. The availability of extensions in VSC, in particular, is exceptional.

## VSC Installation

The primary steps are:

- download the installer from code.visualstudio.com/download
- install on your computer (default settings are fine)
- get familiar with the interface
- install a few useful Extensions (see overview page)

- use it!

# Command Palette

VSC is very easy to customize, and you can spend hours optimizing it for your own personal use and projects. We recommend skipping this for now if you are just getting started. However, it is useful to know about the *Command Palette,* which governs all functionality of VSC, and can be accessed with `CTRL+SHIFT+P` for Windows or `SHIFT+CMD+P` for Mac. *This is the single most important VSC command to remember.*

# Using a CLI (a VSC "Terminal")

VSC refers to a Command Line Interface (CLI) with the term "Terminal."

You can open a new Terminal with the menu bar at the top. Once a Terminal is active you can open additional Terminal instances, choosing amongst the various CLI's available on your computer. It can be useful to have more than one CLI open at a time and keep them collected together with your project files!

To have a specific CLI open by default when you open a Terminal in VSC, open the Command Palette (`CTRL+SHIFT+P`, `SHIFT+CMD+P`), select the option `Terminal: Select Default Profile` and select your preferred CLI from the list.

# Activating a Python Environment

The easiest way to do this is to make sure you have the Python extension installed, then try to run the cells of a notebook or execute a `*.py` file (look for the play button icon once you have either of these file types open). Upon clicking the run button you will see that VSC asks you to choose a Python kernel. You can choose a Python virtual environment (`venv`) if you have one, but you should also be able to see your `conda` environments as well.

Remember that if you need to install more packages, you should open a CLI (this is called a "terminal" in VSC) and use `pip`, which you can do directly from VSC.

# VS Code Extensions

## Contents


- First Time User
- Markdown All-in-One
- Code Spell Checker
- Git


This page gives an overview of useful extensions in VS Code, with short tips on how to install them or what they could be used for. It is organized by various user types.

To install an extension, select the “Extensions” box on the left side of the application (the symbol is four boxes), then search for and install the following (entering the ID in the search box turns up the right result).

As VS Code and its extensions are actively developed by a large international community, we don’t include exhaustive instructions and troubleshooting help in this book; instead, you should search for up-to-date solutions on the internet. For example, the Extension Marketplace page from VS Code is already quite thorough.

## First Time User

New to VS Code? New to programming in general? If you are using Python, we recommend starting with the following extensions:

- Python (Extension ID: `ms-python.python`)
- Jupyter (Extension ID: `ms-toolsai.jupyter`)

> **Tip**
>
> Note that if you are using the Jupyter extension it is good to include `ipykernel` in your Python environments (but VS Code will add this automatically for you if you don’t - isn’t VS Code great?!).

# Markdown All-in-One

Extension ID: `yzhang.markdown-all-in-one`

This is a useful tool for rendering Markdown files in real time. Once installed, open a `*.md` file and use the shortcut `CTRL+SHIFT+V` to open a preview page. It is useful to arrange the preview side-by-side with the source code so you can see the marked up version as you edit. This is extremely useful for checking figures, equations and other Markdown formats before pushing files to other locations (e.g., a Git repository).

> **Tip**
>
> As Markdown is implemented in a wide array of different software, it has many different variants (called “flavors”). For example, see the Wikipedia Summary of Markdown Variants. This means that what you see when your Markdown text is rendered may appear differently on other platforms. For example, the appearance of your file on GitHub maybe different than what you see in VSC. The takeaway? *Always check your Markdown file on the platform where it will be shared!*

# Code Spell Checker

Extension ID: `streetsidesoftware.code-spell-checker`

This extension is useful for catching typos in your work. Read the extension documentation page for instructions on how it can be customized.

# Git

For general use with Git**Hub** use the following:

- GitHub Pull Requests (Extension ID: `GitHub.vscode-pull-request-github`)

# Files and Folders

## Contents


- Your Local File System
- A Typical MUDE Working Directory
- Saving Your Work


MUDE has a repetetive weekly workflow and a *lot* of different technical topics. Because of this, it is important to keep our files well organized. This page gives you the background information to do this efficiently. Your weekly assignments will also guide you through the process of setting this up in a gradual way.

> **Tip**
>
> This may be the first time you have ever had to consciously interact with the file system on your computer. If you are used to using software applications like OneDrive on your laptop, the setup required for MUDE may have been done automatically for you in the past. If you work primarily on a tablet or mobile phone, you probably *never* have to interact with the file system on your device.
>
> If you struggle with file system organization, or would like additional information, do not hesitate to come to weekly question hours and reach out to your MUDE instructors!

The *file system* on a computer is simply a way to organize digital documents. Because it is easily customizable, it can be confirgured in many ways, enabling the computer user to optimize it for their own purposes. For MUDE, it is useful to define a few key concepts:

- **files**: the digital documents on our computers.
- **folders**: the way we organize the **files** on our computer. It is possible to have **folders** inside other **folders**.
- **directory**: a hierarchical structure for organizing data on a computer. For our purposes, it is synonymous with **folder**.
- **sub-directory**: for our purposes, this is a folder within another folder.

- **top level directory** or **root directory**: the highest level in the directory structure. Note that this is a relative term; if we are considering only the scope of MUDE documents, perhaps the `MUDE` directory is the top level, even though it is also a sub-directory itself.
- **lower level directories**: this term refers to travelling "downward" through the hierarchical tree of the directory system
- **working directory**: the place where we will edit files when working on a particular project.

The concept of a **working directory** is especially important, as it is the primary place where we "do work." This includes everything from writing reports, writing and running code and programs, processing data, creating visualizations, etc. Ideally this directory is set up to include all relevant files for a particular project, such that when you are "doing work" you can ignore the rest of the file system on your computer. This has important practical implications, for example:

- you can open your IDE in the working directory and have access to all the files you need
- it is easy to back up (save) your work and share it with others

The goal of this page is thus to give you *just enough* information to be able to confidently set up and understand working directories on your computer.

> **Tip**
>
> Some of the content on this page describes the software Git and a related concept of *repository.* If you are unfamiliar with these concepts, you can ignore them for now and consider a *repository* to be equivalent to a *directory* of *sub-directory.*

# Your Local File System

You will be working with a *lot* of different files and file types in MUDE: `csv`, `pdf`, `ipynb`, `txt`, `md` just to name a few. Remember, there are 16 weeks of class in MUDE—it is good practice to get in the habit of storing files in an organized way, not to mention make it easier to study for the exams. We recommended creating a folder `MUDE` to collect all files associated with the module, then set up a new folder for each week and each project as your **working directory** for each of those activities. This is where you will download files and edit them when working on them.

### File Types and Extensions

File types are typically defined by their “extension,” which is the set of letters that appears after the name of a file, separated by a dot. If you can’t see these in the file explorer on your OS, you need to change your view settings to be able to see them. We often use the convention of a `*` to denote the arbitrary file name, for example `*.csv` is a generic `csv` file. If this is a foreign concept to you, just note for now that extensions make it easier for computer software to find files and decide what to do with them.

```
├── MUDE                          <-- a root directory for MUDE projects
    ├── Project_1                 <-- project files in sub-directories
        ├── Assignment_File.ipynb <-- stay organized with more sub-directories
        ├── auxiliary_files
            ├── data.csv
            ├── figure.png
            ├── ...
    ├── Project_2                 <-- another working directory
        ├── ...
    ├── Project_3                 <-- each of these should be it's own repository
        ├── ...
    ├── Project_4
        ├── ...
```

As you can see, we have included a directory `auxiliary_files` in the directory for Week 1. This is a generic setup to help keep the weekly folders from getting too cluttered: as our MUDE projects become more complex, we may want to use a variety of sub-folders to organize our files (e.g., data, code, figures, etc).

### Tip

If you have not yet learned about the software Git, or its command line invocation `git`, it is enough to know that it is a *version control system* that focuses on tracking changes to files and collaborating with others. It is software that can run on many OS’s and is integrated into other tools provided by 3rd party companies that add additional features, for example, a way to backup and collaborate in the cloud. Widely used examples of these companies are GitHub and GitLab.

The organization of your `MUDE` and working directories can vary widely due to the types of projects and your personal preferences. Your *version control* system may also dictate the structure. For example, in a course where you will have multiple repositories per week (e.g., CEGM1000 MUDE), you might consider organizing these `git` repositories in sub-directories, like this:

```
├── MUDE
    ├── Week_A
        ├── git_repo_for_project_1
            ├── Assignment_File.ipynb
            ├── auxiliary_files
                ├── data.csv
                ├── figure.png
                ├── ...
        ├── git_repo_for_project_2
            ├── ...
    ├── Week_B
        ├── git_repo_for_project_3
            ├── ...
        ├── git_repo_for_project_4
            ├── ...
    ├── Week_C
        ├── ...
    ├── Week_D
        ├── ...
```

**Tip**

If you are not yet working with Git or another version control system, it is OK to back up your work using a cloud-based software like OneDrive. However, note that this can sometimes lead to issues when running code and programs, so you may occasionally need to move the files elsewhere when working. As soon as you are comfortable with a tool like Git, we encourage you to **stop** using cloud-based software to back up your working directories; instead you can use a tool like GitLab (we will show you how in MUDE).

This is additionally explained in a section below.

# A Typical MUDE Working Directory

Within each **working directory** you will collect a wide variety of files. It is important to keep these well organized so that when you return to work on a project in the future, it is easy to pick up where you left off. As a preview of the types of analyses we will do, a typical MUDE working directory is illustrated in the figure below. Note in particular that it contains several file types, along with a few sub-directories for organizing content. The files that are edited directly (e.g., the Analysis or Report files) are included in the top level directory. In addition, a README file is included to provide general information about the working directory and its contents.

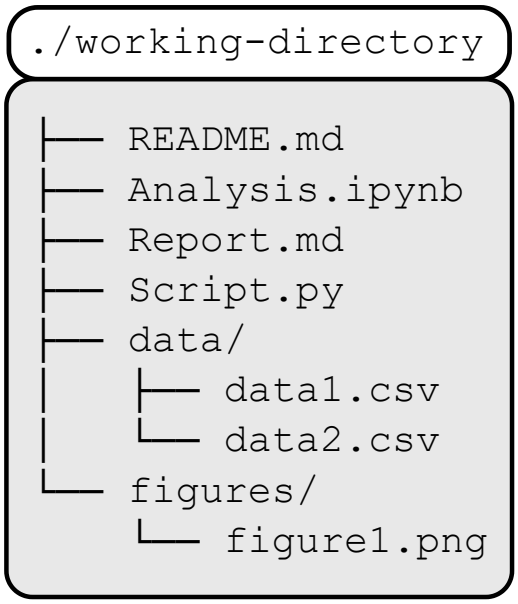



*Fig. 15* Contents and organization of a typical working directory in MUDE. Note the various file types, and how the files that are actively edited are located in the top level directory, whereas files that remain mostly static, or are produced as a result of running a program (i.e., figures) are in the sub-directories.

**Tip**

Markdown is a text-based file format that is used to easily make nicely formatted documents. We will use it to write Reports summarizing our analysis and findings. You can think of it like a (very!) simple alternative to LaTeX.

# Saving Your Work

It is important to keep a backup of your work, and to save these files in a consistent way. Your typical cloud-based backup software is OK for many types of files, but you should note that sometimes there can be issues when running code on your compter in these special sync folders. Imagine: your Python code is running and loading or saving data into files in the same folders that the cloud backup software is using: there are bound to be conflicts!

Over the course of the semester we will gradually increase our reliance on `git` and GitLab, a widely used set of tools meant to store, track and archive files. As you become more comfortable with this tool, you will learn to `git push` more and rely on your cloud storage system less! In summary:

- for now, we recommend you store everything in a location that is backed-up regularly, but keep code and data files in a separate folder. If your backup software causes issues

with notebooks or other Python code, move the working directory temporarily to another location on your computer.
- once you are comfortable with `git`, we recommend that you permanently move your code folder somewhere that is not backed up by cloud software and use your GitLab repository instead.

In case the text above was not clear: to avoid syncing problems, use a system like `git` to backup code, *not* a cloud service like OneDrive!

# Warm-Up: Jupyter Notebooks and Python Dictionaries

In this section we use a Jupyter Notebook to briefly review and practice with dictionaries, a fundamental Python data type that is *extremely* useful, but you probably don't remember from prior courses.

The code on this page can be used interactively: click 🚀 –> **Live Code** in the top right corner, then wait until the message **Python interaction ready!** appears.

When this page is activated a special version of Python will be loaded into your web browser that enables you to interact with this page as if it were a Jupyter Notebook. Note that package import does not work the same as it would on a "normal" computer, so the necessary imports have already been prepared for you.

The download button 📥 provides a `*.ipynb` file which you can run on your computer to make sure you have Python and Jupyter installed correctly in your IDE.

> ⚠ **Warning**
>
> The contents of this page will be updated by the Wednesday, September 4 In-Class Session to include a few brief tips for using Jupyter Notebooks and Python dictionaries.
>
> For now, enjoy the interactive Python page! You can also test your Python session using the download page button (as described above), but note that this file will also be updated when the page is.

```
engineers = {
    'Nikola Tesla': 1856,
    'Thomas Edison': 1847,
    'Albert Einstein': 1879,
    'Isaac Newton': 1643,
    'Leonardo da Vinci': 1452,
}

for engineer, birth_year in engineers.items():
    print(f'{engineer} was born in {birth_year}.')
```

# Week 1.2: Python Topics

> **Attribution**
>
> This chapter reuses material from *Learn Programming for Engineers*. Find out more here.

This week the programming topics were covered in a Jupyter notebook and based on general Python topics, most of which should have been covered in a BSc course on programming. The Learn Python book may be a useful reference.

# Week 1.3: VS Code Live Share!

> **Attribution**
>
> This chapter reuses material from *Learn Programming for Engineers*. Find out more here.

This week you'll learn some first VS Code lifehacks! Let's dive into it!

# Visual Studio Live Share

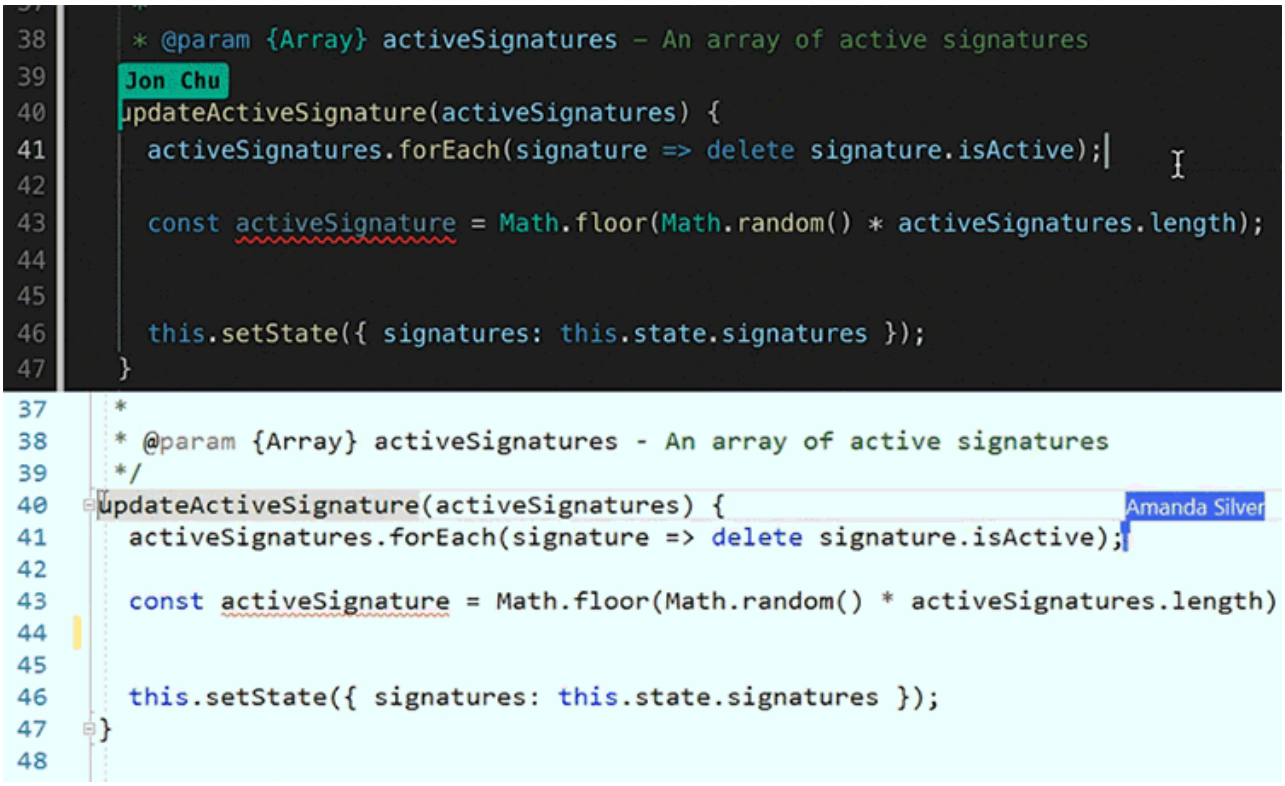



*Fig. 16* Figure taken from Microsoft (n.d.)

Did you ever get frustrated by sharing code with your friends? How nice would it be to do it the Google-Docs style and work on the same document simultaneously! Maybe you know Deepnote, which does this in an online interface. But there's a better solution: Visual Studio Live Share!

These steps are heavily based on the Visual Studio IDE documentation (Microsoft, n.d.).

1. Download and install the Visual Studio Live Share Extension from Marketplace and reload Visual Studio Code. For Linux, see this website.
2. To collaborate, you'll need to sign in to Visual Studio Live Share so everyone knows who you are. This sign-in is a security measure and doesn't add you to marketing activities or other research activities. You can sign in by using a Microsoft personal account (for example, @outlook.com), your TU-Delft account (@student.tudelft.nl), or a GitHub account.
3. Select **Live Share** on the status bar or select **Ctrl+Shift+P** or **Cmd+Shift+P** and then select **Live Share: Sign In**.

4. A prompt to sign in displays. When you select your sign-in preference, a browser opens so you can complete the sign-in process. Close the browser when you're done. If Visual Studio Code isn't picking up your sign-in after you complete the sign-in process in the browser, see this website.

You're ready to go! Continue with the official documentation or just try out, it's not hard!

# Week 1.4: Version control with Git!

> **Attribution**
>
> This chapter reuses material from *Learn Programming for Engineers*. Find out more here.

This week you'll learn how to do version control with Git

# Version Control

## Contents


- What is Version Control?
- File types: text versus binary
- A different way of thinking?
- git and GitLab
- Main concepts and terminology


In a nutshell, **version control** is a process that is concerned with the management and recording of files on your computer. These tools are primarily focused on text-based files, although for convenience we will also apply them to binary files such as PDF or raster images. While you may be used to backup software such as OneDrive, Dropbox, etc, version control software works a bit differently because of the focus on the text-based content. The chapters in this textbook will give you the information you need to take advantage of a version control software, git, for the programming projects you will work on during your MSc program.

## What is Version Control?

While working on personal or university projects, without a doubt you have come across the following situation: You have finished drafting a report and believe you are done with it and save the file as `report.doc`. However, later you decide to experiment and make some changes, but you still want to keep your old working version, so you make a new file called `report-final.doc`. Now imagine that you send the file to a friend of yours to proofread and make some comments on it and they send it back. Next, you incorporate their feedback and end up naming the new one `report-final-2.doc`.

What you have been doing is called version control. Version control systems start with a base version of the document and then record changes you make each step of the way. You can think of it as a recording of your progress: you can rewind to start at the base document and play back each change you made, eventually arriving at your most recent version, as shown in figure Fig. 17:

*Fig. 17* Consecutive file changes

Once you think of changes as separate from the document itself, you can then think about “playing back” different sets of changes on the base document, resulting in different versions of that document. For example, two users can make independent sets of changes on the same document, resulting in 2 independent versions (Fig. 18).

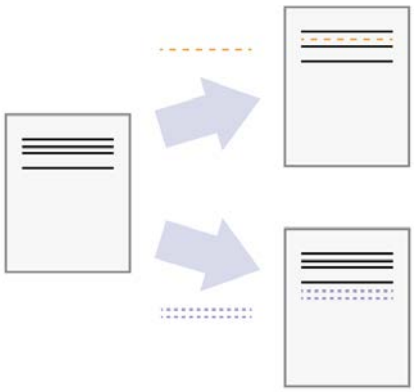


*Fig. 18* Parallel file changes

Unless multiple users make changes to the same section of the document - a conflict - you can incorporate two sets of changes into the same base document (Fig. 19).

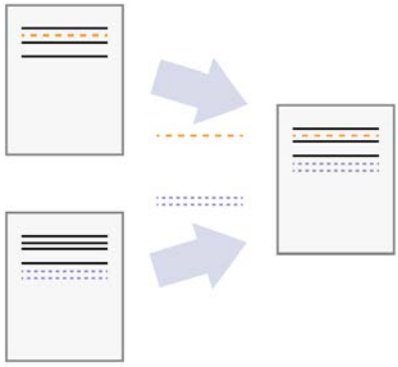


*Fig. 19* Merge of changes

# File types: text versus binary

Version control systems oriented towards software development and programming are typically focused on **text-based files**: files where the contents are viewable on your computer as human-readable text. **Binary files,** on the other hand, are organized and saved with bits (`0`'s and `1`'s) and are not human-readable. Although this may be a simplified description in terms of the way computers store information (you can read more here), it is enough for our purposes to recognize that text-based files are best suited for use with version control system; in other words, your Python code!

- Examples of common text-based file extensions are: `txt`, `md`, `csv`, `ipynb`, `py`, `html`, etc.
- Examples of common binary files are: `pdf`, `ppt`, `xlsx`, `docx`, etc.

> **Try it!**
>
> Try exploring a few files on your computer to confirm wether they are text-based or binary by opening them up in a text editor. You will easily be able to distinguish the difference because one is readable, the other not.
>
> Note that in Windows if you are using Notepad (the default), you will want to select "Word Wrap" under the "Format" menu to fit the contents of very long lines within the visible width of the window.

# Jupyter Notebooks

Jupyter notebooks, `ipynb`, are a special case because while the contents of your Markdown and code cells is saved as text in the file, the output of the code cells is sometimes a binary format. For example, if you create a plot using matplotlib and save the notebook, that plot output will be binary. This unfortunately makes it a little more difficult to use notebooks with VCS, but if we are aware of the issue, it is not a problem—we will show you how.

If you tried opening up a notebook in the text editor you would have noticed a structure with curly braces, `{}`. This is JSON-format] (another file type), which the notebook uses to store information in each cell.

# A different way of thinking?

As you will see in the other chapters on git, when applied to code, version control takes on a very different appearance than what you are used to with traditional backup software, for example, Microsoft Word auto-save, or cloud-based services like OneDrive, Dropbox or even Visual Studio Code Share. All of these platforms are set up in a user-friendly way that is *focused on a single file.* This works fine when we are writing a report like a thesis. However, it does **not** work well when it comes to computer programs, because in addition to the files themselves, the *contents of the file* become critical. As we will see, git is a version control software that allows us to compare and track changes in every character of text within a file, which is very useful when writing code, as well as working with a distributed team of collaborators.

> **Tip**
>
> To use version control (git) effectively, different versions of files are tracked with *commits*, **not** by duplicating or renaming the file.
>
> **Avoid copying and renaming files as much as possible!**

# git and GitLab

**What is git?**

Git is a version control system (VCS), used by a wide variety of engineers and software developers to work on projects in parallel together. It provides multiple benefits such as tracking changes to files, working side by side with other people, and the ability to rollback to previous versions of files without losing track of newer changes. It is a free and open sources software.

Note that while git is free and can be used on a variety of operating systems, there are many 3rd party softwares that *use* git directly, or are heavily dependent on git. For example, GitLab and GitHub are two companies that provide cloud-based servers for hosting git repositories, as well as additional features like user groups, discussion channels, and even hosting of websites

**What is GitLab?**

GitLab is a cloud-based version control system built around git. It provides a lot more features such as Issues, Merge Requests, CI/CD pipelines, etc. TU Delft has a license to

use GitLab on our own local webservers—this means that all of the files are stored digitally on the TU Delft campus. This is also why TU Delft has their our "own" GitLab located at `gitlab.tudelft.nl`, rather than the "normal" GitLab at `gitlab.com`, and is also something you will have access to throughout your studies.

**What is GitHub?**

GitHub is a competitor company to GitLab. It provides very similar services, but they are often called different names, or have slightly different features. Although we will not be using it directly, GitHub provides a free software that is very useful: **GitHub Desktop**! This software allows you to interact in a nice graphical interface with the version control of your files. An alternative is the Git functionality in VS Code

# Main concepts and terminology

Here we present a list of the terminology we may use when referring to version control systems (VCS). Do not panic if you do not understand what each of the following means. Later, we will provide a more elaborate explanation with examples. Bear in mind that the list below is not exhaustive, and more terms may show up. Also, if you only do the GUI option, you might not encounter some of them.

1. **Repository:** storage, where VCS (git, in our case) store their history of changes and information about who made them.
2. **Remote (of repository):** a version control repository stored somewhere else and the changes between the two are usually synchronized. We will refer to the Gitlab repository as a *remote*.
3. **Commit:** Snapshot of the current state of the project. If a commit contains changes to multiple files, all the changes are recorded together.
4. **Staging:** preparation of files to be committed. During the staging we propose files to be committed.
5. **Snapshot:** copy of the current version of the entire repository.
6. **Cloning:** copying (downloading) an existing project on your laptop. Usually, it is done only during the first time of getting the remote repository.
7. **Tracked (files):** files that Git knows about – they are either in the staging area or were previously added to the repository.
8. **Untracked (files):** files that Git does not know about – they are likely new files that have not been staged yet.
9. **Pushing:** uploading new commits (changes) to the remote server.

10. **Pulling:** retrieving new commits from the remote repository.
11. **Fetching:** check for new changes on the remote repository without pulling them yet.
12. **Conflict:** when changes made by multiple users to the same file are incompatible, you can get into a conflict. *Helping users resolve those conflicts is one of the key advantages of VCS.*
13. **Branch:** development (time) line. The main development line is called `main`.
14. **Merge:** combining the commits of two branches, for example, changes on a development branch are merged into the `main` branch.

# Git

## Contents


- Authentication with SSH


**Windows** | **Mac**

1. Download the Git for Windows installer.
2. Run the installer and follow the steps below:
    1. Click on “Next” four times (two times if you’ve previously installed Git). You don’t need to change anything in the Information, location, components, and start menu screens.
    2. From the dropdown menu, “Choosing the default editor used by Git”, select “Use Notepad as Git’s default editor” and click on “Next”. (Note: as long as you know how to use it, you can choose any other editor from the list)
    3. On the page that says “Adjusting the name of the initial branch in new repositories”, select “Override the default…” and type `main` in the box. In addition to being more inclusive, this will make things more compatible with our TU Delft GitLab setup and GitHub.
    4. Ensure that “Git from the command line and also from 3rd-party software” is selected and click on “Next”. (If you don’t do this Git Bash will not work properly, requiring you to remove the Git Bash installation, re-run the installer and to select the “Git from the command line and also from 3rd-party software” option.)
    5. Select “Use bundled OpenSSH”.
    6. Ensure that “Use the native Windows Secure Channel Library” is selected and click on “Next”.
    7. Ensure that “Checkout Windows-style, commit Unix-style line endings” is selected and click on “Next”.
    8. **Ensure that “Use Windows’ default console window” is selected and click on “Next”.**
    9. Ensure that “Default (fast-forward or merge) is selected and click “Next”
    10. Ensure that “Git Credential Manager” is selected and click on “Next”.

11. Ensure that “Enable file system caching” is selected and click on “Next”.
12. Click “Next” in the “Configuring experimental options” section.
13. Click on “Install”.
14. Click on “Finish” or “Next”.

3. If your “HOME” environment variable is not set (or you don’t know what this is):
    1. Open command prompt (Open Start Menu then type `cmd` and press `Enter`
    2. Type the following line into the command prompt window exactly as shown:

        ```
        setx HOME "%USERPROFILE%"
        ```

    3. Press `Enter`, you should see `SUCCESS: Specified value was saved.`
    4. Quit command prompt by typing `exit` then pressing `Enter`

This will provide you with both the software git and the Git Bash program.

## Git Bash (Windows only)

One great benefit for installing git for Windows users is that it automatically comes with Git Bash, a command line interface which has UNIX-style commands. This is useful because *most* of the solutions on the internet use this type of command, as opposed to PowerShell or other Windows-specific options. For example, in Git Bash you can use the command `ls` instead of `dir` to print the contents of a directory. Note that to get your Git Bash to recognize your conda environments , you should run:

```
conda init bash
```

Now you can use this instead of the Anaconda Prompt and you will be able to use UNIX-style commands!

# Authentication with SSH

In order to connect to your repositories on GitHub, we need to properly authenticate ourselves. When we want to visit repositories at github.com, we need to log in via the web browser. The same holds if we want to interact with these repositories remotely with Git from our computer. This means that every time we make a commit, pull, etc., we need to provide our password, which becomes very tedious. Alternatively, if GitHub knows it is indeed us that are making the changes to the repository, we can remove the need to provide a password.

This is accomplished with SSH (Secure Shell Protocol), and this section will show you how to do this in two main steps:

1. Creating an SSH key on your computer, and
2. Adding the SSH key to your GitHub account.

You can also refer to the GitHub Documentation for more information.

1. Open up a terminal window (Git Bash on Windows). Next, type (or copy and paste) the following command:

   ```
   ssh-keygen -t ed25519 -C "GitHub"
   ```

2. You will get output that looks something like this:

   ```
   Generating public/private ed25519 key pair.
   Enter file in which to save the key (/Users/<user>/.ssh/id_ed25519):
   ```

   Press enter to use the default path (the one between parenthesis). Keep note of this path, as we will need to visit it later on.

3. You will now be asked to enter a passphrase. You can give one, but you will be asked for it every time you make a commit, so it's better to leave this field empty and use this key only for GitHub. After the command is done executing, it will generate two files: `id_ed25519` and `id_ed25519.pub`. The `.pub` file is the public part of the SSH-Key which we need.
4. Go to GitHub and log in. Click on your profile icon on the top-right of the page, then go to Settings. In the menu on the left, go to SSH and GPG keys.
5. Navigate to the `id_ed25519.pub` file that we generated in step 2. Open the file with a text editor (Notepad on Windows, textEdit on Mac). Copy its contents and paste it in the 'Key' field of SSH and GPG keys menu. Give the key a Title and click 'Add SSH key'.

   Note: on macOS, the `.ssh/` folder is hidden by default, so you won't be able to find it in Finder. To show hidden files and folders, press **Command + Shift + .**.

   At this point, your screen should look like this:

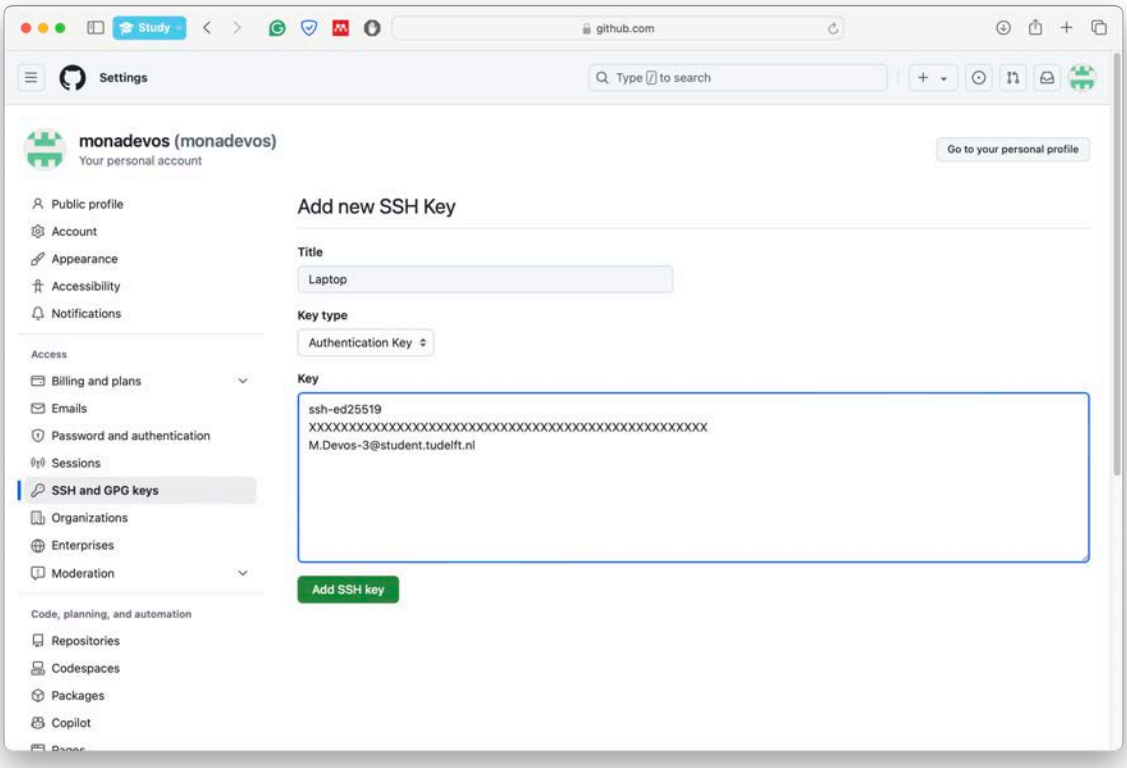



*Fig. 20* Add SSH Key to GitHub Account.

*Note the key contents will be different for you: just make sure you copy the contents of the file* `id_ed25519.pub`.

6. Now test the connection. Go to the terminal (Git Bash on Windows) and type:

```
ssh -vT git@github.com
```

You should get a huge output, but somewhere at the end of that output it should say something like:

```
Hi <username>! You've successfully authenticated, but GitHub does not provide shell a
```

# Setting up Git

Before we start, we first need to locally set up the git environment, so that when we make changes, we can be identified as the author of those commits. Open git-bash on your computer and type the following lines in the command line interface and use your name and email address between the quotation marks.

If you will be uploading your progress on a university instance of GitHub/GitLab, it is recommended to use your student email when committing.

```
git config --global user.name "Kiril Vasilev"
git config --global user.email "k.v.vasilev-1@student.tudelft.nl"
```

Console example in Fig. 21:

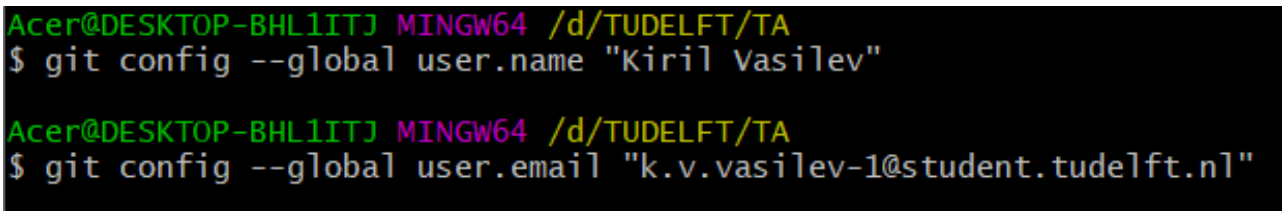



*Fig. 21* Setting up name and email to use for git

Note that in the commands above you **do not need** to type the `$` sign.

# Git

This Chapter assumes you have already installed the git software, as described here.

In practice we refer to the repository stored on our computer as the **local repository.** This is where we typically spend most of our time working on our code, debugging it and running analysis.

The repository on GitHub is our **remote repository.** This can be considered the backup of the files in our repository. When working with multiple people, it can also be considered the most current version of the project. For example, you may be working on improving some plot, whereas a colleague is updating the Python script running your model. You will use the *local repository* to develop your particular task, and the *remote repository* to collect everyone elses work and deploy it to your local computer (if you need to use it there). In education, the remote repositories are used to share your work with your teacher.

# Git on GitHub

This workflow will focus on using Source Control on GitHub: making commits to the remote repository using the online GitHub IDE

> **Note**
>
> The examples in this page will be illustrated using an arbitrary public repository set up by a MUDE TA on GitHub called sandbox-public and a fork called similarly. You can follow along by creating your own repository in your GitHub.
>
> If you create your own repository, keep the box checked to initialize with a `README.md` file, which will look identical to that in the examples in the following pages.

# Making Commits to the Remote Repository (Web IDE)

## Contents


- What is the web IDE?
- Edit a file
- Make a commit


This page will show you how to make commits to the remote repository directly from the web browser. If you only have small changes to make to a small number of files, this can be faster than doing it with your local repository.

## What is the web IDE?

IDE stands for integrated development environment. It is a software that makes it easier to develop code and software, for example, by highlighting code syntax, or including command line interpreters to run the code. You may be familiar with other IDE's such as Matlab, Spyder or Visual Studio Code (VS Code). In fact, the IDE integrated in the GitLab website is actually VS Code.

## Edit a file

To edit a file via the web browser we need to open it in the web IDE. First click on the file in the GitHub repository, then click on the arrow next to the pencil icon and select open with 'github.dev' (Fig. 22).

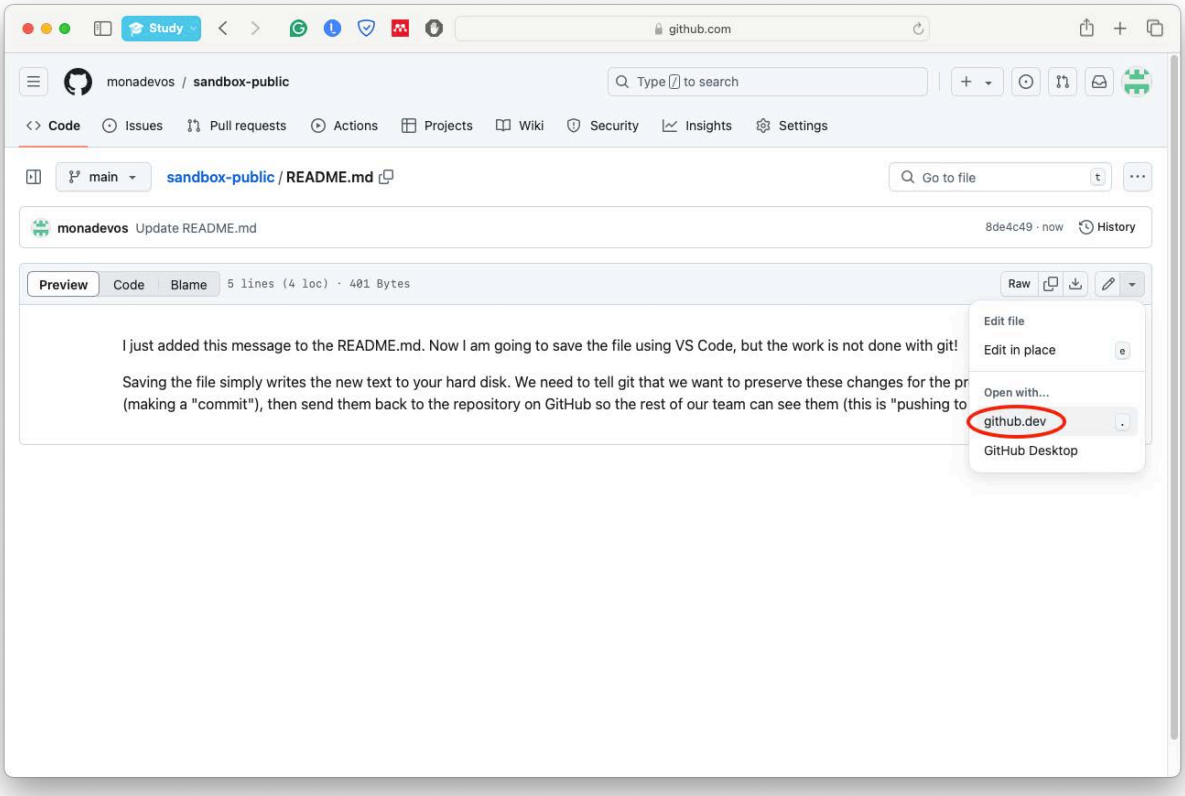



*Fig. 22* Button to edit file in GitHubs's Web IDE.

Once this opens, you will see a familiar editor (VS Code). The file will be visible and you can edit the text as needed (Fig. 23). Note that if more files were present in this repository, you would be able to see them in the left panel, and clicking on them would open them in additional tabs next to `README.me` in the editing panel.

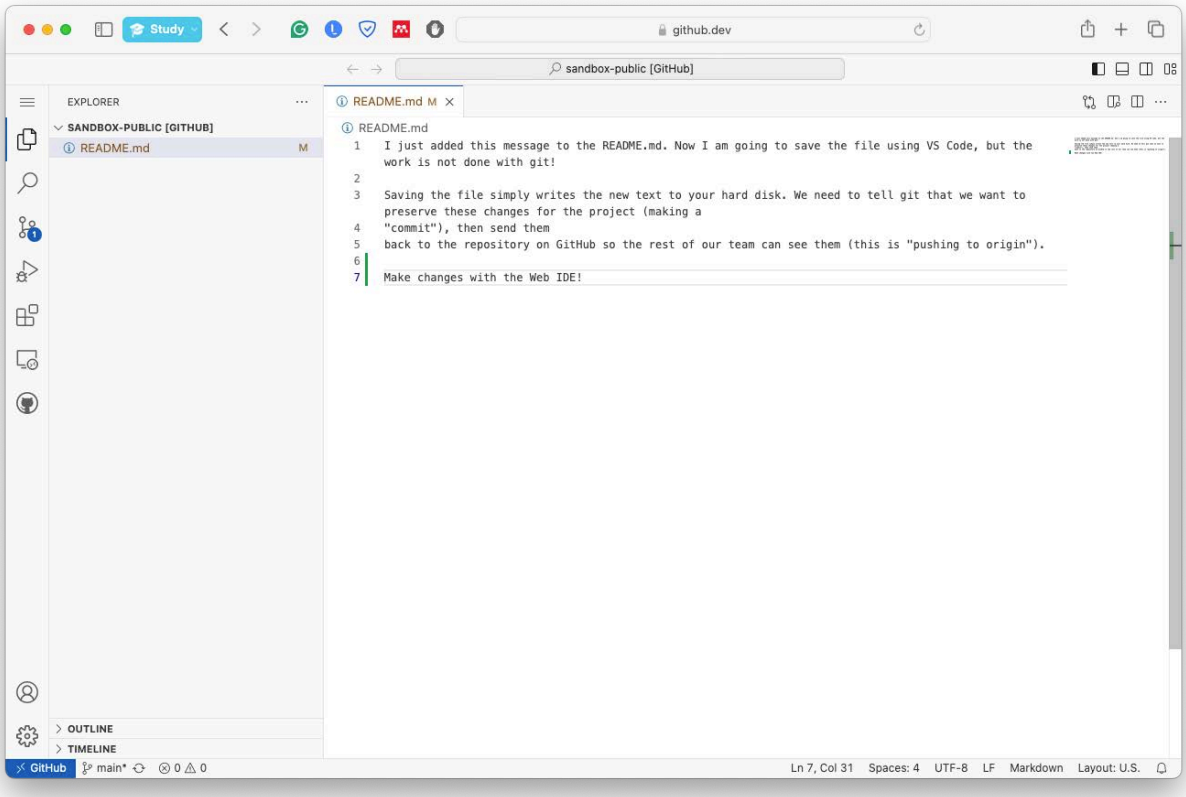



*Fig. 23* View of Web IDE: editing `README.md`.

Make your changes to the file as needed. Note, however, that there is no “save” button in the Web IDE: changes are automatically preserved as long as you keep the tab open.

# Make a commit

At this point we are ready to make our first **commit**, but what does that mean? A **commit** is the way that we tell the git software to make a record of the change that we just made to the file. That’s it!

Making a commit is very easy via the git Source Control panel, which you can access by clicking the branch-like icon on the far left side of the browser window ( Fig. 24, just below the magnifying class icon). As illustrated in the figure, all changes are listed in the left panel, and you can easily edit the commit message and make a commit using the blue button “Commit & Push”. The commit message can be extremely useful for reviewing changes, however, especially when working with multiple people. *It is important to write concise (limit it to 50 characters max), yet descriptive commit messages.* It could be useful to write your commit message such that it completes the following sentence: *This commit will….*

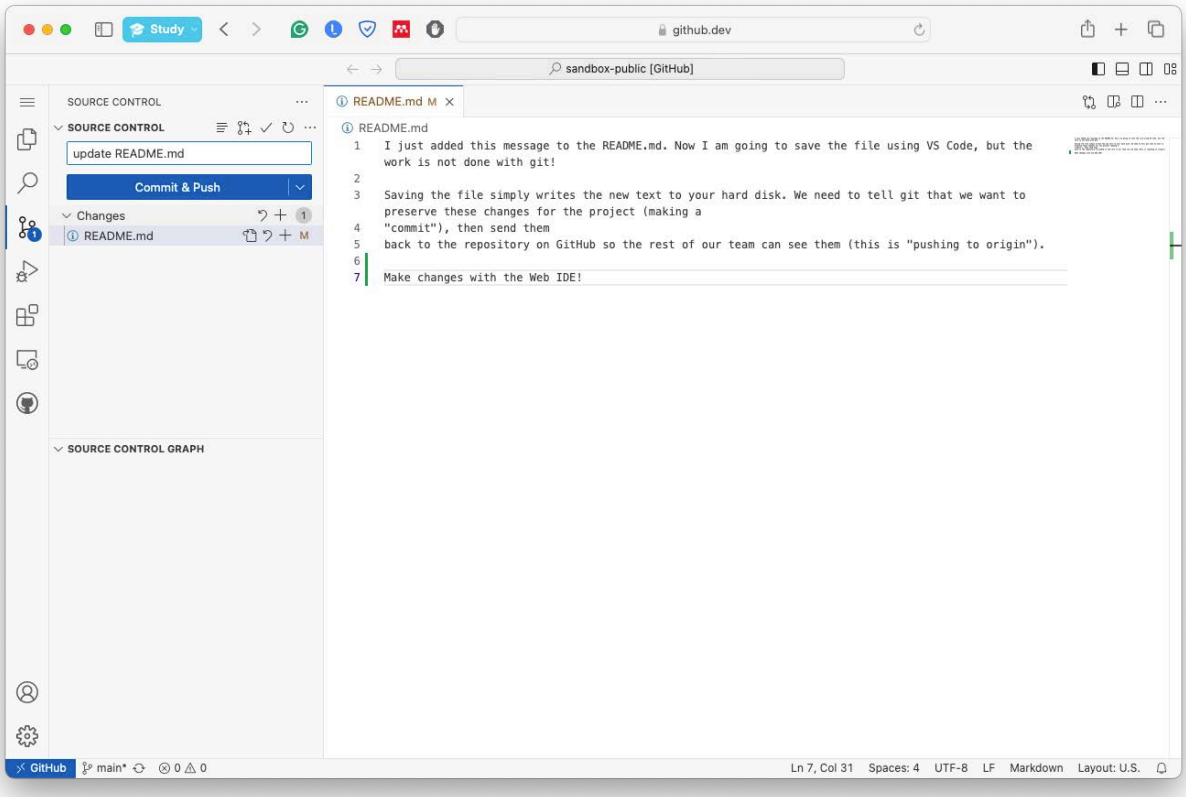



*Fig. 24* Making a commit in the Web IDE.

Once you have made the commit, you can return to the main page of the repository in GitHub to see the changes, visible in Fig. 25.

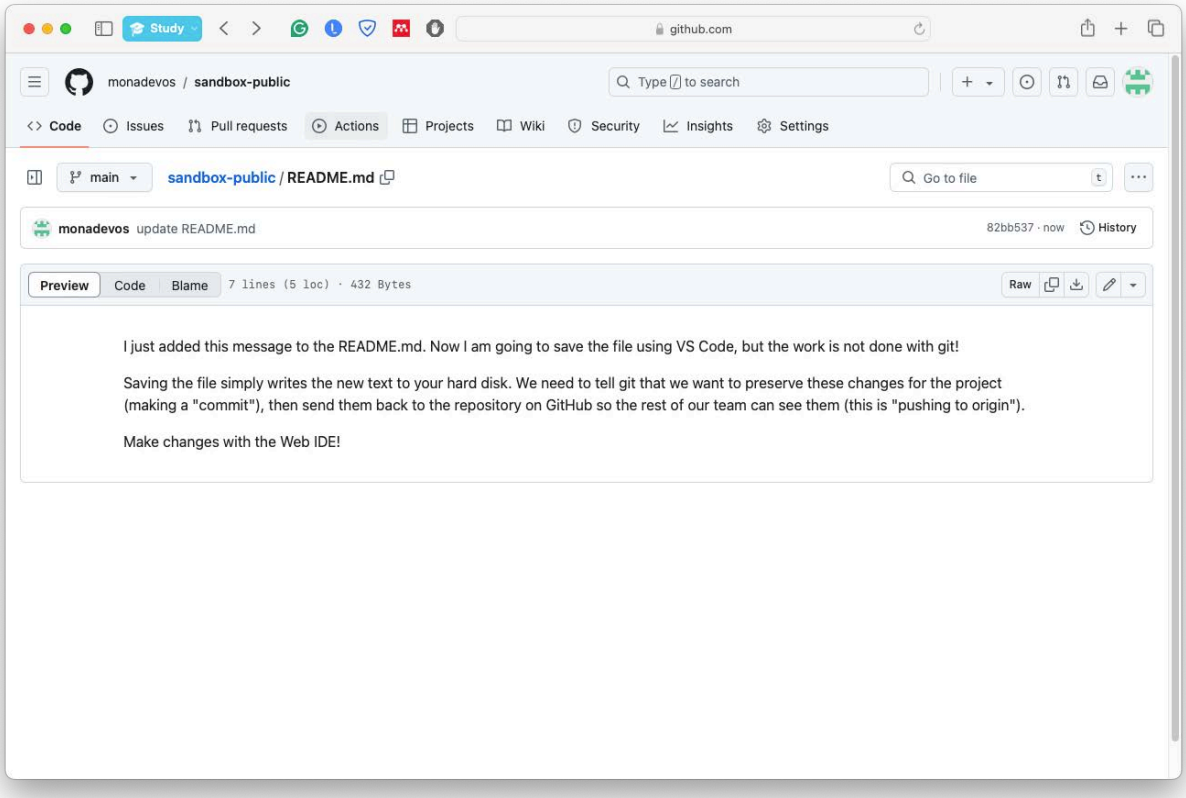


*Fig. 25* Confirmation of the Web IDE commit.

# Git in VS code

This workflow will focus on using Source Control in VS Code in combination with GitHub:

1. Cloning a remote repository from your TU Delft GitHub
2. Making commits to the local repository and Pushing your commits back to the remote repository
3. Fetching and pulling commits from the remote repository to your local repository

> **Note**
>
> The examples in this page will be illustrated using an arbitrary public repository set up by a MUDE TA on GitHub called sandbox-public. You can follow along by creating your own repository in your GitHub.
>
> If you create your own repository, keep the box checked to initialize with a `README.md` file, which will look identical to that in the examples in the following pages.

# Cloning a Repository

## Contents


- Opening the Source Control tab in VS Code
- Get the SSH address
- Set the local path
- Creating the clone


This page will show you how to use Source Control in VS Code to **clone** a repository. **Cloning** is the process of duplicating not only the files to your own computer, but also the record of all changes that have been made to those files in the past (that's what the git software is doing!). Because the history of these files is preserved, even if we make changes to these files in one of the repositories, the git software will provide a way to update the other software some time in the future. This allows multiple people to work on the same files at the same time; or, you as an individual to work on the same files using several different computers (e.g., a work and home laptop).

## Opening the Source Control tab in VS Code

Upon opening VS Code, we see that the first thing that is needed is to clone a repository (Fig. 26). First, we have to get the address from our GitHub account.

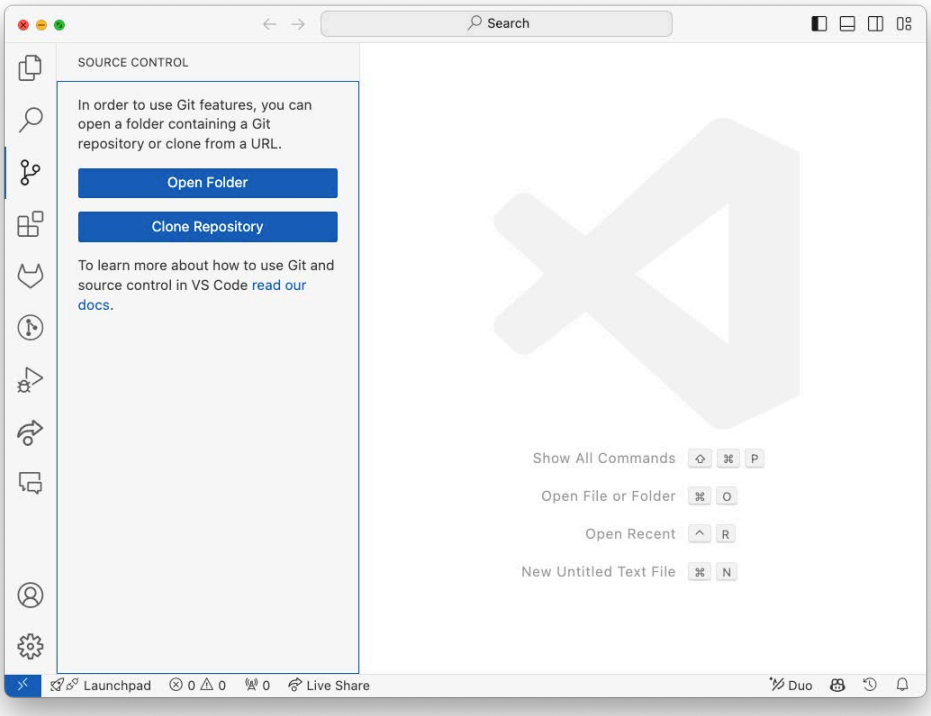



*Fig. 26* Opening the VS Code Source Control tab for the first time–we need to clone a repository.

Visit the repository you would like to clone to your computer on GitHub. As illustrated in Fig. 27, you can see a bright green button “Clone” on the home page of the repository.

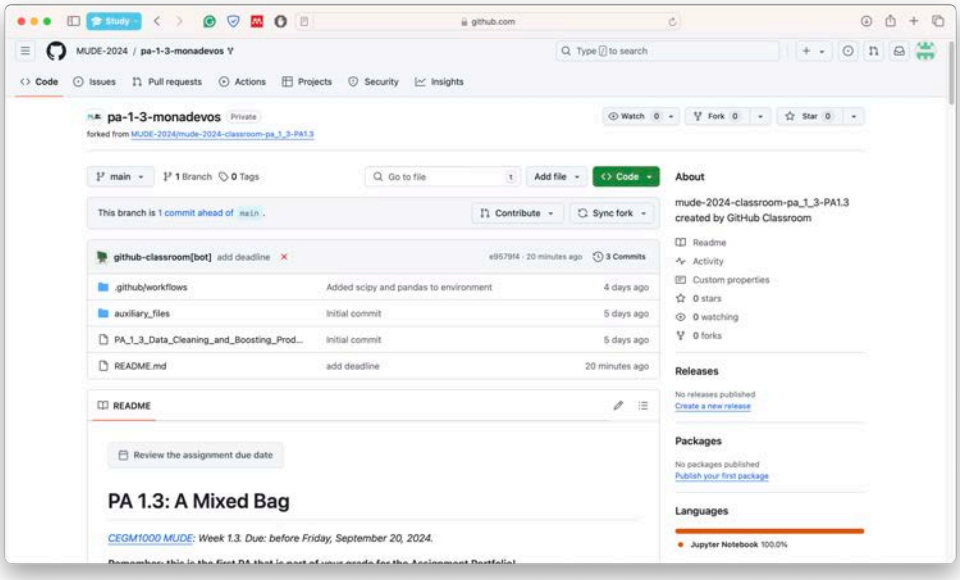



*Fig. 27* Home page of the repository in GitHub; note the “Clone” button.

# Get the SSH address

To clone the repository, we need the address–its location on the internet. There are several ways to do this, but the most secure is using SSH. As illustrated in Fig. 28, under “Clone with SSH,” click the icon to copy the URL to your clipboard, then go back to VS Code.

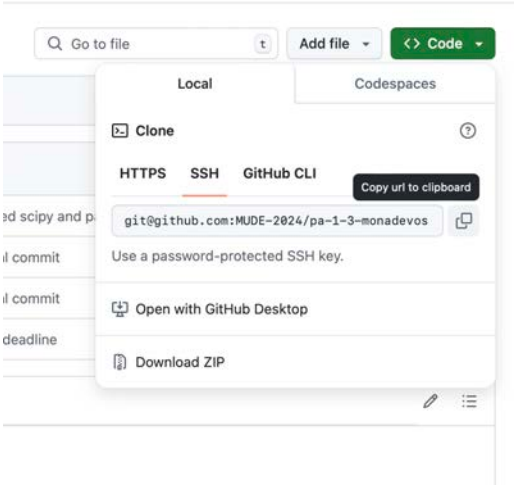



*Fig. 28* Copy SSH address of repository to your clipboard.

# Set the local path

Now you are ready to click the “Clone Repository” option in the Source Control tab of VS Code (`clone1`). Paste the address you copied from GitHub into the top bar and press ‘enter’ (as shown in Fig. 29). Then select the folder or location on your local machine where you would like the *local repository* to be stored (Fig. 30).

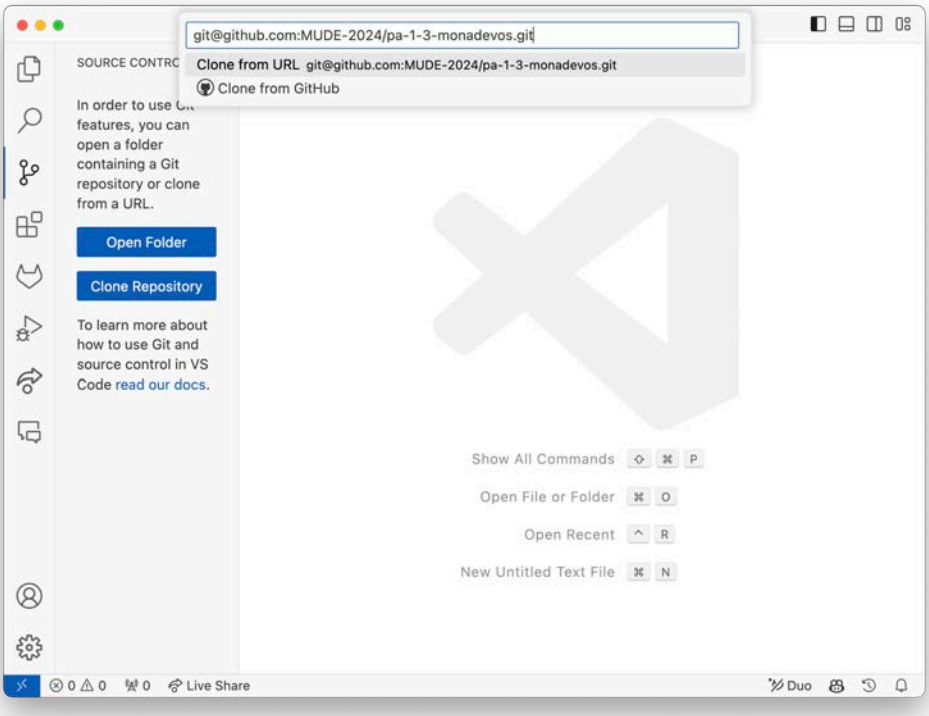



*Fig. 29* Enter the URL from GitHub and press 'enter'.

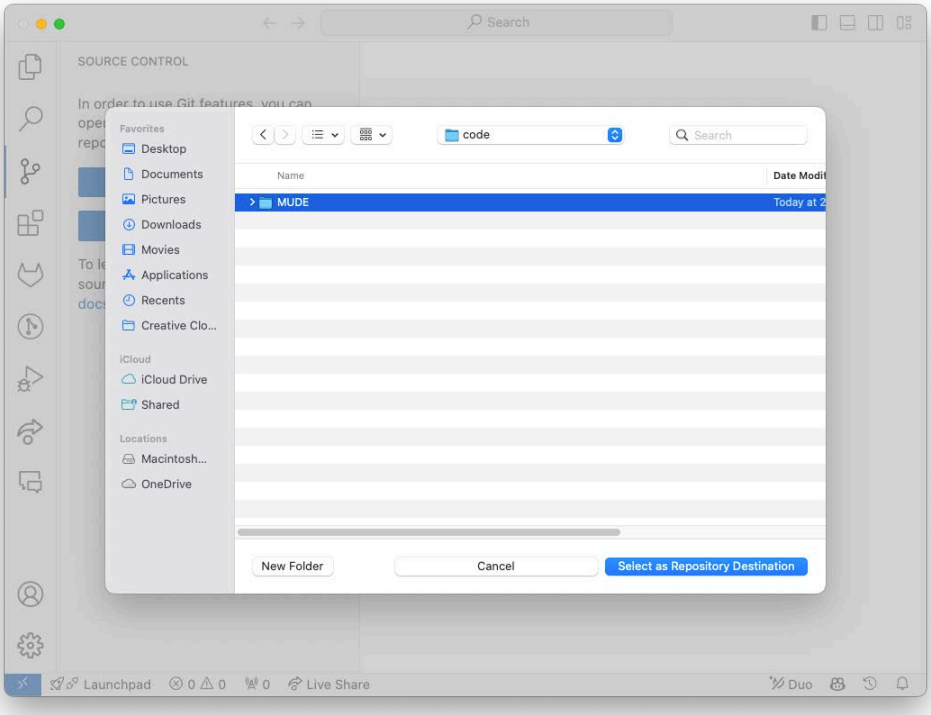



*Fig. 30* Browse to the location you want to store the *local repository*.

As you can see in Fig. 30, we have chosen to put the cloned repository in a directory in the place where we store all code, `code\MUDE`. You don't need to create a new folder for the repository (e.g., `pa-1-3-monadevos` in this example) because the process of cloning a repository will do that automatically.

**Where to put your local repositories**

We strongly encourage you follow these pieces of advice:

1. **Do not** store your local repositories in a location that is backed up using cloud software (e.g., OneDrive, Dropbox, etc). This often interferes with the functioning of git. Instead, we will push to the *remote repositories* to backup our work.
2. **Do not** store your local repositories in locations with spaces in the file path, especially on Windows. While there are ways to deal with this if it happens, you will save yourself trouble down the line if you avoid using spaces in your folder and file names.
3. **Do** store your local repositories for your project in an organized way. We advise creating a directory for all repositories you have, where each of the sub-directories would be a local git repository. Here is an illustrating of such a structure for your working directories:

```
.
├── Git_repos
│   ├── Repo_1
│   ├── ...
│   ├── Repo_2
│       ├── PA-04.ipynb
│       └── ...
│   └── ...
```

# Creating the clone

At this point you can create the local repository by clicking “Select Repository Destination,” which will start the process of downloading the files from GitHub to your computer at the location you chose for local path.

**Note**

If you were not successful in creating an SSH key and linking it to your GitHub account, this is when you will find out, as a message like this will appear:

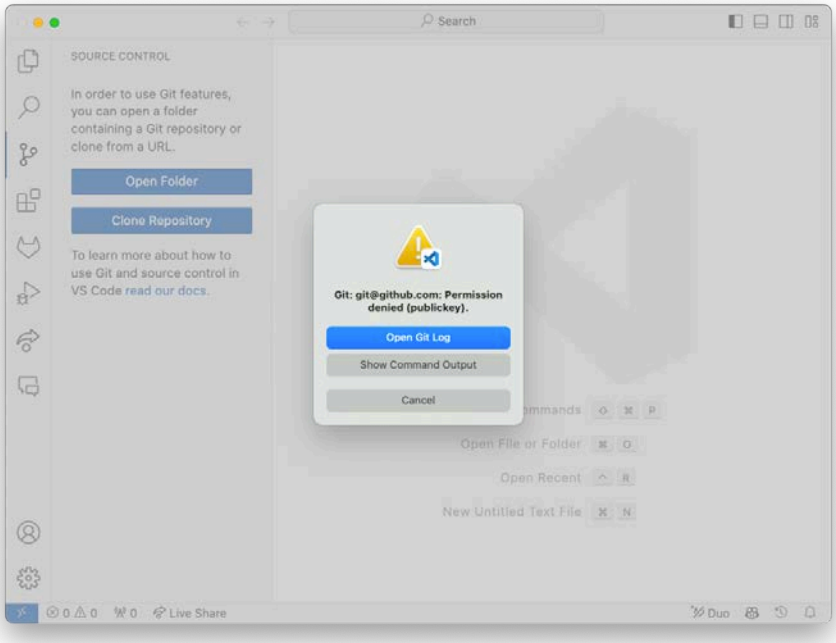



*Fig. 31* Error message due to failed SSH setup.

If this happens, go back to the SSH setup instructions

If you were successful in cloning the repository, you will see something similar to `clone7` and you can choose to open your *local repository*. Now we are ready to work on the files and preserve the changes by making our first local **commit**!

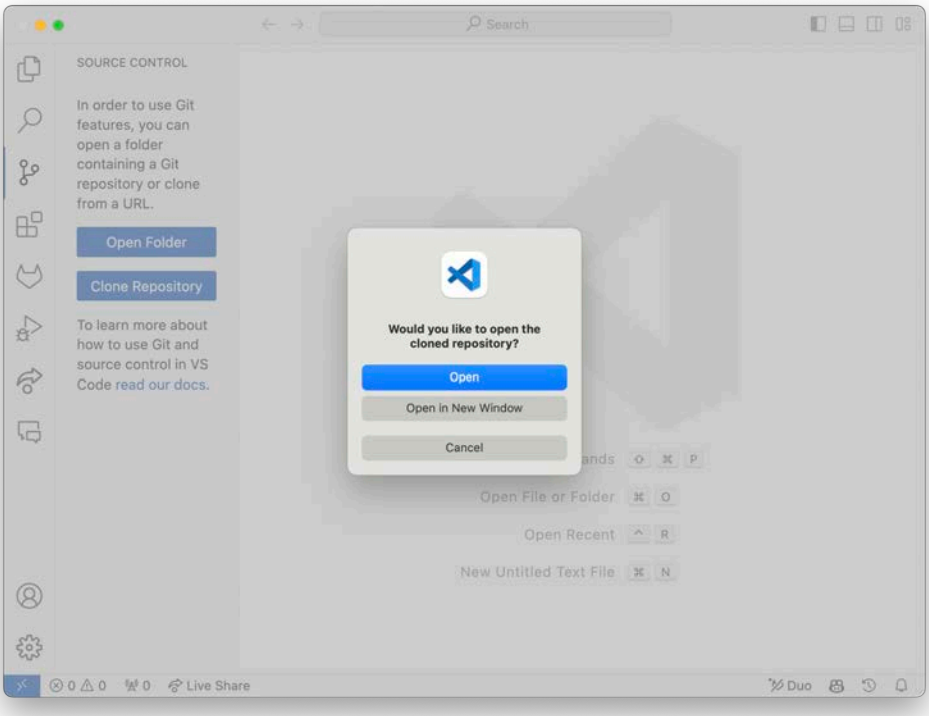



*Fig. 32* View of the local repository, successfully cloned from the remote on GitLab.

# Making Commits to the Local Repository

## Contents


- The remote repository
- Edit the local file
- Another commit
- Push commits to origin
- A closer look at our commits


Now that we have learned how to clone a repository, we will find out how to make changes to it and record by making **commits**.

## The remote repository

We begin by identifying the repository on which we would like to work, in this case `sandbox-public`; the starting condition is visible in Fig. 33 after cloning and opening your repository (see previous page). We can see that it began as a completely empty repository, except for the default README.md file that GitHub creates automatically.

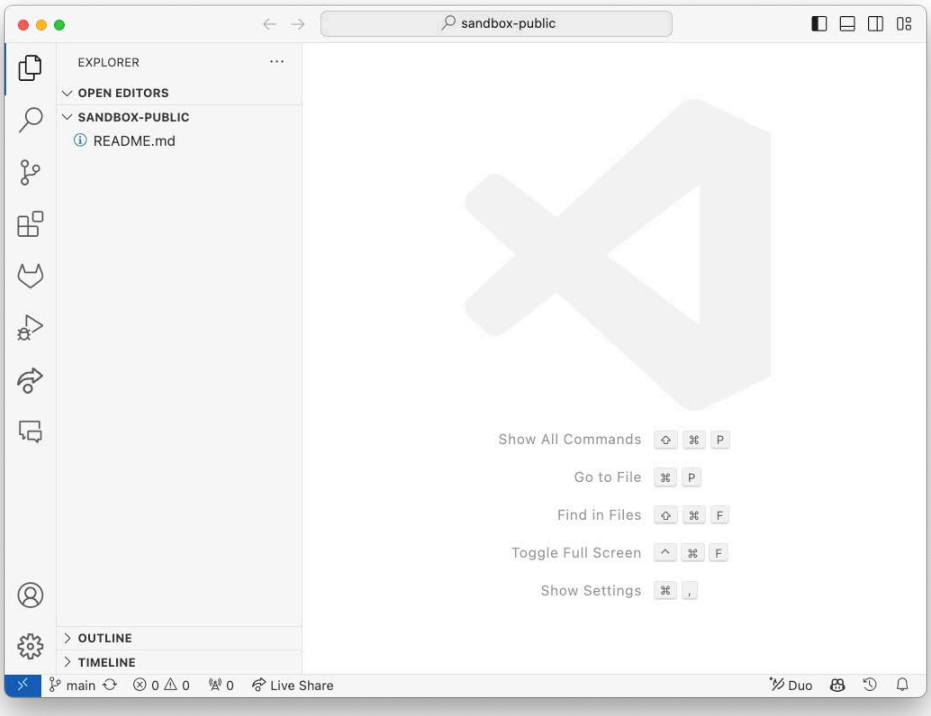



*Fig. 33* Starting position after cloning GitHub repository and opening it in VS Code.

# Edit the local file

Now that our repository is cloned, we are ready to get to work! First we will make a change to `README.md` in VS Code. As shown in Fig. 33, we have removed the title and added some relevant text. Save the changes to the file then click the Source Control tab.

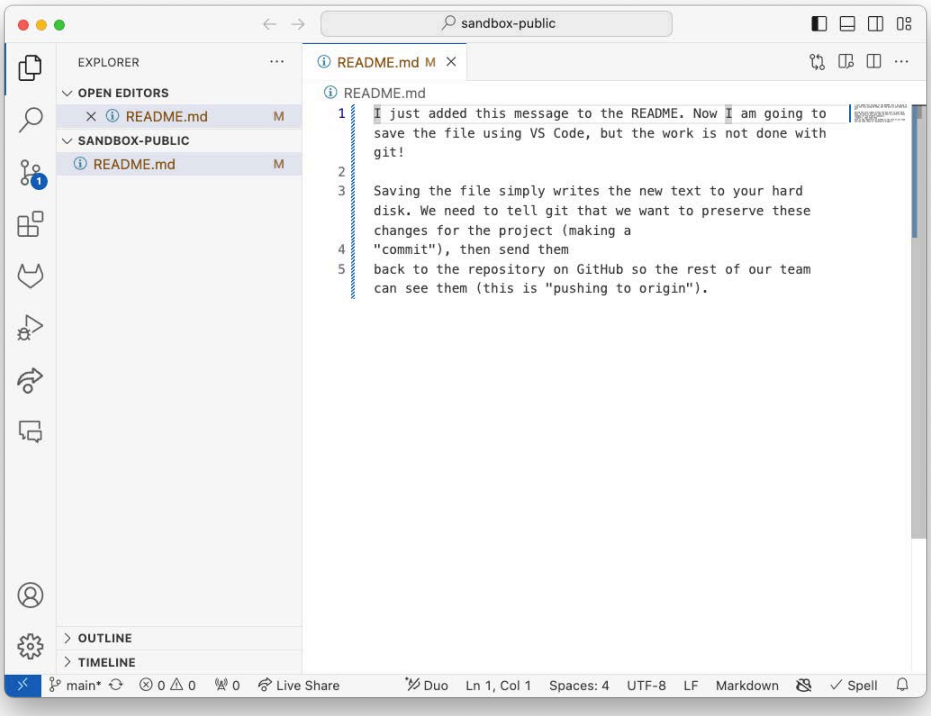



*Fig. 34* Example of changes to the original `README.md`` file in text editor.

> **Don’t forget to save changes!**
>
> It is important to remember to save the changes you make to files (either `CTRL`+`s` or by clicking the save button). This is how git recognizes that a file has been changed, after which it registers what those changes are.

Once the changes have been saved, open the Source Control view in VS Code. You will see the changed files listed under Changes.

If we click on the README under Changes we can see in Fig. 35, git has now registered the changes in our file. git uses a few text symbols to identify and describe the changes, for example, a `+` and `-`. VS Code highlights the changed lines in green and red, making it very obvious what happened.

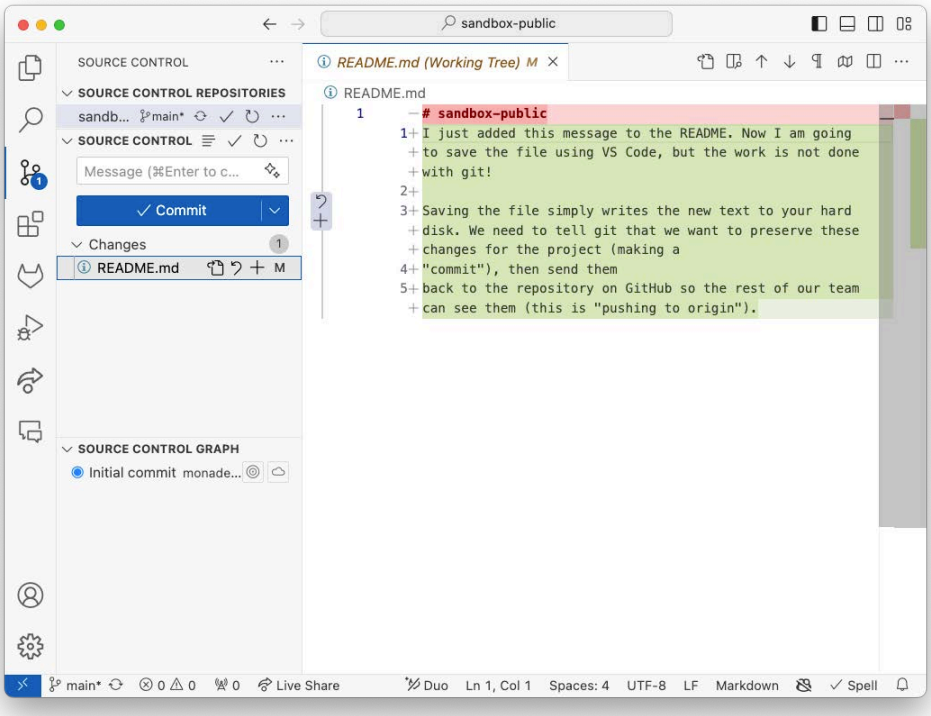



*Fig. 35* Summary of changes made to `README.md` in VS Code Source Control.

We now need to stage the changes that we wish to commit, either click the + icon next to the file or use the Stage All Changes option if there are multiple files you wish to commit. Staging refers to the process of preparing changes to be committed. When you stage changes, you are essentially marking them to be included in the next commit. This allows you to selectively choose which changes to include in a commit, providing more control over your project's history.

Fig. 35 also illustrates the space above the "Commit" button where you can provide a commit message.

> **Note**
>
> A local commit can record changes made to more than one file at once–we are only editing one file at a time now, to keep things easy to visualize.

# Another commit

Because the change to our `README.md` was so large, it was difficult to see all the changes. Let's try making another one. Using the change summary in Fig. 36, can you tell what happened to the file?

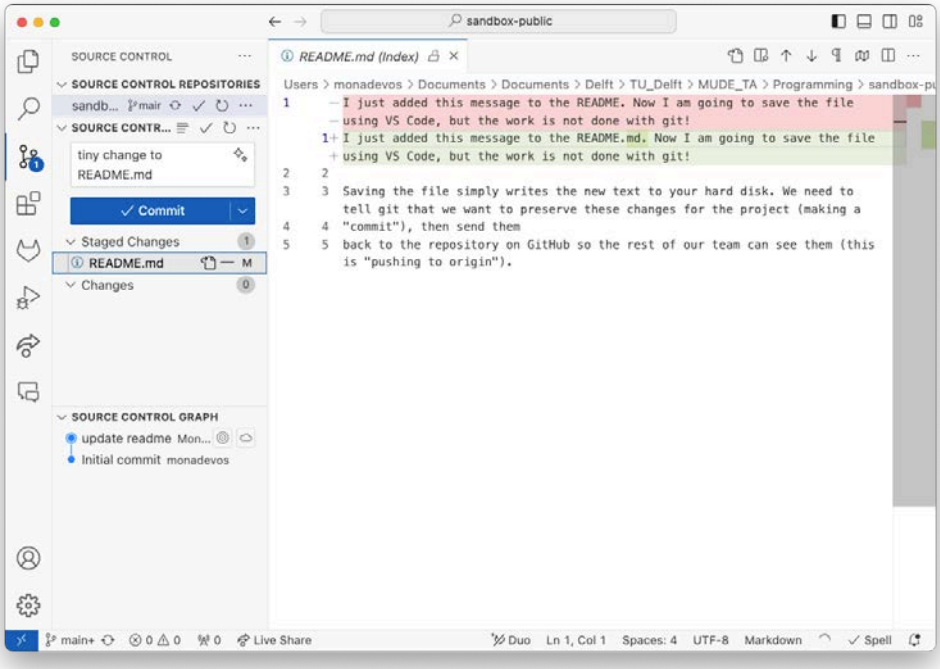



*Fig. 36* A small change to `README.md`.

**See the change here**

All we did was add the `.md` to the end of `README`.

README.md

```
1   I just added this message to the README.md. Now I am
    going to save the file using VS Code, but the work is
    not done with git!
```

*Fig. 37* The small change to `README.md` revealed.

`commit4` also illustrates the staged changes as well as the commit message.

# Push commits to origin

Now that we have recorded our commits with git, there is one last thing to do: update the *remote repository.* We can do this by **pushing** the commit, using the blue “Sync Changes” button shown in Fig. 38.

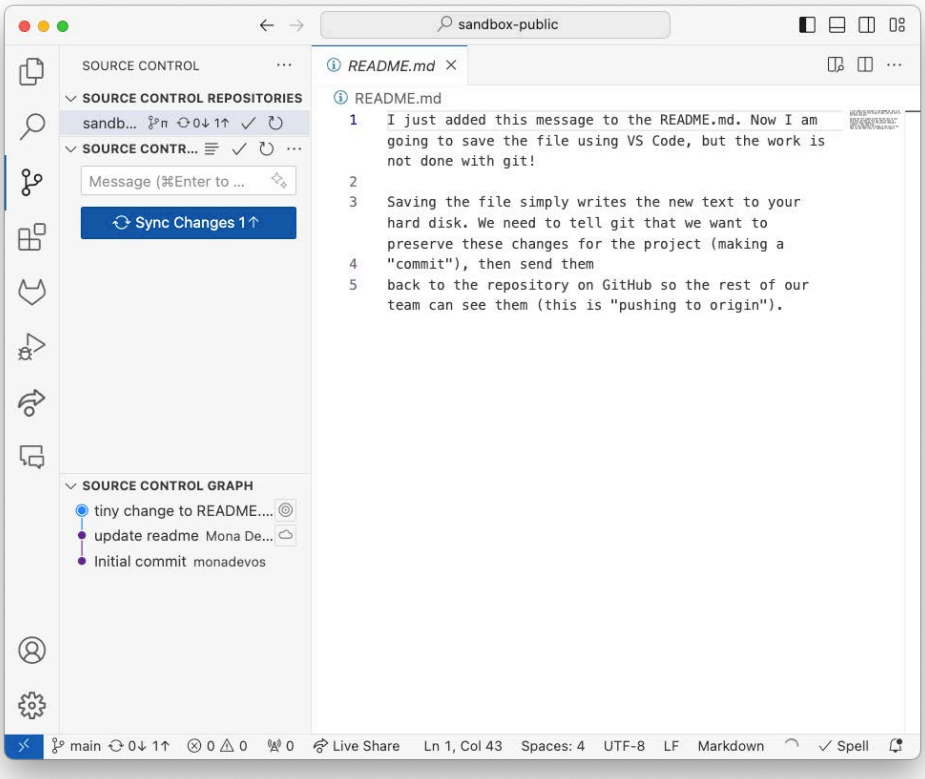



*Fig. 38* Pushing the commits to origin, our remote repository, by syncing changes.

**What is “origin?”**

Origin is the name git uses to identify the original *remote repository* where our *local repository* was cloned from. In our case: the repository on GitHub!

# A closer look at our commits

Now that we have pushed to origin, it is easy to confirm that things worked. First, we can see that in VS Code Source Control once again shows no changes, as in Fig. 39.

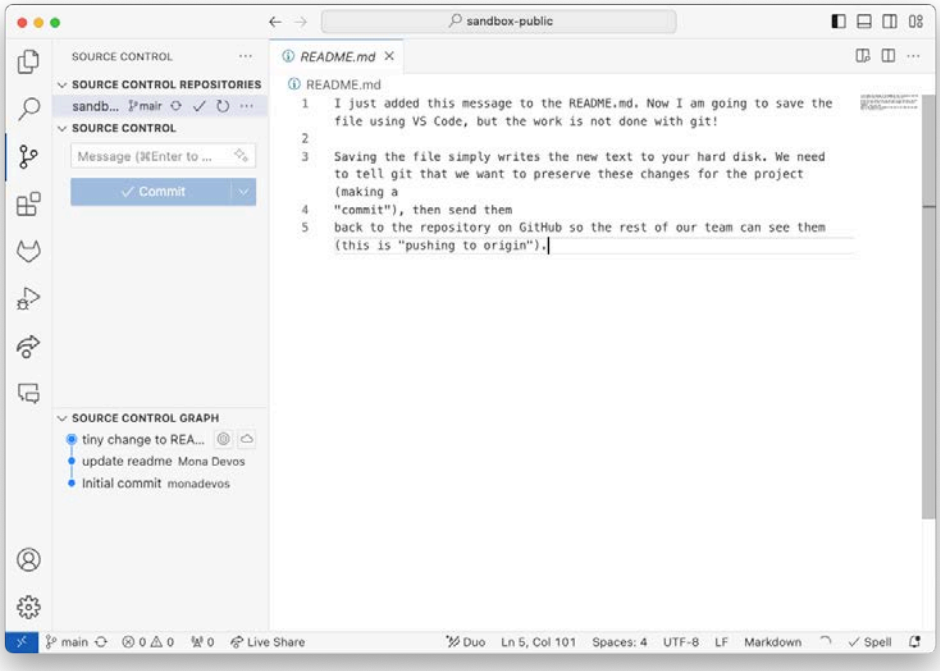



*Fig. 39* View of VS Code Source Control with no local changes.

In addition, we can check the remote repository directly. Visiting the home page of the repository is the best way to check the current status. Fig. 40 illustrates how this paged looked just after we pushed our commit to origin. As you can see the screenshot was made very soon afterwards (the “now” message), the commit message is visible, and we can see the actual change in in the text of `README.md`.

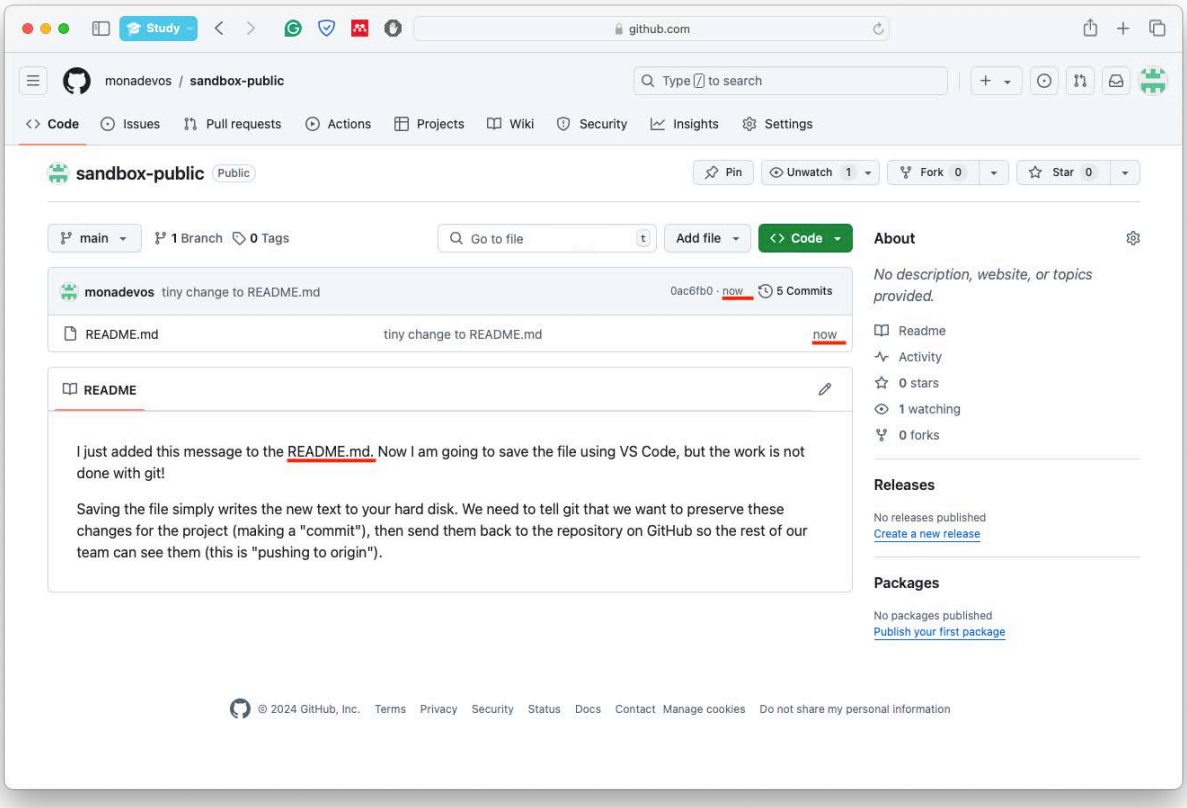


*Fig. 40* View of remote repository (origin) on GitHub just after pushing, illustrating changes.

By clicking on “commits” you can see all commits that have been made to the *remote* repository. Clicking on each commit will give an easy-to-read summary of the changes in each commit. For example, the first and second commits made on this page can be viewed here and here.

# Fetch and Pull from the Remote Repository

## Contents


- Definitions
- Fetch-pull from remote
- Confirm local changes


At this point we have seen how to clone a remote repository to make a local repository, make commits to either of them directly, and push from local to remote. There is one more critical skill for being able to use git in MUDE: getting changes on remote back to our local repository. We accomplish this in two stages: **fetching** and **pulling**.

Both of these terms refer to an action we take to get information about updates (e.g., new commits) that have been made to the remote repository (origin). We execute the commands from the local repository.

## Definitions

**Fetch** is when we ask git to connect to the remote repository and see what changes have been made. **Pull** is when we actually implement those changes locally and change the contents of the file on our computer.

**Why two steps?**

Sometimes we are making changes to the files on our computer and we just want to see if something new has happened. **Fetch** allows us to check this without making conflicting changes to our local machine.

> **Note**
>
> It is good practice to make it a habit of fetching first, then pulling once you are sure there are no conflicts.

# Fetch-pull from remote

Let’s first make a commit to the remote repository as explained in the first chapter

Now we can include the commit in our local repository that we have open in VS Code. We can either right-click on the three dots next to “SOURCE CONTROL” and “Fetch” (Fig. 41) or click on the little “Fetch” icon next to “SOURCE CONTROL GRAPH”.

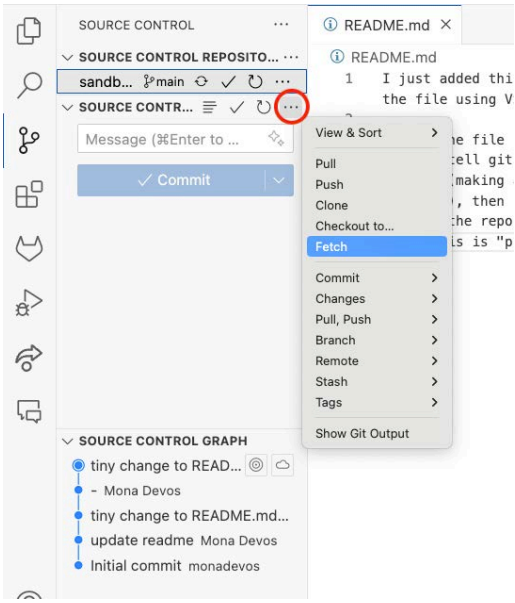



*Fig. 41* How to fetch from remote, option 1.

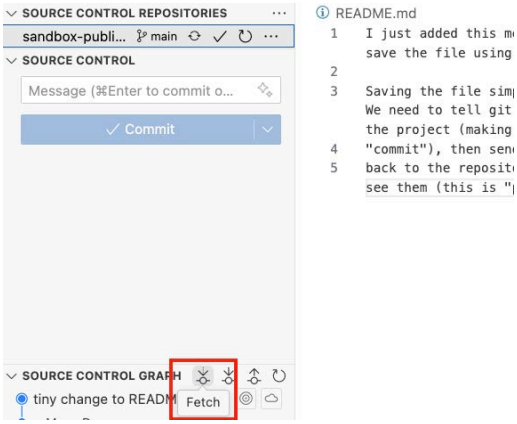



*Fig. 42* How to fetch from remote, option 2.

As you can see in (Fig. 43), under the Source Control Graph we can now see the new commits that have been made (purple) and we can click on one to see the specific changes that have been made. We can now pull all the new commits to our local repository by clicking “Sync Changes”.

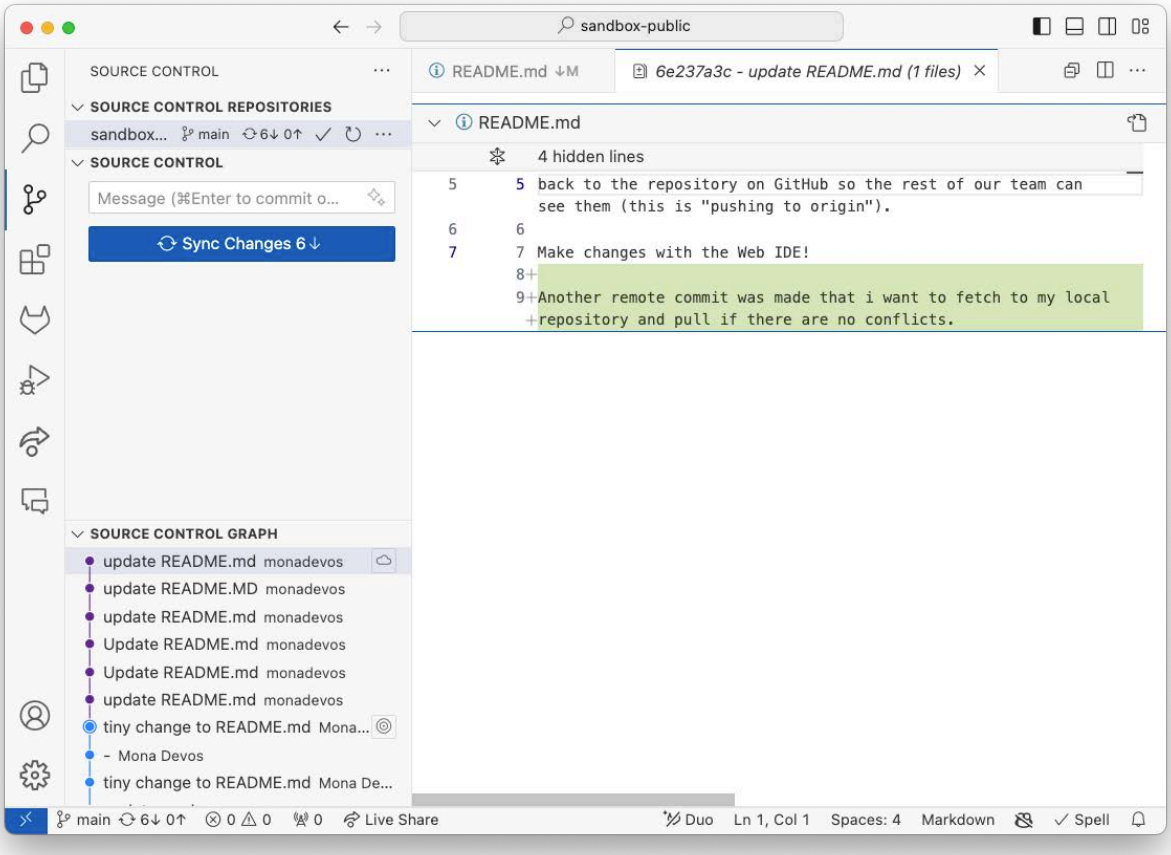



*Fig. 43* Check the new changes and pull the commits to the local repository

# Confirm local changes

Once the commit has been pulled to the local repository, we can open up the file to confirm. Fig. 44

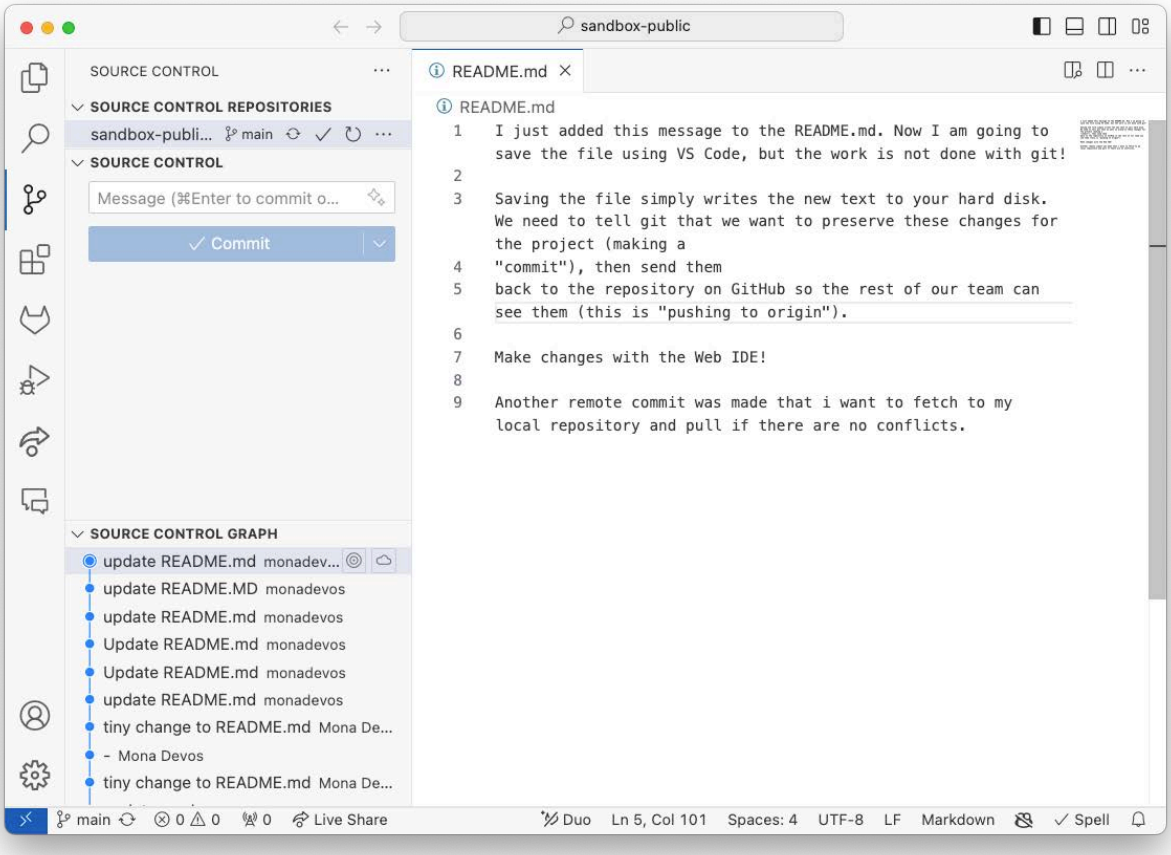



*Fig. 44* Local file `README.md` has the changes made with the Web IDE commit on the remote repository.

It's the commits we made on the remote repository!

# Week 1.5: Collaboration with Git!

This week you'll learn how to actually do version control with Git in collaborative sense.

In MUDE, you can do all assignments without this workflow. However, feel free to use it in your collaborative projects in MUDE or elsewhere!

# Branching and merging

## Contents

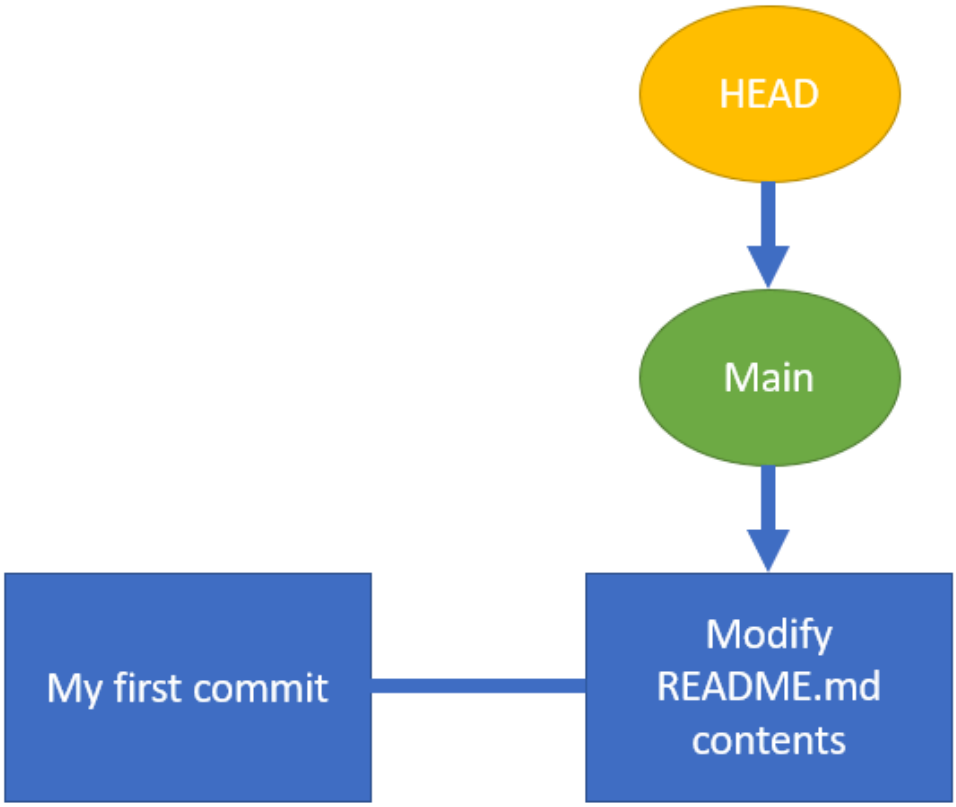

- What are branches
- Creating a branch
- Merging branches / pull requests
- Deleting branches
- Branches locally
- Collaborating with branches


## What are branches

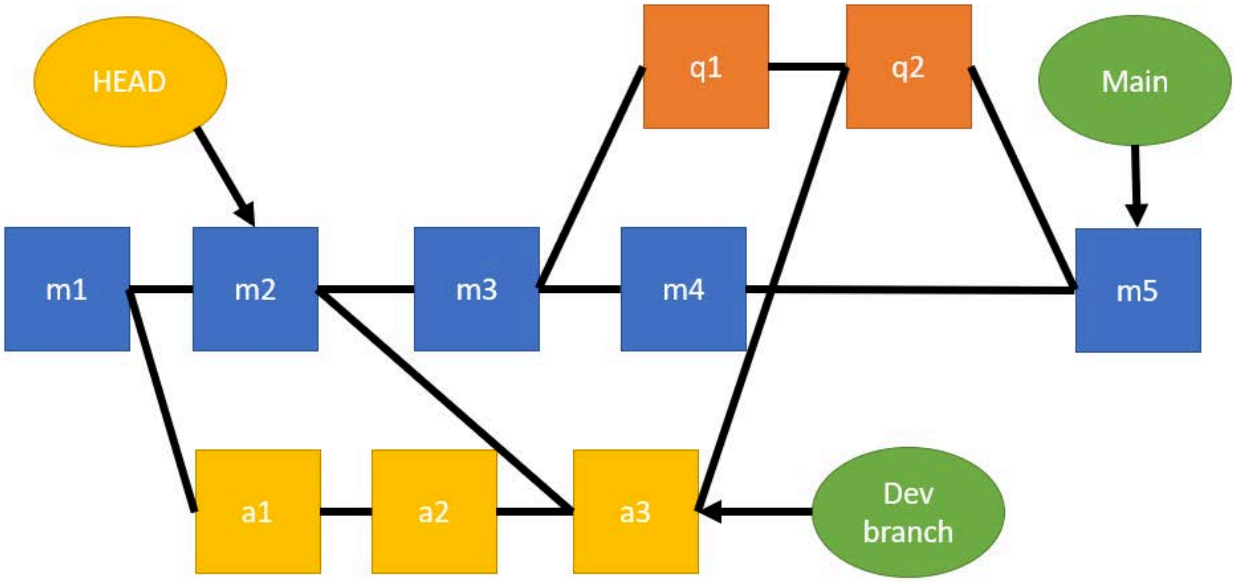
So far, we have only been making commits using the `main` or `master` branch, however, in practice, you will work on a separate branch, whose progress you will later merge with `main`.

Commits in git have a graph structure, where every node is a commit and edges represent the transition (flow) between commits. Branches can be thought of as pointers to commits, whereas HEAD points to the current commit that we are at. When you switch between branches, you can think of HEAD as the most recent commit on that branch.

The graph below shows the commit history of a repo with two commits:

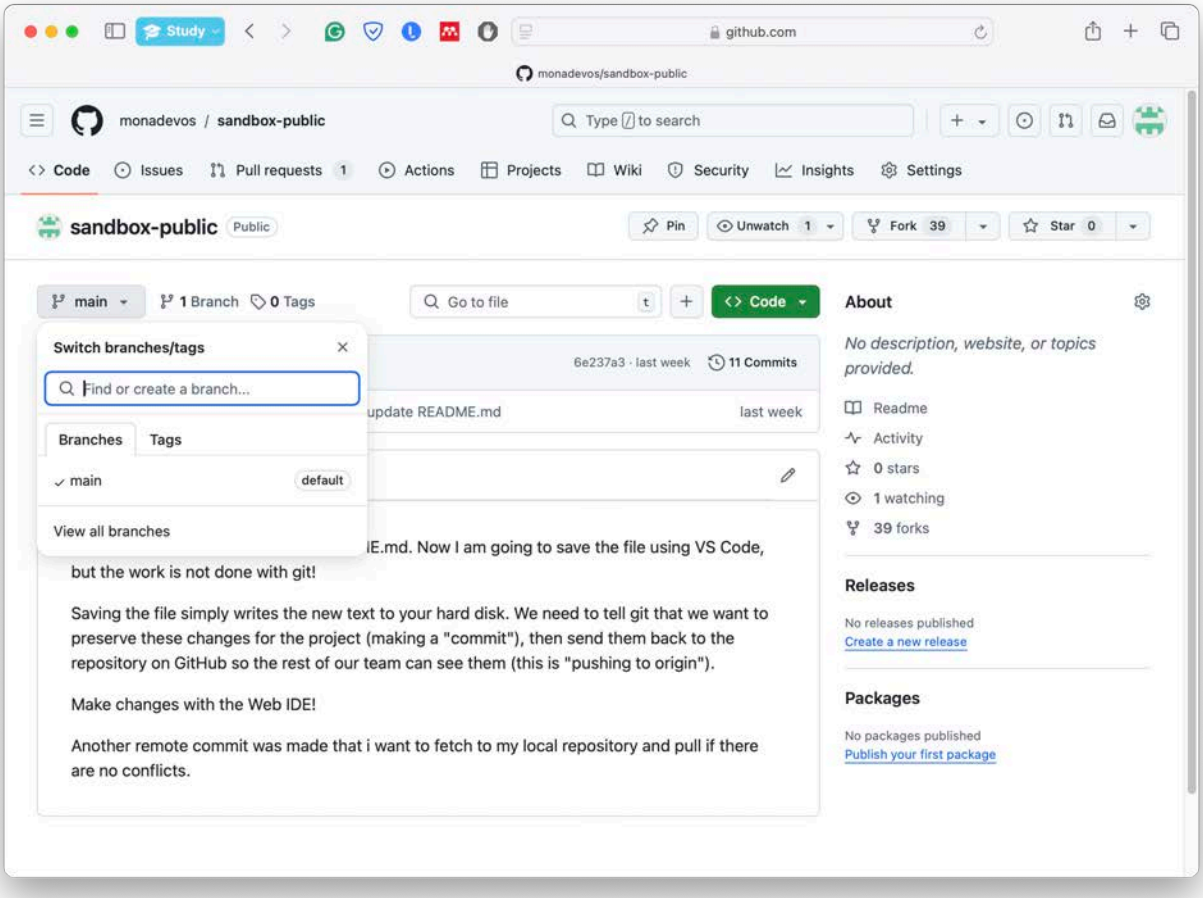



The main advantage of version control is that it allows developers to work together in parallel. During projects, you will be working on “feature” branches and separating the work to review and merge it later. A common graphical structure of commits is shown below, where we have developers working on 3 separate branches and merging their work when necessary. This separation offers flexibility, parallelization of work, and offers more control over the development process.



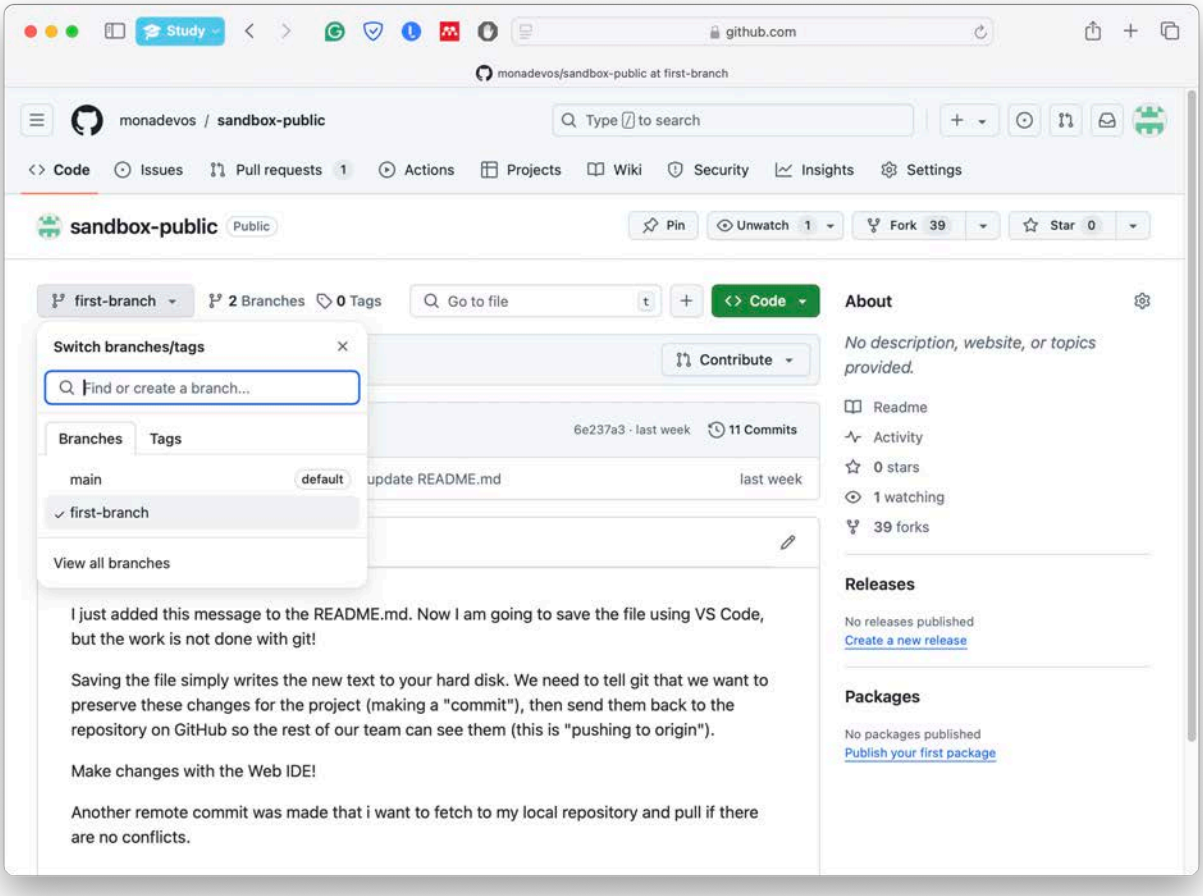
# Creating a branch

First, let us check on what branch we are currently at. On GitHub, navigate to your repository and click on the “Branch: main” dropdown to see the list of branches. The branch with a checkmark is your current branch.




Let us create a new branch from `main` on GitHub, which we will call `first-branch`. Click on the “Branch: main” dropdown, type `first-branch` in the text box, and click “Create branch: first-branch from main”.

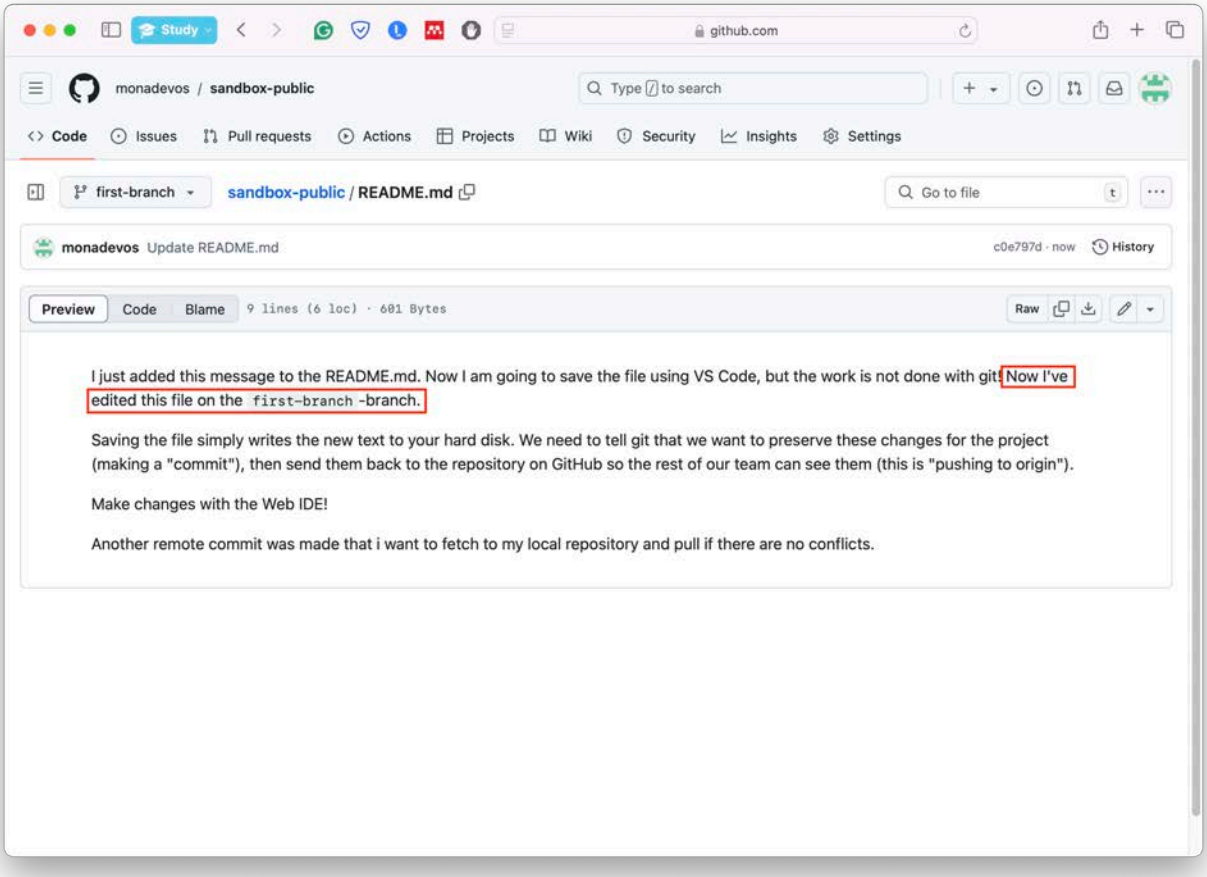
We can move to the newly created branch by selecting it from the branch dropdown.




Let us modify our `README.md` file again and add a sentence to the first line of it. Navigate to the `README.md` file, click the pencil icon to edit it, and add your content to the first line:

```
Now I've edited this file on the `first-branch`-branch.
```

We will commit the new changes by filling out the commit message and clicking “Commit changes”.




# Merging branches / pull requests

Note that when we switch to `main`, the `README.md` file does not contain the changes we made on the other branch.

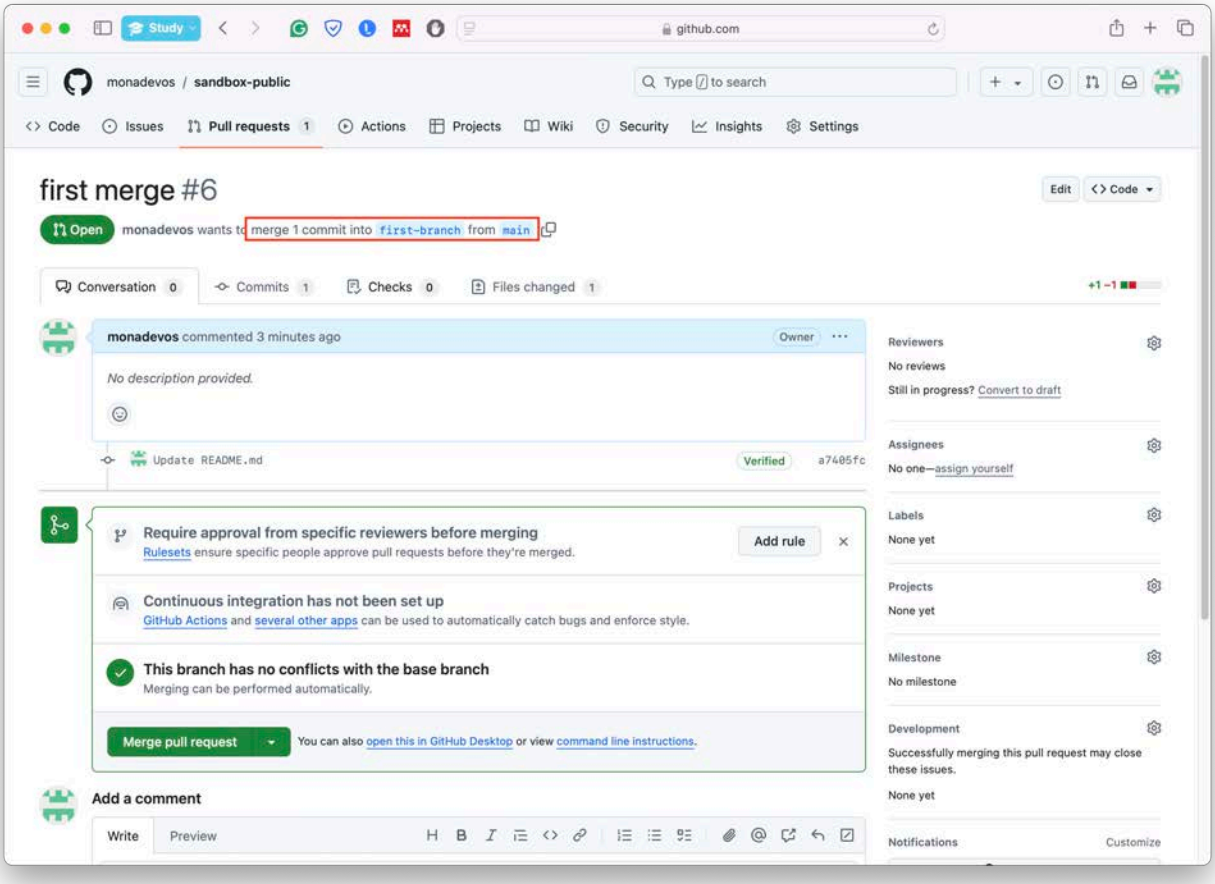
Before merging our progress from `first-branch`, it is advisable to merge `main` into it first. This is done because we regard the `main` branch as the “face” of our project. When somebody is looking at our repository, they are likely to check the things on `main` first. Moreover, if there are any bugs from merging, it is better to resolve them on our separate branch. Sometimes it happens that after merging your progress to the `main` branch, the code on the `main` branch is no longer working correctly.

Therefore, we will first go back to our branch `main`. Note that the edit we made is no longer visible. Let’s add a commit here too to `README.md` at the last line:

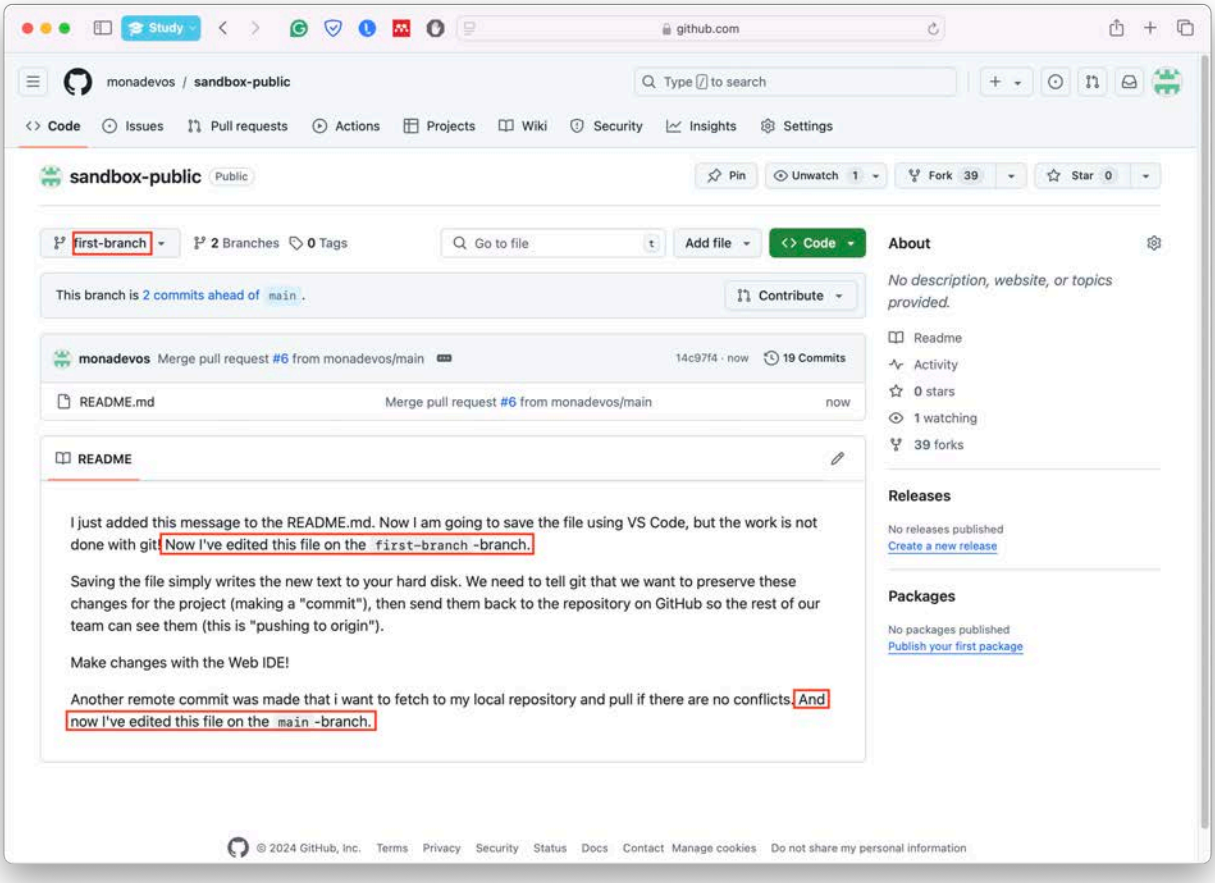
```
And now I've edited this file on the `main`-branch.
```

We can merge the `main` branch into `first-branch` by creating a pull request. On GitHub, go to the “Pull requests” tab, click “New pull request”, select `first-branch` as the base branch and `main` as the compare branch, and click “Create pull request”. We can check (as shown

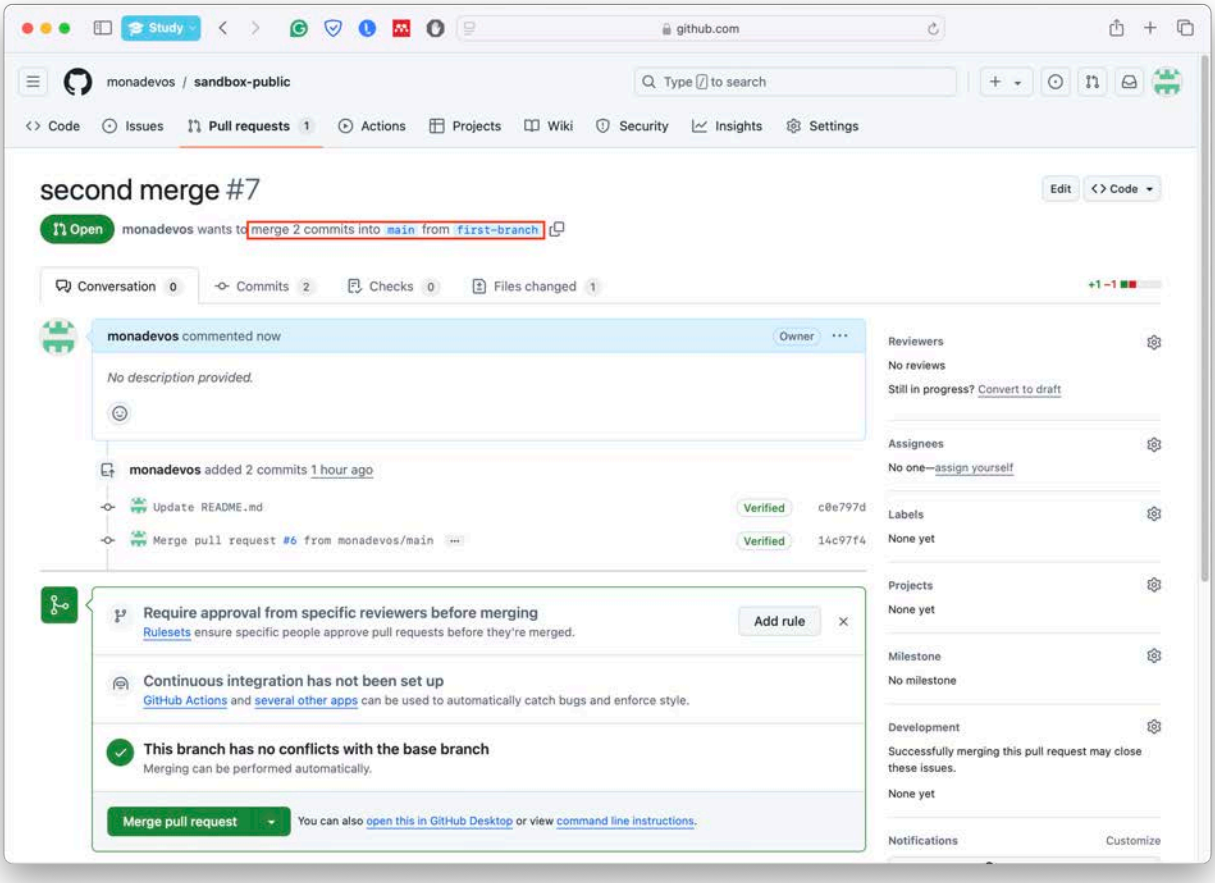
below) that there are no conflicts and then we click “Merge pull request” to merge `main` into `first-branch`.




We can now inspect the README.md again with the new changes. The progress of the other branch (`main`) has been moved to `first-branch` as seen below.

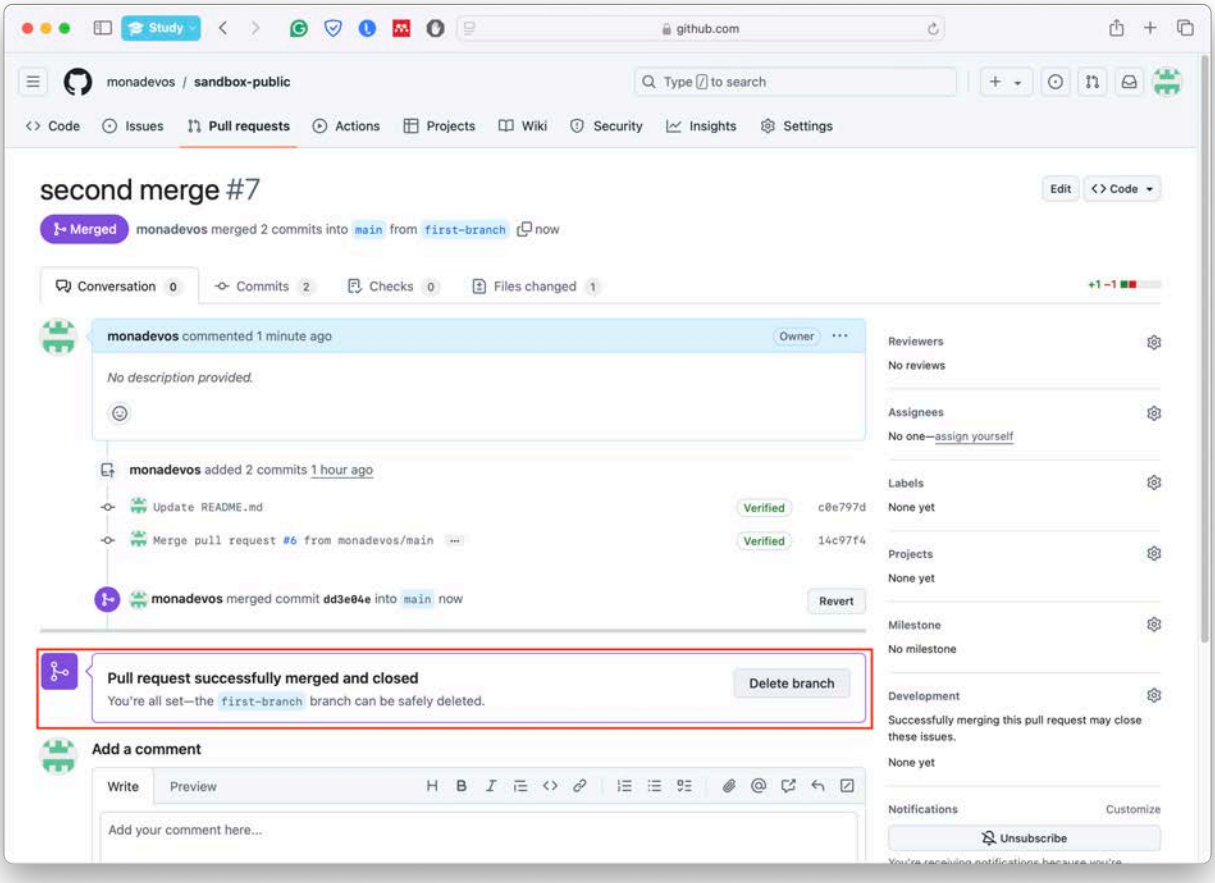



We will now return to `main` and merge `first-branch` into it with another pull request. Follow the same steps as before, but this time select `first-branch` as the compare branch and `main` as the base branch.




# Deleting branches

Branch `first-branch` has lived its use, so we can delete it as we do not plan on making new changes to it. This will not remove any of the commits made on it. It will only remove the pointer itself. Do this from the pull request page by clicking “Delete branch”.




# Branches locally

All of the above operations can also be performed locally using a Git client like VS Code. Here are the basics:

1. Check Current Branch: Open the source control panel in VS Code to see the current branch.
2. Create a New Branch: Use the branch icon in the status bar or the command palette (`Ctrl+Shift+P`) and select `Git: Create Branch`.
3. Switch Branches: Click on the branch name in the status bar and select the branch you want to switch to.
4. Make Changes and Commit: Edit your files, then use the source control panel to stage and commit your changes.
5. Merge Branches: Use the command palette to select `Git: Merge Branch` and choose the branch to merge from.
6. Delete Branches: Use the command palette to select `Git: Delete Branch` and choose the branch to delete.

# Collaborating with branches

When working together on a branch, it is important to coordinate with your team to avoid conflicts and ensure smooth collaboration. Here are some best practices:

- Keep in touch with your team about who is working on what.
- Frequently pull the latest changes from the remote branch to keep your local branch up to date. This helps in minimizing conflicts.
- Make small, frequent commits with clear messages. This makes it easier to track changes and resolve conflicts.
- Before merging changes, review pull requests carefully. Use GitHub’s review features to comment on specific lines and suggest improvements.
- If conflicts arise, communicate with your team to resolve them.

# Forking

## Contents

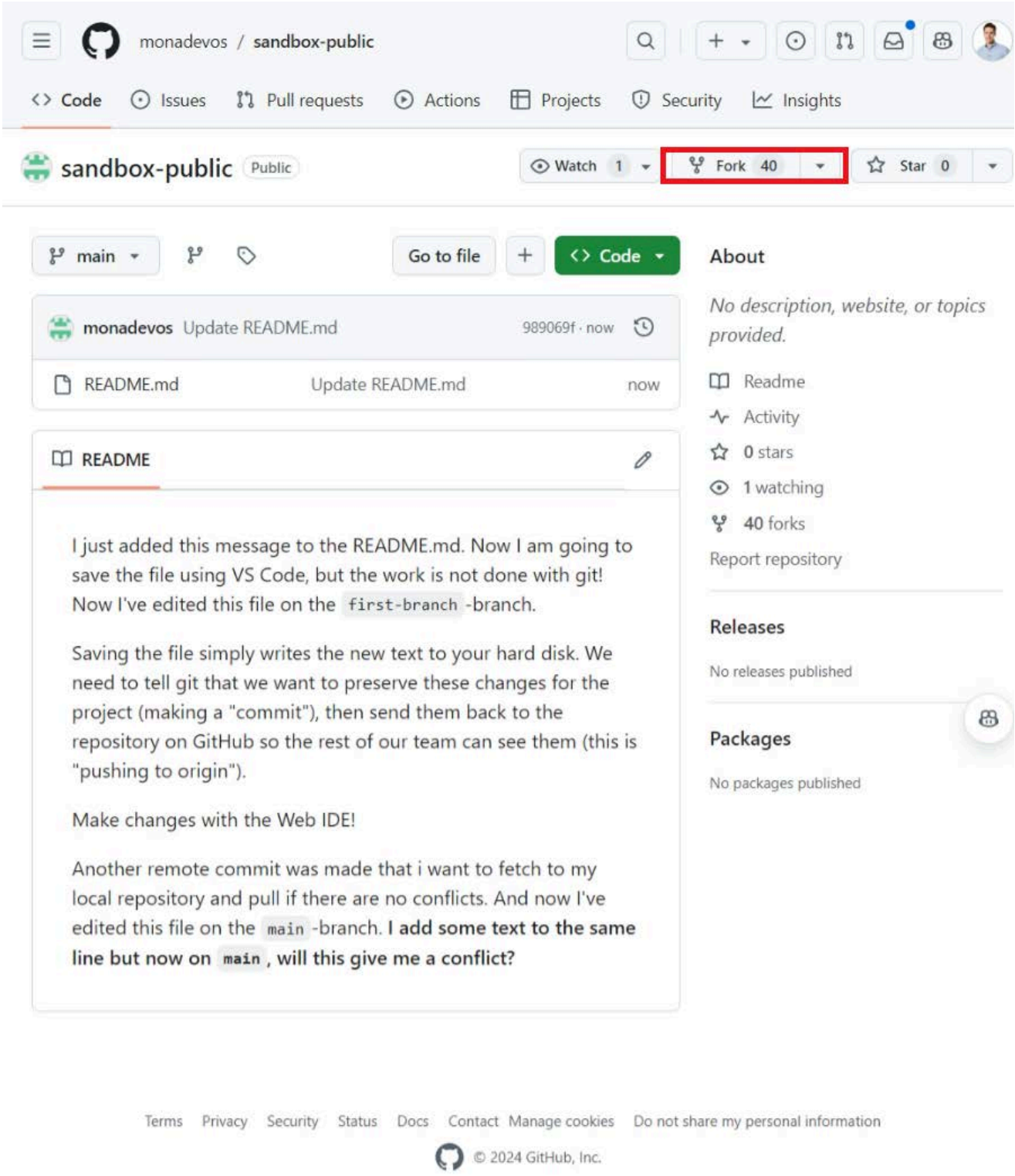

- Forking a Repository
- Contributing Back to the original repository


## Forking a Repository

Forking a repository on GitHub allows you to create a personal copy of someone else's project. This is particularly useful when you want to contribute to a project but do not have write access to the original repository.

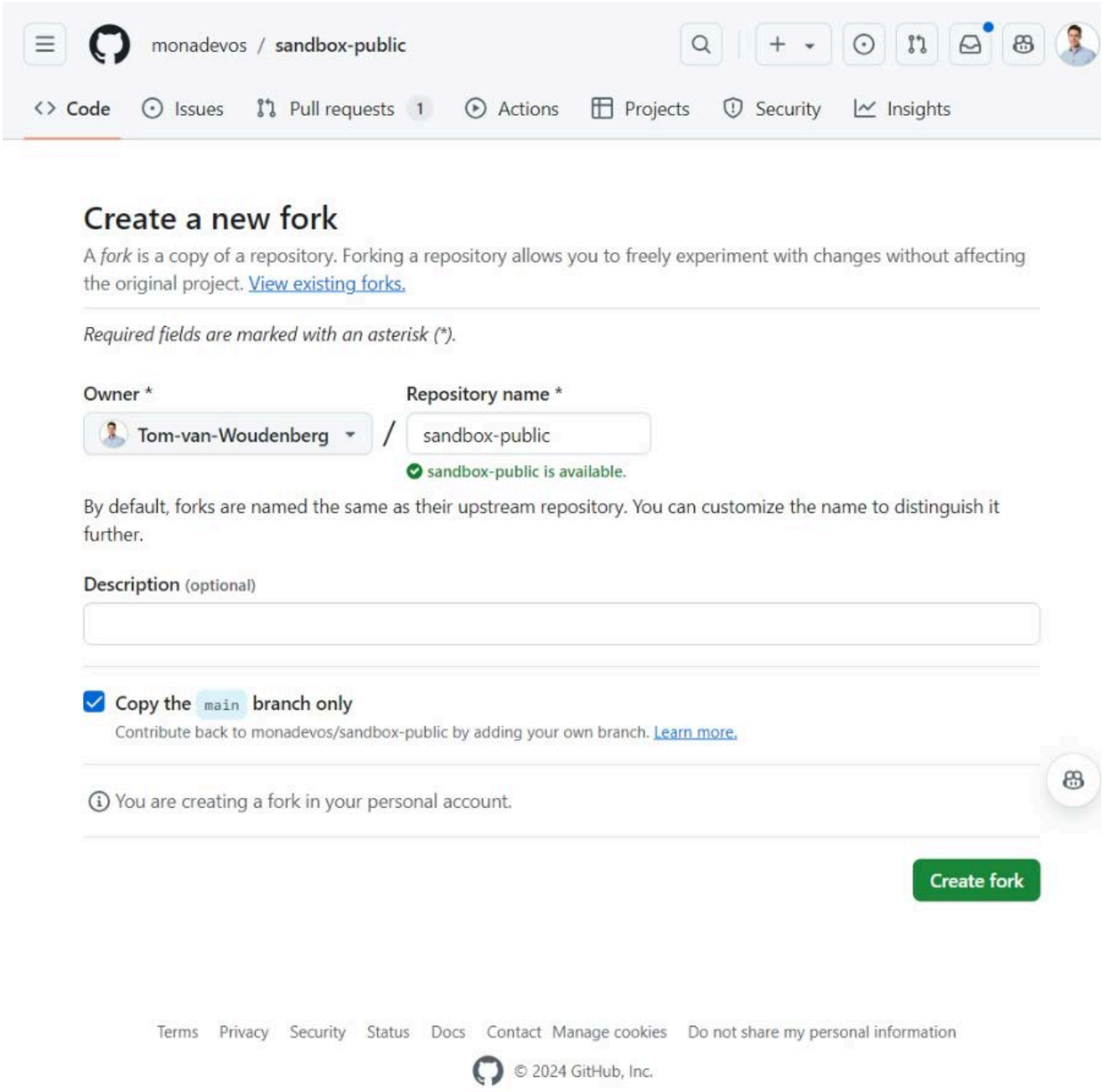
To fork a repository, follow these steps:

1. Navigate to the repository you want to fork on GitHub.
2. Click the "Fork" button at the top right of the repository page.




This will create a copy of the repository under your GitHub account.

3. Add the repo under your own account:



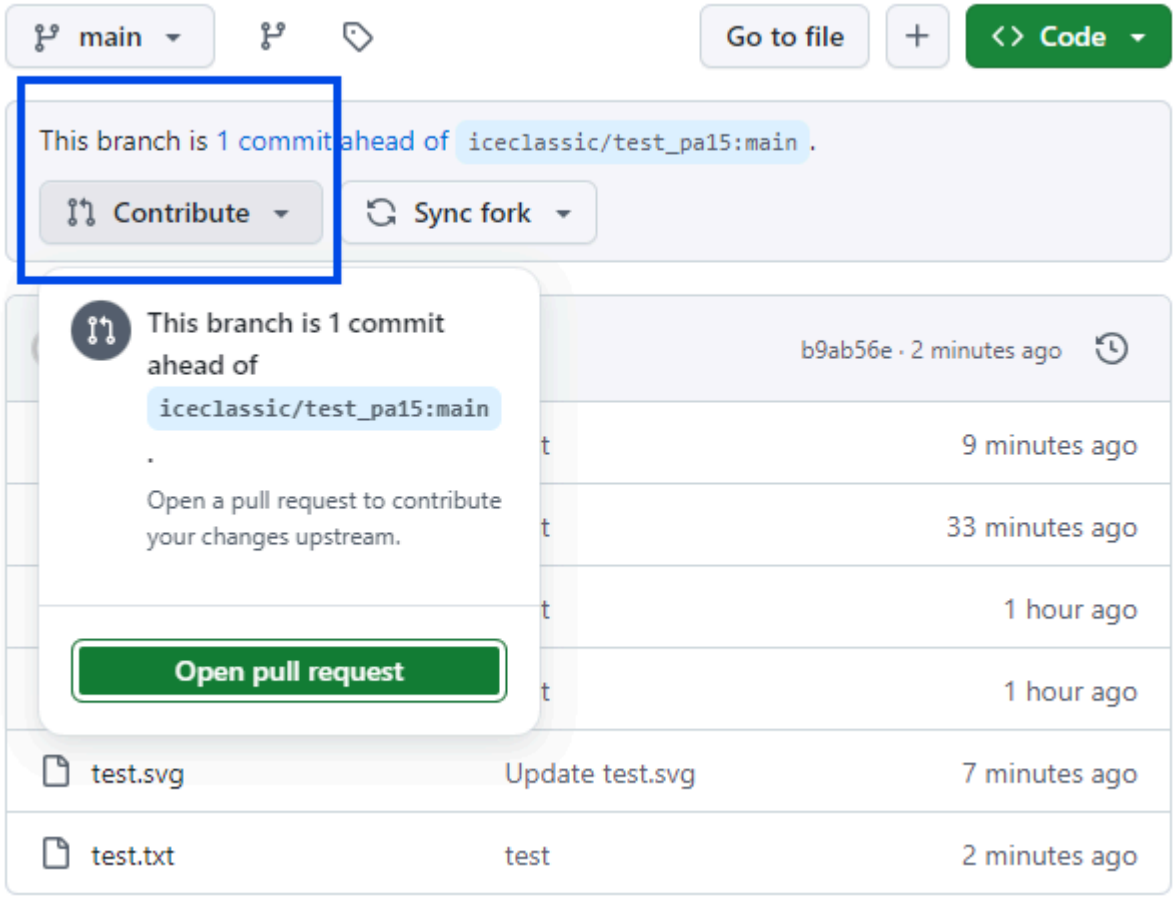

If you want, you can now clone this forked repository to your local machine and make changes as you would with any other repository.

# Contributing Back to the original repository

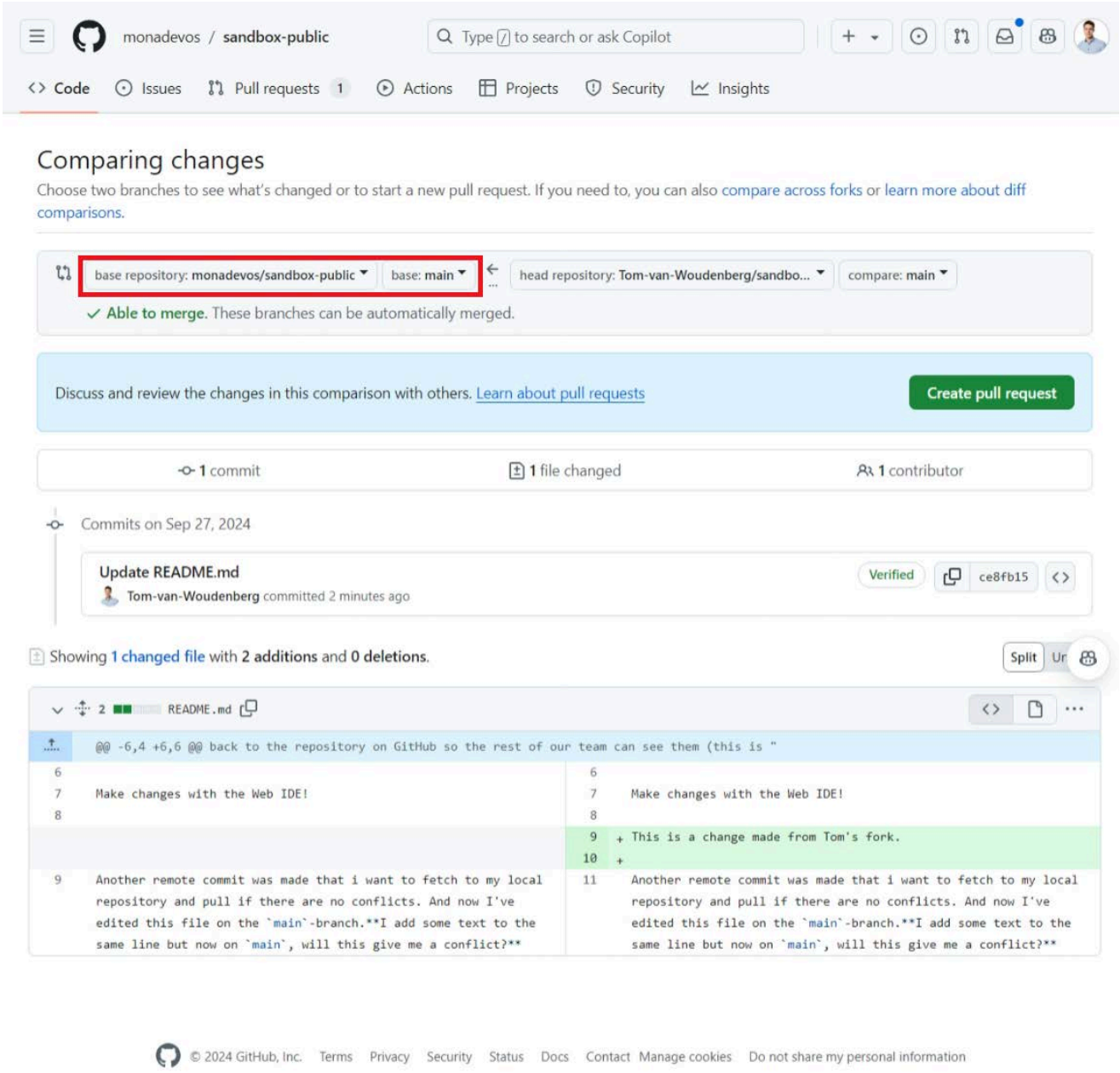
Once you have made changes to your forked repository, you can contribute back to the original repository by creating a pull request. Make sure to push any changes from your local repository to your Git repository! The process is similar to merging branches within a repository:

1. Make some changes to your forked repository

2. Select ‘Contribute’
3. Create a new pull request




3. Select the branch of the original repository to the original repository to which you want to contribute back




4. Select the branch from your forked repository that contains your changes.
5. Provide a title and description for your pull request, then click “Create pull request”.

The maintainers of the original repository will review your pull request. If they approve it, your changes will be merged into the original repository.

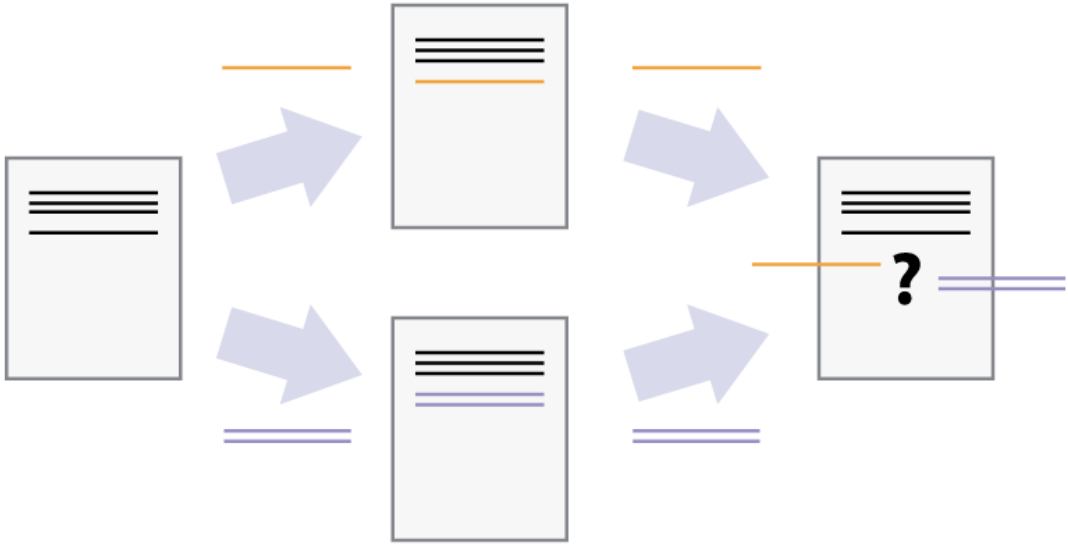
Forking and pull requests are powerful features of GitHub that enable collaborative development across different repositories, even when you do not have direct access to the original project.

# Merge conflicts

## Contents


- Why Merge into the conflicting branch first?

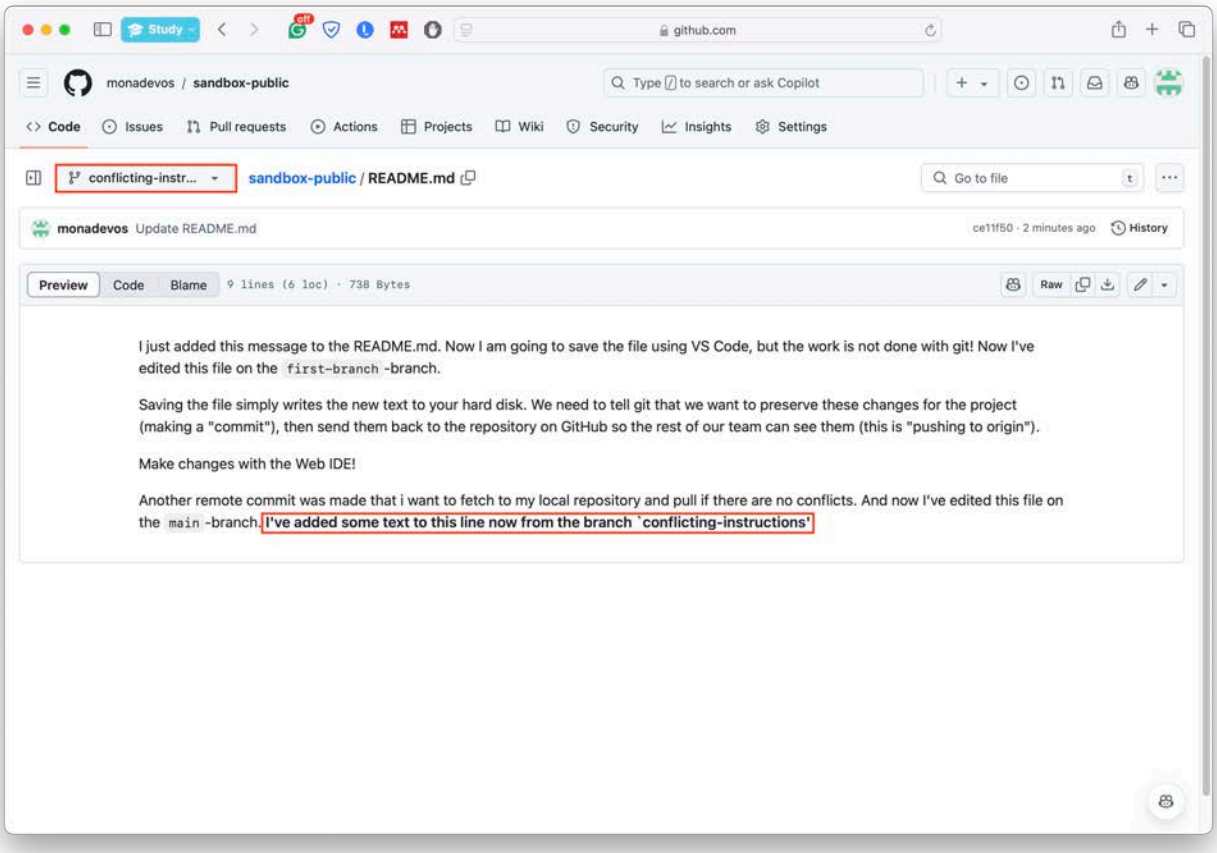

Merge conflicts arise when people on separate branches modify the same parts of one (or multiple) files. Since Git does not know how to handle that and whose changes to consider, it prompts the user to decide instead.




Let us create a merge conflict ourselves:

We will start by making a new branch `conflicting-instructions` on GitHub. We will edit the `README.md` file on the last line

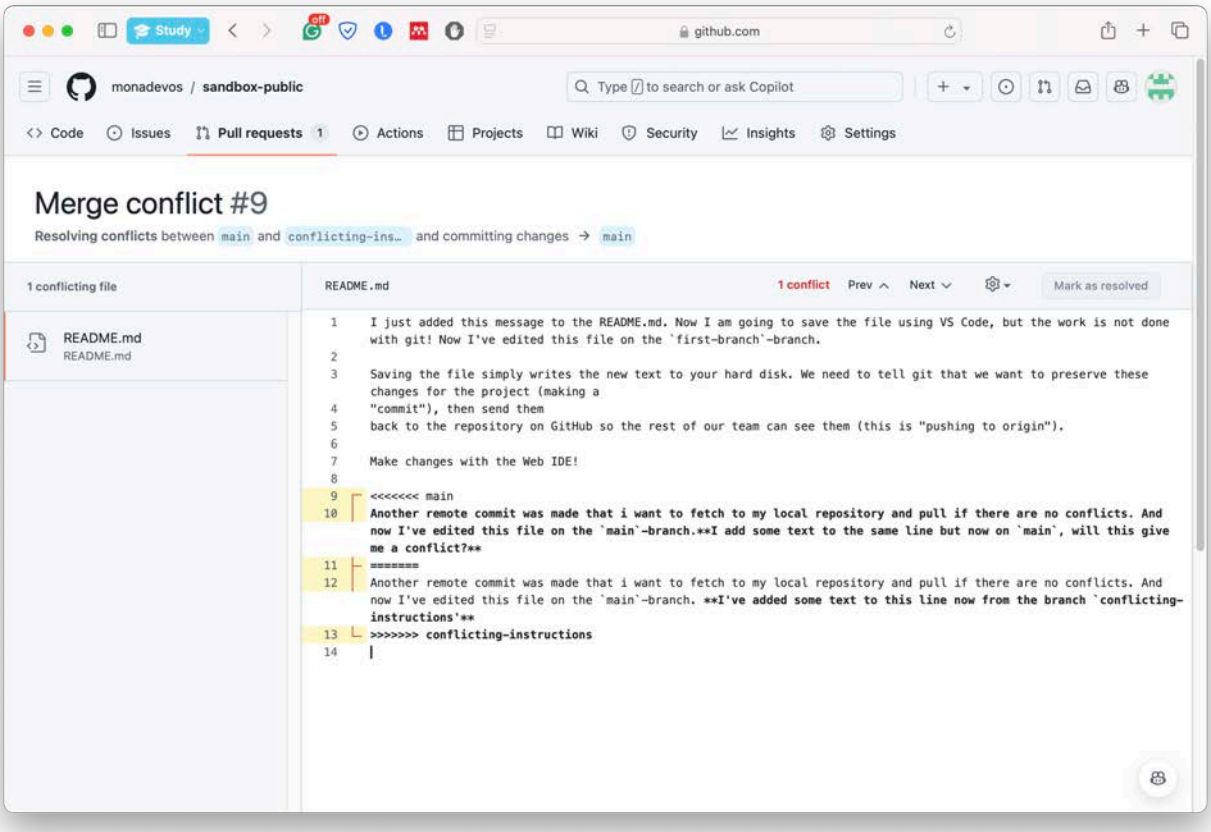
```
...

And now I've edited this file on the `main`-branch. **I've added some text to this line n
```



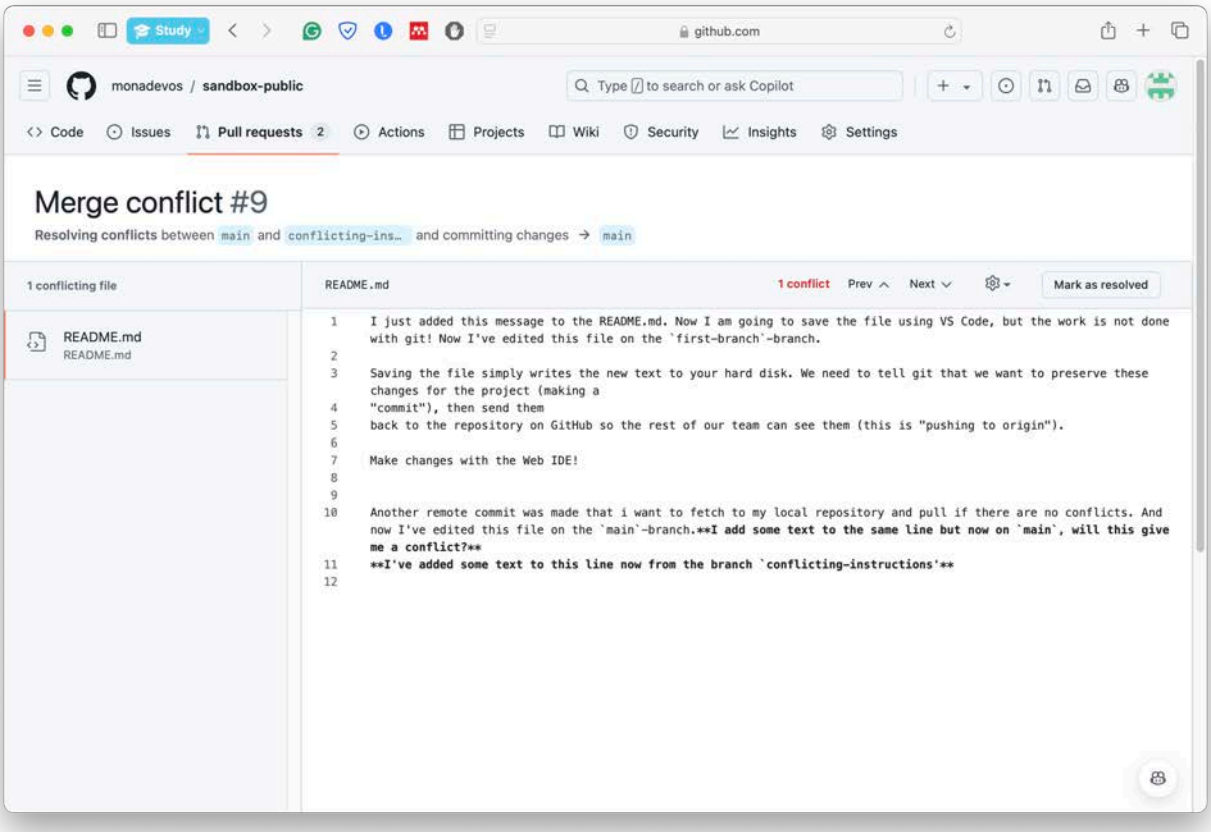

Next, we will return to the `main` branch on GitHub. We will edit the same line of the markdown file with some other text:

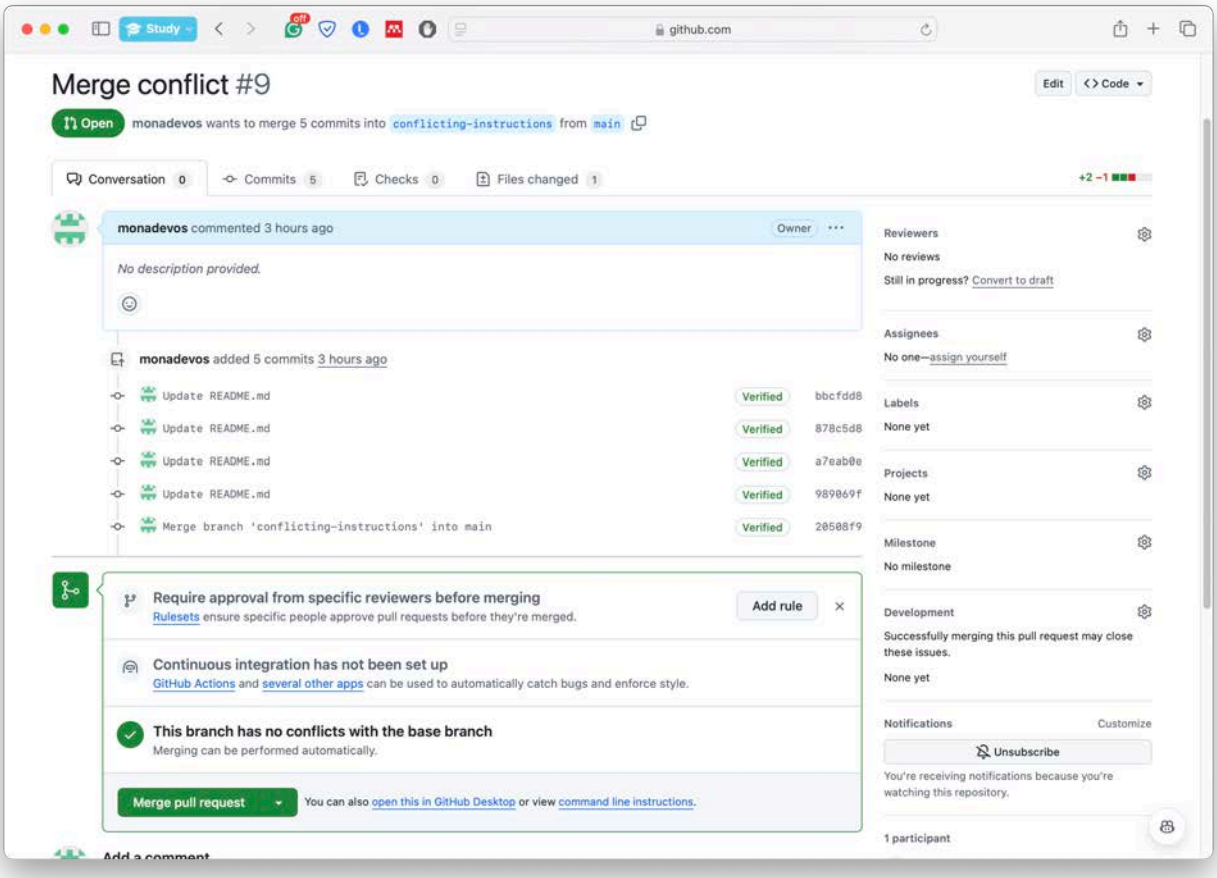
```
...

And now I've edited this file on the `main`-branch. **I add some text to the same line bu
```

Finally, we will attempt to merge the `main` branch into the `conflicting-instructions` branch using the pull request interface on GitHub. This will result in a merge conflict, as seen in the figure below, because we modified the same parts of the file.




The changes on the current branch are preceded by `<<<<<<< HEAD`, while the changes from the `main` branch are preceded by `=======` and followed by `>>>>>>> main`. To fix this conflict, we need to open the file in the GitHub editor and resolve the conflict by keeping the desired changes and removing the unnecessary parts. Once done, we can commit the resolved changes directly from the GitHub interface. Note that the current branch is now in a “MERGING” state. We have decided to take the best out of the two branches and merge their changes.


github.com
monadevos / sandbox-public
Type / to search or ask Copilot
Code
Issues
Pull requests 2
Actions
Projects
Wiki
Security
Insights
Settings
Merge conflict #9
Resolving conflicts between main and conflicting-ins… and committing changes → main
1 conflicting file
README.md
README.md
README.md
1 conflict
Prev
Next
Mark as resolved
1 I just added this message to the README.md. Now I am going to save the file using VS Code, but the work is not done with git! Now I've edited this file on the `first-branch`-branch.
2
3 Saving the file simply writes the new text to your hard disk. We need to tell git that we want to preserve these changes for the project (making a
4 "commit"), then send them
5 back to the repository on GitHub so the rest of our team can see them (this is "pushing to origin").
6
7 Make changes with the Web IDE!
8
9
10 Another remote commit was made that i want to fetch to my local repository and pull if there are no conflicts. And now I've edited this file on the `main`-branch.**I add some text to the same line but now on `main`, will this give me a conflict?**
11 **I've added some text to this line now from the branch `conflicting-instructions`**
12


github.com
Merge conflict #9
Edit
Code
Open
monadevos wants to merge 5 commits into conflicting-instructions from main
Conversation 0
Commits 5
Checks 0
Files changed 1
+2 −1
monadevos commented 3 hours ago
Owner
No description provided.
monadevos added 5 commits 3 hours ago
Update README.md Verified bbcfdd8
Update README.md Verified 878c5d8
Update README.md Verified a7eab0e
Update README.md Verified 989069f
Merge branch 'conflicting-instructions' into main Verified 20508f9
Require approval from specific reviewers before merging
Rulesets ensure specific people approve pull requests before they're merged.
Add rule
Continuous integration has not been set up
GitHub Actions and several other apps can be used to automatically catch bugs and enforce style.
This branch has no conflicts with the base branch
Merging can be performed automatically.
Merge pull request
You can also open this in GitHub Desktop or view command line instructions.
Add a comment
Reviewers
No reviews
Still in progress? Convert to draft
Assignees
No one—assign yourself
Labels
None yet
Projects
None yet
Milestone
No milestone
Development
Successfully merging this pull request may close these issues.
None yet
Notifications
Customize
Unsubscribe
You're receiving notifications because you're watching this repository.
1 participant

# Why Merge into the conflicting branch first?

Merging into the conflicting branch first allows you to resolve conflicts in a controlled environment. Here are some reasons why this approach is beneficial:

- By resolving conflicts in the feature or conflicting branch, you isolate the changes and ensure that the main branch remains stable and conflict-free.
- In case of coding, you can thoroughly test the merged changes in the conflicting branch before integrating them into the main branch, reducing the risk of introducing bugs.
- Keeping the main branch clean and conflict-free simplifies the integration process and makes it easier for other team members to continue their work without interruptions.

# Week 1.6: Errors

> **Attribution**
>
> This chapter reuses material from *Python for Engineers*. Find out more here.

At this point you are used to the fact that bugs in your code are a fact of life, but probably don't have a methodical way of dealing with them. Perhaps sometimes you get errors and and have no idea how to fix them. This week we will consider various *error types* and promote a structured approach to debugging by learning about the *traceback* feature in Python.

This week we introduce several tools that while simple, form foundational elements of the broader field of *software testing.* If you ever use or write more complex (i.e., a collection of Python functions, modules or a package), these tools are essential.MUDE or elsewhere!

# Errors

This chapter focuses on a number of different topics to help you identify and fix *errors,* which are an essential component of *software testing.* Incorporating the concepts in this chapter into your own coding practice will help you fix problems in the future, especially as you work on larger projects with multiple people.

**Why should we learn about software testing?**

Testing our code is necessary, because we as humans all make mistakes. Some of our mistakes may not be that important, but some can be dangeours and harmful. There are numerous examples of the latter such as the explosion of the Ariane 5 rocket or the problem with NASA's Mars Climate Orbiter. You can read more on these examples here. Testing our code for hidden flaws is the first step towards ensuring that the programs we write are robust.

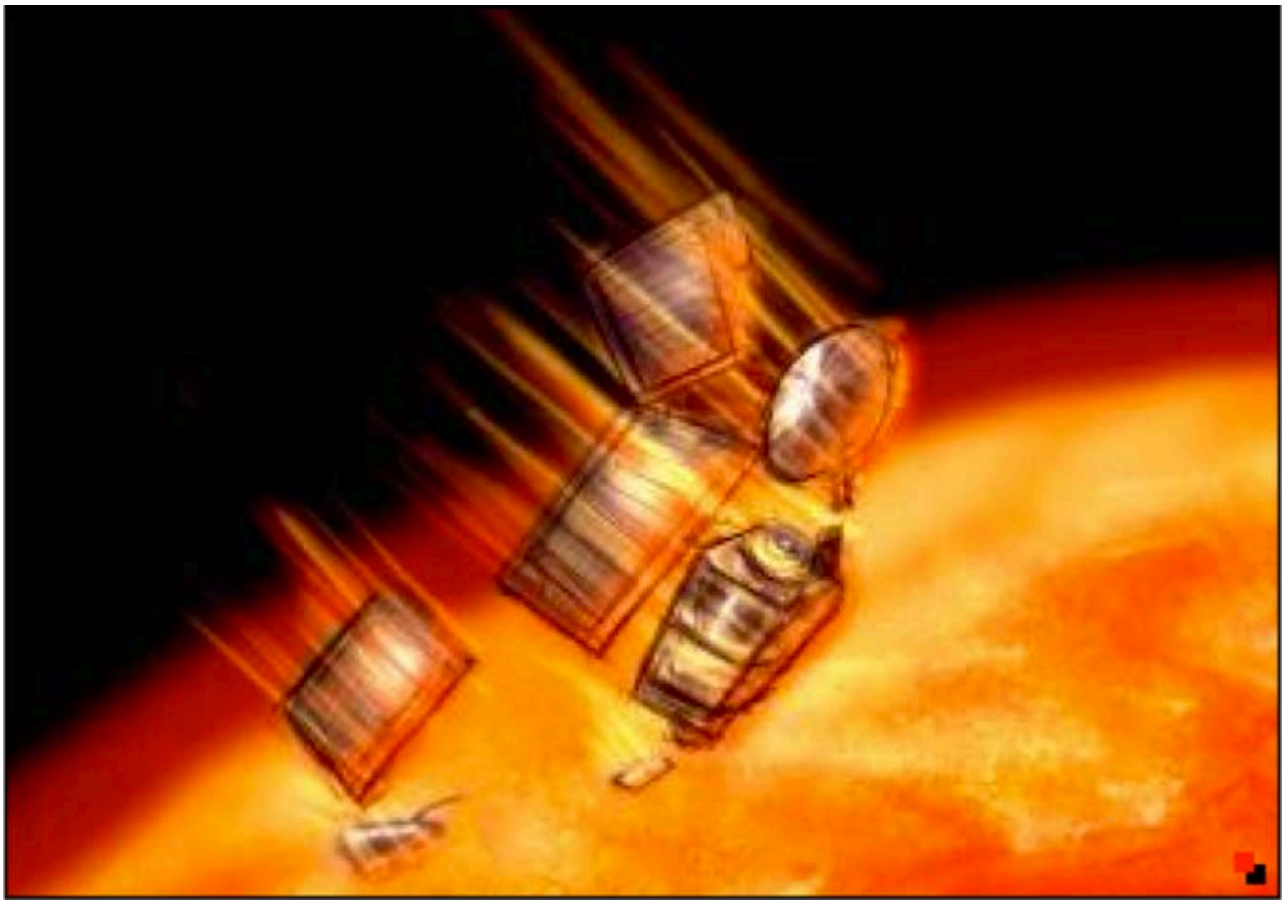


*Fig. 45* A depiction of the NASA's Mars Climate Orbiter disaster. The incongruence in the units used by scientists at NASA (metric unit) and Lockheed Martin (US customary units) led the space probe too close to the planet. It was either destroyed in the atmosphere or escaped the planet's vicinity and entered an orbit around the Sun. (Source: Unleesh.com)

**What is debugging?**

Debugging is the process of identifying code-related problems (bugs) by analysing the behaviour of the program we have written. Testing and debugging are complementary to one another. Testing is used to identify a bug, while debugging is used to fix a bug.

In the sections that follow, we will start to learn about errors as a practical way of identifying issues with our code. Then we will learn how to improve our code with testing to make it easier to debug in the future.

# Error Types

## Contents


- Syntax errors
- Exceptions
- Logical errors
- Summary of Error Types


You have probably encountered errors before in your code—no one is perfect! But have you ever spent time reading through the error message to help you figure out what went wrong? This Section will give you some information that will help interpret these sometimes cryptic message to more efficiently identify the problem and fix it.

Essentially there are 3 types of errors, each of which is described below:

- Syntax errors
- Exceptions
- Logical errors

Click 🚀 –> **Live Code** on the top right corner of this screen and then execute the cells as you go through the page. In each case, see if you can fix the code (re-run the cell until the error is gone), as well as try to replicate the error with your own example.

In each of the examples, you should look at the error report that is printed after executing each cell and identify the *type* of error; in each case they should look like `XxxxError`, where `Xxxx` identifies the specific error type.


*The work on this page is derived from work that is Copyright © University of Cape Town under CC BY-SA 4.0. Specifically, material from this page was used.*


# Syntax errors

Syntax errors are the most common for beginner developers. When an error happenes, Python parser will show in which line it is and point to it in the code with an arrow `^`. Run the code snippet below for an example:

```
if 5 > 3:
    print("5 is bigger than 3")
else
    print("3 is bigger than 5")
```

```
  Cell In[2], line 3
    else
        ^
SyntaxError: invalid syntax
```

As we can see from the error report, Python tells us directly that this is a `SyntaxError`. It even includes a carat (the `^` symbol) to identify the location within the line of code where it occurs. Useful, right?!

# Exceptions

Exceptions is a general set of errors that, unlike syntax errors, appear during code execution. If an exception stops your code from running, we often refer to this as *raising an exception.* Many exceptions are implemented directly in the Python code base, but it is important to recognize that anyone writing Python code can create exceptions themselves. Determining how and when to *raise* exceptions (i.e., "cause" the error) during code execution is a useful way to make sure code and software runs as designed; we will learn to do this later. For now we will look at several examples of exceptions that are implemented in the Python code base:

```
16 / 0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Cell In[2], line 1
----> 1 16 / 0

ZeroDivisionError: division by zero
```

```
2 + "3"
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[3], line 1
----> 1 2 + "3"

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
2 * pi * r ** 2
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[4], line 1
----> 1 2 * pi * r ** 2

NameError: name 'pi' is not defined
```

As you see in the examples above, we have encountered 3 different types of exceptions - `ZeroDivisionError`, `TypeError` and `NameError`. A nice thing about Python is that it tells us the specific type of exception we are dealing it, making it easy for you, the developer, to figure out the cause of the raised exceptions. Although in some cases the cause of the exception is obvious, it is often useful to look up the Python documentation that describes it (especially when your code gets complex, and you are having trouble identifying the source of the problem). For example, if you look at the description of `NameError`, you will find:

> exception **NameError**
> Raised when a local or global name is not found. This applies only to unqualified names. The associated value is an error message that includes the name that could not be found.
>
> The name attribute can be set using a keyword-only argument to the constructor. When set it represent the name of the variable that was attempted to be accessed.
>
> *Changed in version 3.10:* Added the `name` attribute.

As you can see, the documentation also provides information about changes relative to older versions of Python.

For more exception types, you can have a look at the complete list here. Keep in mind that you are not required to understand the cause of every exception, but rather the importance of having different types of exceptions and that each one may require a different strategy in resolving it.

# Logical errors

Logical errors are generally the most difficult to discover, because they can be invisible to the developer. Logical errors can exist without raising an exception or an error. Particular examples of logical errors are:

- using the wrong variable name
- indenting a block to the wrong level
- using integer division instead of floating-point division
- getting operator precedence wrong
- making a mistake in a boolean expression
- forgetting to add an `else` statement to an `if` clause
- off-by-one, and other numerical errors

Here is a brief example of using integer division instead of floating-point division:

Consider you have to compute speed from distance and time using the basic formula $speed = \frac{distance}{time}$. Here is the correct version, using floating-point division:

```
distance = 10            # m
time = 3.63              # s

speed = distance / time    # m/s
print('Speed = ', speed)
```

```
Speed =  2.7548209366391188
```

If we use integer division by mistake, the result is incorrect:

```
distance = 10            # m
time = 3.63              # s

speed = distance // time    # m/s
print('Speed = ', speed)
```

```
Speed =  2.0
```

The difference between the 2 results is nearly 0.75, which can be very high depending on its use.

# Summary of Error Types

On this page we considered three types of errors. One way to help distinguish them is to recognize when they may occur:

- Syntax error: before the code is executed
- Exceptions: during execution

- Logic error: code executes without error

When executing the code in the examples above, did you read the error report? It is called a **traceback** and it should have been useful to identify the error type. We will learn more about it in the next section.

# The Python Traceback

## Contents


- Python Traceback
- Exercise
- Complicated Tracebacks


How often has your Python code “broken,” spitting out a long error message that you then completely ignore while going back to your code to make changes in a vain attempt to solve the problem as quickly as possible? Or perhaps you executed a piece of code and, upon seeing the error, turn to a friend or teacher, providing such “useful” information as: “help, my code doesn’t work!”

The error message you recieve from Python is providing useful information to identify the problem source, and after reading this page you should be able to utilize this to solve them more efficiently. Even if you can’t use the information to solve your problem directly, it is important to be able to communicate effectively about the problem when asking for help. For example, there is a big difference between asking someone for help by saying “my code is broken,” versus “I have an index error and I can’t figure out why it’s happening.”

Click 🚀 –> **Live Code** on the top right corner of this screen and then execute the cells as you go through the page. In each case, see if you can fix the code (re-run the cell until the error is gone), as well as try to replicate the error with your own example.


*The work on this page is derived from work that is Copyright © The Carpentries under CC BY 4.0. Specifically, material from this page was used.*


# Python Traceback

The **traceback** is simply a listing and description of the sequence of function calls leading to an error; specifically, it will help identify exceptions that have been raised during code execution. Let’s take a look at one!

```
def favorite_ice_cream():
    ice_creams = [
        'chocolate',
        'vanilla',
        'strawberry'
    ]
    print(ice_creams[3])

favorite_ice_cream()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[1], line 9
      2     ice_creams = [
      3         'chocolate',
      4         'vanilla',
      5         'strawberry'
      6     ]
      7     print(ice_creams[3])
----> 9 favorite_ice_cream()

Cell In[1], line 7, in favorite_ice_cream()
      1 def favorite_ice_cream():
      2     ice_creams = [
      3         'chocolate',
      4         'vanilla',
      5         'strawberry'
      6     ]
----> 7     print(ice_creams[3])

IndexError: list index out of range
```

This particular traceback has two levels. You can determine the number of levels by looking for the number of arrows on the left hand side. In this case:

1. The first shows code from the cell above, with an arrow pointing to Line 9 (which is `favorite_ice_cream()`).
2. The second shows some code in the function `favorite_ice_cream`, with an arrow pointing to Line 7 (which is `print(ice_creams[3])`).

The last level is the actual place where the error occurred. The other level(s) show what function the program executed to get to the next level down. So, in this case, the program first performed a function call to the function `favorite_ice_cream`. Inside this function, the program encountered an error on Line 6, when it tried to run the code `print(ice_creams[3])`.

**Note**

Sometimes, you might see a traceback that is very long – sometimes they might even be 20 levels deep! This can make it seem like something horrible happened, but the length of the error message does not reflect severity, rather, it indicates that your program called many functions before it encountered the error. Most of the time, the actual place where the error occurred is at the bottom-most level, so you can skip down the traceback to the bottom.

So what error did the program actually encounter? In the last line of the traceback, Python helpfully tells us the category or type of error (in this case, it is an `IndexError`) and a more detailed error message (in this case, it says “list index out of range”).

If you encounter an error and don’t know what it means, it is still important to read the traceback closely. That way, if you fix the error, but encounter a new one, you can tell that the error changed. Additionally, sometimes knowing where the error occurred is enough to fix it, even if you don’t entirely understand the message.

If you do encounter an error you don’t recognize, try looking at the official documentation on errors. However, note that you may not always be able to find the error there, as it is possible to create custom errors. In that case, hopefully the custom error message is informative enough to help you figure out what went wrong.

In general, we recommend you think through the following questions whenever reading a traceback:

**Reading the Traceback (Error Messages)**

Use the traceback to answer the following questions:

1. How many levels does the traceback have?
2. What is the function name where the error occurred?
3. On which line number in this function did the error occur?
4. What is the type of error?
5. What is the error message?

# Exercise

After reading and executing the Python code and the resulting traceback, answer the following questions

1. How many levels does the traceback have?
2. What is the function name where the error occurred?
3. On which line number in this function did the error occur?
4. What is the type of error?
5. What is the error message?

```
def print_message(day):
    messages = [
        'Hello, world!',
        'Today is Tuesday!',
        'It is the middle of the week.',
        'Today is Donnerstag in German!',
        'Last day of the week!',
        'Hooray for the weekend!',
        'Aw, the weekend is almost over.'
    ]
    print(messages[day])

def print_sunday_message():
    print_message(7)

print_sunday_message()
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[2], line 16
     13 def print_sunday_message():
     14     print_message(7)
---> 16 print_sunday_message()

Cell In[2], line 14, in print_sunday_message()
     13 def print_sunday_message():
---> 14     print_message(7)

Cell In[2], line 11, in print_message(day)
      1 def print_message(day):
      2     messages = [
      3         'Hello, world!',
      4         'Today is Tuesday!',
   (...)
      9         'Aw, the weekend is almost over.'
     10     ]
---> 11     print(messages[day])

IndexError: list index out of range
```

**Solution**

1. 3 levels
2. `print_message`
3. 11
4. `IndexError`
5. `list index out of range` You can then infer that `7` is not the right index to use with `messages`.

# Complicated Tracebacks

The examples above generate relatively simple tracebacks; however, when you are working on your own computer the “real” tracebacks may seem more complicated. Often this is simply because of the way Python files are stored in your computer. Although this online textbook is not the same as the Python distribution you have on your computer, we can get a sense of the problem by executing the cell below:

```
import numpy as np
x = np.array([5, 6, 7, 8])
np.this_method_does_not_exist_in_numpy
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[3], line 3
      1 import numpy as np
      2 x = np.array([5, 6, 7, 8])
----> 3 np.this_method_does_not_exist_in_numpy

File c:\Users\tomvanwoudenbe\Anaconda3\lib\site-packages\numpy\__init__.py:333, in __geta
    330     "Removed in NumPy 1.25.0"
    331     raise RuntimeError("Tester was removed in NumPy 1.25.")
--> 333 raise AttributeError("module {!r} has no attribute "
    334                      "{!r}".format(__name__, attr))

AttributeError: module 'numpy' has no attribute 'this_method_does_not_exist_in_numpy'
```

**Can you answer the list of 5 questions for this traceback?**

**Solution**

1. 2 levels
2. Numpy; specifically: `np.this_method_does_not_exist_in_numpy`
3. 3
4. `AttributeError`
5. Module ‘numpy’ has no attribute ‘this_method_does_not_exist_in_numpy’. It’s pretty clear that this method doesn’t exist in Numpy!

In addition, the location of the file where the exception was raised is given; it is quite long and (perhaps) unclear. This is simply the location where the Python file (`numpy/__init__.py`) responsible for code execution are stored in your internet browser (`/lib/python/...`). Note that `__init__.py` is used in every Python Package. Tou can see references to the method within numpy responsible for finding the numpy methods or attributes: `__getattr__(attr)`. Finally, we can actually see how the exception is *raised* (we will learn about this later!): `raise AttributeError...`.

Hopefully this explanation makes it easier for you to identify and solve exceptions when working with Python!

# Raising errors

## Contents


- Exercise


When writing good code, it is beneficial to raise exceptions yourself if you are provided with incorrect input.

Consider the example method `calculate_weight` below, which calculates the weight of an object, given that we know its density and volume:

```
def calculate_weight(density, volume):
    """
    Calculates weight of an object, given its density and volume.

    Args:
        density (int): Density of an object.
        volume (int): Volume of an object.

    Returns:
        int: Weight of an object.
    """
    return density * volume
```

```
calculate_weight(3, 10)
```

```
30
```

```
calculate_weight(-4, 5)
```

```
-20
```

If we provide invalid input such as negative density or volume, the method will still give us an answer, because it is following a formula. However, using that incorrect calculation can be harmful in applications. Therefore, we can raise exceptions ourselves using the keyword `raise`:

```python
def calculate_weight(density, volume):
    """
    Calculates weight of an object, given its density and volume.

    Args:
        density (int): Density of an object.
        volume (int): Volume of an object.

    Returns:
        int: Weight of an object.

    Raises:
        ValueError: if density or volume are negative values.
    """
    if density < 0:
        raise ValueError(
            "Invalid density. Density parameter can only take positive numbers"
        )
    if volume < 0:
        raise ValueError(
            "Invalid volume. Volume parameter can only take positive numbers"
        )
    return density * volume
```

```python
calculate_weight(75, 5)
```

```
375
```

```python
calculate_weight(-4, 50)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[13], line 1
----> 1 calculate_weight(-4, 50)

Cell In[11], line 16, in calculate_weight(density, volume)
      2 """
      3 Calculates weight of an object, given its density and volume.
      4
   (...)
     13     ValueError: if density or volume are negative values.
     14 """
     15 if density < 0:
---> 16     raise ValueError(
     17         "Invalid density. Density parameter can only take positive numbers"
     18     )
     19 if volume < 0:
     20     raise ValueError(
     21         "Invalid volume. Volume parameter can only take positive numbers"
     22     )

ValueError: Invalid density. Density parameter can only take positive numbers
```

# Exercise

You are given the method `calculate_area` again to calculate the *shuttering area* of a concrete rectangular column. The shuttering is a temporary arrangement done for vertical surfaces to support the wet concrete till it attains the desired strength (source: Civilread.com).

If incorrect output is provided to it, the method will return `-1`. Instead of that, we would like to raise a `ValueError`. Modify the method to raise this exception on incorrect output. Furthermore, update the documentation of the method to account for this change.

```
def calculate_area(height, length, breadth):
    """
    Calculates shuttering area of a concrete rectangular column.

    Args:
        height (int): Height of the column.
        length (int): Length of the column.
        breadth (int): Breadth of the column.

    Returns:
        int: The area or -1 if the input is invalid.
    """
    if height <= 0 or length <= 0 or breadth <= 0:
        return -1
    else:
        return (2 * breadth + 2 * height) * length
```

**Solution**

```
def calculate_area(height, length, breadth):
    """
    Calculates shuttering area of a concrete rectangular column.

    Args:
        height (int): Height of the column.
        length (int): Length of the column.
        breadth (int): Breadth of the column.

    Returns:
        int: The area.
    Raises:
        ValueError: if height, length or breadth are non-positive values.
    """
    if height <= 0 or length <= 0 or breadth <= 0:
        raise ValueError("Invalid height or length or breadth provided.")
    else:
        return (2 * breadth + 2 * height) * length
```

# Handling errors: the try - except block

## Contents

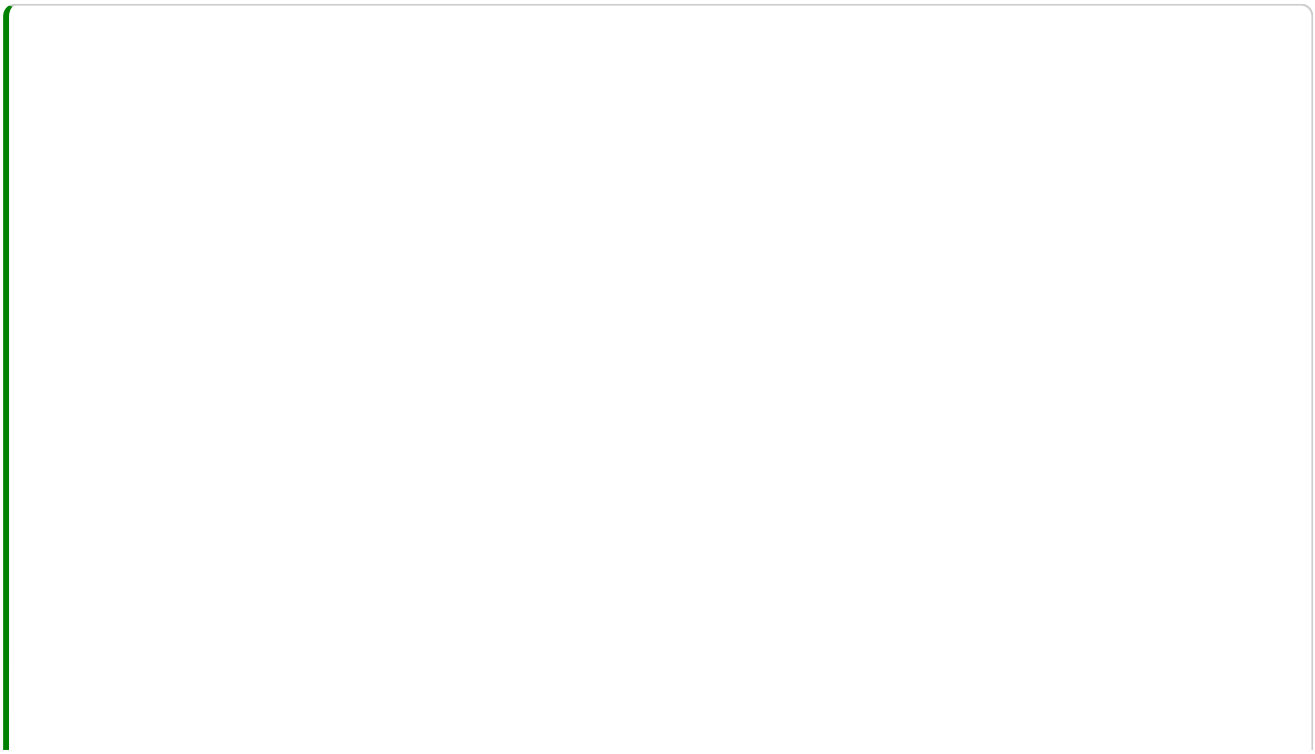

- `else` and `finally`


If exceptions are not handled properly, the application will crash during execution as we have seen in the examples above. As a result, handling exceptions is important, because we want to minimise application crashes. Sometimes this is called “catching” errors.

It is important to also think about *when* we should catch errors: this is when you expect the error to be *recoverable*. In other words, dealing with isthe error should not affect future code that will run. Some errors are *fatal* and should be left to crash the program since things won’t work out anyways.

To handle exceptions, we can make use of `try - except` block, where we can catch exceptions and handle them in the way we wish. Thus, if an exception is *raised*, the application is not going to crash.

Let’s see how that works by reusing the function from the previous section:



```
def calculate_weight(density, volume):
    """
    Calculates weight of an object, given its density and volume.

    Args:
        density (int): Density of an object.
        volume (int): Volume of an object.

    Returns:
        int: Weight of an object.

    Raises:
        ValueError: if density or volume are negative values.
    """
    if density < 0:
        raise ValueError(
            "Invalid density. Density parameter can only take positive numbers"
        )
    if volume < 0:
        raise ValueError(
            "Invalid volume. Volume parameter can only take positive numbers"
        )
    return density * volume
```

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except ValueError:
    print("An exception occurred!")
```

```
An exception occurred!
```

Note that if we do not handle the correct exception, one will still be raised. Therefore, it is very important that we catch the correct one based on our knowledge of which error can occur.

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except NameError:
    print("An exception occurred!")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[15], line 6
      3 volume = 5
      5 try:
----> 6     weight = calculate_weight(density, volume)
      7 except NameError:
      8     print("An exception occurred!")

Cell In[11], line 16, in calculate_weight(density, volume)
      2 """
      3 Calculates weight of an object, given its density and volume.
      4
   (...)
     13     ValueError: if density or volume are negative values.
     14 """
     15 if density < 0:
---> 16     raise ValueError(
     17         "Invalid density. Density parameter can only take positive numbers"
     18     )
     19 if volume < 0:
     20     raise ValueError(
     21         "Invalid volume. Volume parameter can only take positive numbers"
     22     )

ValueError: Invalid density. Density parameter can only take positive numbers
```

If we do not know what exception will be raised, we can use `except Exception`, which will handle any exception raised by the code inside the `try` clause block. (`Exception` is the super class that comprises all types of exceptions and errors; for the purpose of this activity, think of it as a generic error.):

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except Exception:
    print("An exception occurred!")
```

```
An exception occurred!
```

## `else` and `finally`

There are 2 more clauses that we can add to our `try-except` block, namely `else` and `finally`:

- `else` will be executed only if no exception is raised by the code in the `try` clause block. This is useful in case we want some code to be executed after successful

completion of the `try` clause. For example, printing a message or sending a response to somebody.

- `finally` will be executed all the time, regardless of whether an exception was raised or not. In practice this is mostly used to finish a process. For example, if you are reading from a file, it is customary to close the file when you finish reading from it.

Study the example below, which shows the execution of `except` and `finally` clauses:

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except Exception:
    print("An exception occurred!")
else:
    print(
        f"The weight of an object with density of {density} kg/m^3 and "
        f"volume of {volume} m^3 is {weight} kg."
    )
finally:
    print("Calculation finished!")
```

```
An exception occurred!
Calculation finished!
```

We will now use valid input to our function `calculate_weight` to show the execution of `try` and `else` clauses:

```
weight = 0
density = 7870
volume = 40

try:
    weight = calculate_weight(density, volume)
except Exception:
    print("An exception occurred!")
else:
    print(
        f"The weight of an object with density of {density} kg/m^3 and "
        f"volume of {volume} m^3 is {weight} kg."
    )
finally:
    print("Calculation finished!")
```

```
The weight of an object with density of 7870 kg/m^3 and volume of 40 m^3 is 314800 kg.
Calculation finished!
```

It is also possible to interact with any exception object. Have a look at the example below, where `e` is the object of the raised exception:

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except ValueError as e:
    print(f"An exception occurred: {e}")
```

```
An exception occurred: Invalid density. Density parameter can only take positive numbers
```

It is possible to have multiple `except` statements in case you are expecting more than 1 type of exceptions to be raised. If an exception is raised, the first `except` clause that catches it will be evaluated.

You may be wondering why should we handle multiple exceptions. Depending on the exception raised, we can set up different behaviour for our application. For example, if a `ZeroDivisionError`, the issue is most likely in the user providing arguments, so we could inform them about it specifically. However, if a `ValueError` is thrown, it may be related to our logic. In that case we should be able to investigate the cause of it.

```
weight = 0
density = -40
volume = 5

try:
    weight = calculate_weight(density, volume)
except ZeroDivisionError:
    print("Division by zero exception was raised!")
except ValueError:
    print("Value error exception was raised!")
except TypeError:
    print("Type error exception was raised!")
```

```
Value error exception was raised!
```

# Assertions

## Contents


- Exercise
- Exercise
- Asserts and floating point numbers


The main purpose of *assertions* is to catch logical errors: we need checks that verify that our code is working correctly.

Assertions can be defined by the keyword `assert`. They take an expression that evaluates to a boolean and if its value is `False`, then an `AssertionError` is raised. We can set a message for the raised exception by providing it in the assertion.

You can think of assertion statements `assert <condition>, <message>` as equivalent to:

```
if not condition:
  raise AssertionError(message)
```

Assertions are very useful for *unit testing* your code. A *unit* can be seen as the smallest piece of code that can be individually tested. In our case, this would mean testing a Python function.

It is advisable to make one or multiple test functions that test your code and run those functions every time you make a modification to the *codebase* (i.e., all the code you develop). This can ensure that any new modifications to the existing code do not damage the previous functionalities.

Below you can find a few examples where we have correct assertions, using the function from above and the `calculate_weight` function we've seen before. Therefore, no exception is occurring. Note that setting a message is not mandatory, but highly recommended for debugging purposes:

```
def calculate_weight(density, volume):
    """
    Calculates weight of an object, given its density and volume.

    Args:
        density (int): Density of an object.
        volume (int): Volume of an object.

    Returns:
        int: Weight of an object.
    """
    return density * volume
```

```
assert calculate_weight(30, 10) == 300
assert calculate_weight(70, 4) == 280, "The weight of an object with density of 70 kg/m^3
assert calculate_weight(12, 500) == 6000, "The weight of an object with density of 12 kg/
```

Below, there is an example of a failing assertion for a function converting Kelvin to Celsius. This assertion should trigger the developer into adjusting the value of the zero_point.

```
def kelvin_to_celsius(kelvin_temperature):
    """
    Conerts [K] into [°C]

    Args:
        kelvin_temperature (float): Temperature in [K]

    Returns:
        float: Temperature in [°C].
    """
    zero_point = 273
    return kelvin_temperature-zero_point

assert kelvin_to_celsius(0)==-273.15, "0 Kelvin are equal to -273.15 Celsius"
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[3], line 14
     11     zero_point = 273
     12     return kelvin_temperature-zero_point
---> 14 assert kelvin_to_celsius(0)==-273.15, "0 Kelvin are equal to -273.15 Celsius"

AssertionError: 0 Kelvin are equal to -273.15 Celsius
```

When writing tests we should focus on covering “boundary” values. Those are values that are part of a condition statement. For example, in the statement `if weight > 0`, the boundary value is 0.

By a rule of thumb, it is customary, to have at least 4 tests for every condition:

- one that is on the boundary of condition and evaluates it to `True` (i.e. setting `weight` to 1)
- one that is on the boundary and evaluates it to `False` (i.e. setting `weight` to 0)
- one that is not on the boundary and evaluates to `True` (i.e. setting `weight` to 100)
- one that is not on the boundary and evaluates to `False` (i.e. setting `weight` to -70)

Moreover, you should try testing with different types as well. For example, floats or strings to observe the behaviour of your code. Finally, you could also consider testing edge cases, such as: infinity, zero, NAN, empty strings, etc.

Below you can find several examples of tests for boundary values:

# Exercise

Try to change any of the values for density and volume to cause an assertion to fail. What do you observe?

```
def test_calculate_weight(calculate_weight):
    assert calculate_weight(0, 3) == 0, "The weight of an object with density of 0 kg/m^3
    assert calculate_weight(50, 0) == 0, "The weight of an object with density of 50 kg/m
    assert calculate_weight(0, 0) == 0, "The weight of an object with density of 0 kg/m^3

    assert calculate_weight(100, 1) == 100, "The weight of an object with density of 100
    assert calculate_weight(45, 2) == 90, "The weight of an object with density of 45 kg/

test_calculate_weight(calculate_weight)
```

# Exercise

You are given the method `calculate_area` again to calculate the *shuttering area* of a concrete rectangular column. The shuttering is a temporary arrangement done for vertical surfaces to support the wet concrete till it attains the desired strength (source: Civilread.com).

Write assertions in `test_calculate_area` that can verify that method `calculate_area` is free of bugs. Make sure you include meaningful messages in case a test fails. Try to write tests that test the boundary values of the `if` statement. For example, tests that evaluate the condition to `True` and/or `False`.

```

```

```
def calculate_area(height, length, breadth):
    """
    Calculates shuttering area of a concrete rectangular column.

    Args:
        height (int): Height of the column.
        length (int): Length of the column.
        breadth (int): Breadth of the column.

    Returns:
        int: The area or -1 if the input is invalid.
    """
    if height <= 0 or length <= 0 or breadth <= 0:
        return -1
    else:
        return (2 * breadth + 2 * length) * height


def test_calculate_area(calculate_area):
    YOUR CODE HERE

test_calculate_area(calculate_area)
```

**Solution**

```
    # Tests, which contain values on the boundary of a condition and evaluate it
    assert calculate_area(0, 10, 20) == -1, "Zero height should result in invali
    assert calculate_area(50, 0, 70) == -1, "Zero length should result in invali
    assert calculate_area(35, 5, 0) == -1, "Zero breadth should result in invali

    # Tests, which do not contain values on the boundary of a condition and eval
    assert calculate_area(-20, 24, 54) == -1, "Negative height should result in
    assert calculate_area(49, -3, 5) == -1, "Negative length should result in in
    assert calculate_area(47, 10, -70) == -1, "Negative breadth should result in

    # Tests, which contain values on the boundary of a condition and evaluate it
    assert calculate_area(1, 25, 4) == 58, "Shuttering area for column with heig
    assert calculate_area(36, 1, 12) == 936, "Shuttering area for column with he
    assert calculate_area(23, 13, 1) == 644, "Shuttering area for column with he

    # Tests, which do not contain values on the boundary of a condition and eval
    assert calculate_area(5, 10, 20) == 300, "Shuttering area for column with he
    assert calculate_area(26, 17, 41) == 3016, "Shuttering area for column with
    assert calculate_area(100, 20, 50) == 14000, "Shuttering area for column wit
```

# Asserts and floating point numbers

Suppose that we are dealing with floating point numbers in our calculations and we would like to run test on them. Try running the code below and think whether the result makes sense:

```
assert 0.1 + 0.2 == 0.3
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[24], line 1
----> 1 assert 0.1 + 0.2 == 0.3

AssertionError:
```

The assertion above fails due to the fact that we are dealing with floating point numbers. For example, we might be using `0.3` in our calculations, but in practice, `0.30004` could be used instead due to the way floating point numbers are stored in memory. (If interested, have a look at the following link: https://0.30000000000000004.com/).

To mitigate this issue, when making assertions for floating point numbers, you can calculate the absolute value of the difference between the expected and actual values and verify that this difference is smaller than some epsilon value:

```
epsilon = 1e-6 # exponential notation for 0.000001
assert abs((0.1 + 0.2) - 0.3) < epsilon
```

# Week 1.7: OOP

> **Attribution**
>
> This chapter reuses material from *Learn Programming for Engineers*. Find out more here.

This chapter contains a lot of information that is useful for improving your programming skills, however you are not required to learn all of it, and not required to memorize everything for the exam.

You should be able to understand the fundamental concepts of classes and object-oriented programming (OOP) in Python as well as the key principles of encapsulation, inheritance, and polymorphism in OOP, which will enable you to better understand and use the classes that are everywhere in Python packages. For example, the class `rv_continuous` in `scipy.stats`, which is used for defining probability distributions, are used heavily in MUDE!

One way to check if you understand this material sufficiently: by the end of Week 1.7 you should realize why OOP is so useful for the continuous parametric distributions we use. For example, recognizing that all distributions can be defined and used in the same way. For example:

- methods like `.pdf()` and `.cdf()` can be used predictably
- you can easily define distributions in terms of parameters or moments
- fitting distributions is also straightforward

For non-probability topics, this will help you recognize why objects in Python packages like Numpy have syntax like `object.mean()` or `object.shape`.

We hope you enjoy this eye-opening experience!

# Classes and Object-Oriented Programming in Python

## Contents


- What are classes?
- Adding parameters to the class
- Encapsulation
- Inheritance
- Polymorphism
- Exercise
- Documentation: Good Practices


Object-Oriented Programming (OOP) is a way of programming that organizes code into reusable blocks of code called *classes*, which bundle data (*attributes*) and behaviors (*methods*) together. When you want to use a class in one of your programs, you make an **object** from that class, which is where the phrase “object-oriented” comes from.

OOP promotes:

- modularity: making multiple modules or functions and then combining them to form a complete system
- reusability: shared attributes and functions can be reused without the need of rewriting code
- better code organization: less code and more reuse

OOP is defined by 4 main concepts (each of which is described below):

- Encapsulation: concept of bundling data and methods that operate on that data into a single unit called a class. Its purpose is to help in organizing and protecting an object’s internal state and behavior;
- Inheritance: mechanism that allows a new class to inherit attributes and methods from an existing class. Its purpose is to promote code reusability, extensibility, and the

creation of class hierarchies, where subclasses can build upon the functionality of their parent classes;
- Polymorphism: ability of different objects to respond to the same method name in a way that is specific to their class. It simplifies code by enabling flexibility and dynamic behavior;
- Abstraction: process of simplifying complex systems by modeling classes based on their essential properties. It reduces complexity, making code more understandable and maintainable.

These concepts may seem abstract for now, but don't be concerned because you have already been using them quite a lot: everything you do in Python uses objects! This should become obvious as you go through this page, after which you should be able to:

1. Understand the fundamental concepts of classes and object-oriented programming (OOP) in Python.
2. Comprehend the key principles of encapsulation, inheritance, and polymorphism in OOP.
3. Define and create classes in Python.
4. Create objects from classes and understand the relationship between classes and objects.

> **Tip**
>
> Click 🚀 –> **Live Code** in the top-right corner of this screen and the live code feature will be enabled.

**There is an interactive exercise at the end of this notebook.**

# What are classes?

Classes can be thought of as blueprints or templates for creating objects.

A **class** is a body of code that defines the **attributes** and **behaviors** required to accurately model something you need for your program. Each class encapsulates both properties (attributes) and functions (methods) related to that type of object.

An **attribute** is a piece of information. In code, an attribute is just a variable that is part of a class.

A **method** is an action that is defined within a class, i.e., just a function that is defined for the class.

An **object** is a particular instance of a class. An object has a certain set of values for all of the attributes (variables) in the class. You can have as many objects as you want for any one class.

Let's consider an example using a rocket ship in a game. When defining the Rocket class, we have to imagine what are properties and actions that are common to all rockets in this hypotethic game. For example, the a very simple rocket will have some x and y coordinates and will be able to move up. Here is what the rocket class can look like in code:

```
class Rocket:
    # Rocket simulates a rocket ship for a game.

    def __init__(self):
        # Each rocket has an (x,y) position.
        self.x = 0
        self.y = 0

    def move_up(self):
        # Each rocket can move along the y-axis.
        self.y += 1
```

Now let's examine how we created this class.

The first line, with the keyword **class**, tells Python that you are about to define a class. The naming convention for classes is the CamelCase, a convention where each letter that starts a word is capitalized, with no underscores in the name. It is recommended that the class name does not have any parentheses after it, unless it inherits from another class (we'll see this better later).

It is good practice to write a comment at the beginning of your class, describing the class. There is a more formal syntax for documenting your classes, but you can wait a little bit to get that formal. For now, just write a comment at the beginning of your class summarizing what you intend the class to do. Writing more formal documentation for your classes will be easy later if you start by writing simple comments now.

One of the first things you do when creating a class is to define the `__init__()` method. The `__init__()` method is called automatically when you create an object from your class and sets the values for any parameters that need to be defined when an object is first created. 0 We call this method a ***constructor***. In this case, The `__init__()` method initializes the x and y values of the Rocket to 0.

The ***self*** part is a syntax that allows you to access variables and methods from anywhere else in the class. The word “self” refers to the current object that you are working with. Basically, all methods in a class need the *self* object as their first argument, so they can access any attribute that is part of the class.

Methods define a specific action that the class can do. A method is just a function that is part of a class. Since it is just a function, you can do anything with a method that you learned about with functions. Each method generally accepts at least one argument, the value **self**. This is a reference to the particular object that is calling the method.

With this code, we just defined the class, which is a blueprint. To define the actual object, we have to create an *instance* of the class. In other words, you create a variable that takes all of the attributes and methods as the Rocket class.

This can be done simply as follows:

```
class Rocket:
    # Rocket simulates a rocket ship for a game,

    def __init__(self):
        # Each rocket has an (x,y) position.
        self.x = 0
        self.y = 0

    def move_up(self):
        # Increment the y-position of the rocket.
        self.y += 1

# Create a Rocket object.
my_rocket = Rocket()
print(my_rocket)
```

Once you have a class, you can define an object and use its methods. Here is how you might define a rocket and have it start to move up:

```
# Create a Rocket object, and have it start to move up.
my_rocket = Rocket()
print(f"Rocket altitude: {my_rocket.y}", )

my_rocket.move_up()
print(f"Rocket altitude: {my_rocket.y}")

my_rocket.move_up()
print(f"Rocket altitude: {my_rocket.y}")
```

To access an object’s variables or methods, you give the name of the object and then use *dot notation* to access the variables and methods. So to get the y-value of `my_rocket`, you use `my_rocket.y`.

To use the `move_up()` method on my_rocket, you write `my_rocket.move_up()`. This tells Python to apply the method `move_up()` to the object `my_rocket`. Python finds the y-value associated with `my_rocket` and adds 1 to that value. This process is repeated several times, and you can see from the output that the y-value is in fact increasing.

# Making multiple objects from a class

One of the goals of object-oriented programming is to create reusable code. Once you have written the code for a class, you can create as many objects from that class as you need. It is worth mentioning at this point that classes are usually saved in a separate file, and then imported into the program you are working on. So you can build a library of classes, and use those classes over and over again in different programs. Once you know a class works well, you can leave it alone and know that the objects you create in a new program are going to work as they always have.

You can see this “code reusability” already when the Rocket class is used to make more than one Rocket object. Each object is its own instance of that class, with its own separate variables. All of the objects are capable of the same behavior, but each object’s particular actions do not affect any of the other objects. Here is the code that made a fleet of Rocket objects:

```
class Rocket:
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self):
        # Each rocket has an (x,y) position.
        self.x = 0
        self.y = 0

    def move_up(self):
        # Increment the y-position of the rocket.
        self.y += 1

# Create a fleet of 3 rockets, and store them in a list.
rocket_1 = Rocket()
rocket_2 = Rocket()
rocket_3 = Rocket()

# Show that each rocket is a separate object.
print(rocket_1, rocket_2, rocket_3)
```

You can see that each rocket is at a separate place in memory and therefore printed in a different way.

You can show that each rocket has its own x and y values by moving just one of the rockets:

```
# Move the third rocket up.
rocket_3.move_up()

# Show that only the third rocket has moved.
print(f"Rocket altitudes: {rocket_1.y}, {rocket_2.y}, {rocket_3.y}")
```

## A quick check-in

If all of this makes sense, then the rest of your work with classes will involve learning a lot of details about how classes can be used in more flexible and powerful ways. If this does not make any sense, you could try a few different things:

- Reread the previous sections, and see if things start to make any more sense.
- Type out these examples in your own editor, and run them. Try making some changes, and see what happens.
- Read on. The next sections are going to add more functionality to the Rocket class. These steps will involve rehashing some of what has already been covered, in a slightly different way.

Classes are a huge topic, and once you understand them you will probably use them for the rest of your life as a programmer. If you are brand new to this, be patient and trust that things will start to sink in.

# Adding parameters to the class

The Rocket class so far is very simple. It can be made a little more interesting with some refinements to the `__init__()` method, and by the addition of some methods.

## Accepting parameters for the __init__() method

The `__init__()` method is run automatically one time when you create a new object from a class. The `__init__()` method for the Rocket class so far is pretty simple. But we can easily add keyword arguments so that new rockets can be initialized at any position:

```
```

```
class Rocket:
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y

    def move_up(self):
        # Increment the y-position of the rocket.
        self.y += 1
```

Now when you create a new Rocket object you have the choice of passing in arbitrary initial values for x and y:

```
# Make a series of rockets at different starting places.
rockets = []
rockets.append(Rocket())
rockets.append(Rocket(0,10))
rockets.append(Rocket(100,0))

# Show where each rocket is.
for index, rocket in enumerate(rockets):
    print(f"Rocket {index} is at (x,y)=({rocket.x}, {rocket.y}).")
```

# Accepting parameters in a method

The `__init__()` method is just a special method that serves a particular purpose, which is to help create new objects from a class. Any method in a class can accept parameters of any kind. With this in mind, the `move_up()` method can be made much more flexible. By accepting keyword arguments, the `move_up()` method can be rewritten as a more general `move_rocket()` method. This new method will allow the rocket to be moved any amount, in any direction:

```
class Rocket():
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y

    def move_rocket(self, x_increment=0, y_increment=1):
        # Move the rocket according to the paremeters given.
        #  Default behavior is to move the rocket up one unit.
        self.x += x_increment
        self.y += y_increment
```

The paremeters for the move() method are named x_increment and y_increment rather than x and y. It's good to emphasize that these are changes in the x and y position, not new values for the actual position of the rocket. By carefully choosing the right default values, we can define a meaningful default behavior. If someone calls the method `move_rocket()` with no parameters, the rocket will simply move up one unit in the y-direciton. Note that this method can be given negative values to move the rocket left or right:

```
class Rocket():
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y

    def move_rocket(self, x_increment=0, y_increment=1):
        # Move the rocket according to the paremeters given.
        #  Default behavior is to move the rocket up one unit.
        self.x += x_increment
        self.y += y_increment


# Create three rockets.
rockets = [Rocket() for x in range(0,3)]

# Move each rocket a different amount.
rockets[0].move_rocket()
rockets[1].move_rocket(10,10)
rockets[2].move_rocket(-10,0)

# Show where each rocket is.
for index, rocket in enumerate(rockets):
    print(f"Rocket {index} is at ({rocket.x}, {rocket.y}).")
```

# Adding a new method

One of the strengths of object-oriented programming is the ability to closely model real-world phenomena by adding appropriate attributes and behaviors to classes. One of the jobs of a team piloting a rocket is to make sure the rocket does not get too close to any other rockets. Let's add a method that will report the distance from one rocket to any other rocket. Note how this method uses another instance of the same class as one of its arguments!

If you are not familiar with distance calculations, there is a fairly simple formula to tell the distance between two points if you know the x and y values of each point. This new method performs that calculation, and then returns the resulting distance.

```
```

```
from math import sqrt

class Rocket:
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y

    def move_rocket(self, x_increment=0, y_increment=1):
        # Move the rocket according to the paremeters given.
        #  Default behavior is to move the rocket up one unit.
        self.x += x_increment
        self.y += y_increment

    def get_distance(self, other_rocket):
        # Calculates the distance from this rocket to another rocket,
        #  and returns that value.
        distance = sqrt((self.x-other_rocket.x)**2+(self.y-other_rocket.y)**2)
        return distance

# Make two rockets, at different places.
rocket_0 = Rocket()
rocket_1 = Rocket(10,5)

# Show the distance between them.
distance = rocket_0.get_distance(rocket_1)
print(f"The rockets are {distance:.6f} units apart.")
```

Hopefully these short refinements show that you can extend a class' attributes and behavior to model the phenomena you are interested in as closely as you want. The rocket could have a name, a crew capacity, a payload, a certain amount of fuel, and any number of other attributes. You could define any behavior you want for the rocket, including interactions with other rockets and launch facilities, gravitational fields, and whatever you need it to! There are techniques for managing these more complex interactions, but what you have just seen is the core of object-oriented programming.

# Encapsulation

As we have mentioned before using dictionaries is not useful to create objects that consist of several attributes. Encapsulation entails the wrapping of attributes and methods within one unit. This is beneficial as it enables engineers to write code that is easy to maintain. For example, if you add a new attribute or a method to a class, you will not need to update all your objects, but only the class itself.

A nice feature that encapsulation provides is private attributes and private methods. Those are units, which are meant to only be used internally in a class and not accessed outside of

it. By convention programmers should not access them via the object. Furthermore, in order to create a private attribute or a private method, you need to put 2 leading underscores (`_`) before their name.

You can think of the `__init__` method for an example. You are not supposed to call the method, as it is automatically called when you create a new object. Furthermore, note that it has 2 leading and 2 trailing underscores. This is meant to show that this method is resereved in Python. **Therefore, you should not make attributes or methods that have both leading and trailing underscores, because you may mess up how Python works**. Besides the `__init__` method, there are more built-in methods that start and end with 2 underscores. These are called **magic methods**, and determine the behaviour when certain operations (for example: +, - or `print()`) are performed on an object.

Have a look at the Rocket class below, which contains a private attribute `creation_date`. We will call this attribute `__creation_date` to tell Python that we want it to be private. This attribute is set inside the `__init__` method and should not be accessed outside the class unless we create a method, which returns it:

```
from math import sqrt
import datetime

class Rocket:
    # Rocket simulates a rocket ship for a game,
    #  or a physics simulation.

    def __init__(self, x=0, y=0):
        # Each rocket has an (x,y) position.
        self.x = x
        self.y = y
        self.__creation_time = datetime.datetime.now()

    def move_rocket(self, x_increment=0, y_increment=1):
        # Move the rocket according to the paremeters given.
        #  Default behavior is to move the rocket up one unit.
        self.x += x_increment
        self.y += y_increment

    def get_distance(self, other_rocket):
        # Calculates the distance from this rocket to another rocket,
        #  and returns that value.
        distance = sqrt((self.x-other_rocket.x)**2+(self.y-other_rocket.y)**2)
        return distance

    def get_creation_time(self):
        # Returns the time the Rocket was made.
        return self.__creation_time

# Make a rocket.
rocket_0 = Rocket()

# Try to get the creation time via a method.
date = rocket_0.get_creation_time()
print(f"Rocket was made in: {date}")

# Try to get the creation time directly.
date = rocket_0.__creation_time
print(f"Rocket was made in: {date}")
```

As seen in the example above, we can only access `__creation_time` via the method `get_creation_time` and get an `AttributeError` if we attempt to directly use the *dot notation* on the object `rocket_0`.

# Inheritance

Another of the most important goals of the object-oriented approach to programming is the creation of stable, reliable, reusable code. If you had to create a new class for every kind of object you wanted to model, you would hardly have any reusable code. In Python and any other language that supports OOP, one class can **inherit** from another class. This means you can base a new class on an existing class; the new class *inherits* all of the attributes and behavior of the class it is based on. A new class can override any undesirable attributes or behavior of the class it inherits from, and it can add any new attributes or behavior that

are appropriate. The original class is called the **parent** class or **superclass**, and the new class is a **child** or **subclass** of the parent class.

The child class inherits all attributes and behavior from the parent class, but any attributes that are defined in the child class are not available to the parent class. This may be obvious to many people, but it is worth stating. This also means a child class can ***override*** behavior of the parent class. If a child class defines a method that also appears in the parent class, objects of the child class will use the new method rather than the parent class method.

To better understand inheritance, let's look at an example of a class that can be based on the Rocket class.

# The Shuttle class

If you wanted to model a space shuttle, you could write an entirely new class. But a space shuttle is just a special kind of rocket. Instead of writing an entirely new class, you can inherit all of the attributes and behavior of a Rocket, and then add a few appropriate attributes and behavior for a Shuttle.

One of the most significant characteristics of a space shuttle is that it can be reused. So the only difference we will add at this point is to record the number of flights the shutttle has completed. Everything else you need to know about a shuttle has already been coded into the Rocket class.

Here is what the Shuttle class looks like:

```
class Shuttle(Rocket):
    # Shuttle simulates a space shuttle, which is really
    #  just a reusable rocket.

    def __init__(self, x=0, y=0, flights_completed=0):
        super().__init__(x, y)
        self.flights_completed = flights_completed

shuttle = Shuttle(10,0,3)
print(shuttle.x)
```

When a new class is based on an existing class, you write the name of the parent class in parentheses when you define the new class:

```
class NewClass(ParentClass):
```

The `__init__()` function of the new class needs to call the `__init__()` function of the parent class. The `__init__()` function of the new class needs to accept all of the parameters required to build an object from the parent class, and these parameters need to be passed to the `__init__()` function of the parent class. The `super().__init__()` function takes care of this:

```
class NewClass(ParentClass):

    def __init__(self, arguments_parent_class, arguments_new_class):
        super().__init__(arguments_parent_class)
        # Code for initializing an object of the new class.
```

The `super()` function passes the *self* argument to the parent class automatically.

Inheritance is a powerful feature of object-oriented programming. Using just what you have seen so far about classes, you can model an incredible variety of real-world and virtual phenomena with a high degree of accuracy. The code you write has the potential to be stable and reusable in a variety of applications.

# Polymorphism

Another important goal of the object-oriented approach to programming is to provide flexibility of your code. This can be achived by Polymorphism, which entails that an entity is able to take multiple forms. In Python polymorphism allows us to create methods in a child class with the same name as a method in a parent class. This would mean that a method can serve one purpose in a parent class and different one in a child class.

Child classes inherit all the methods of their parent classes, however, sometimes those methods need to be modified to fit the function of the child. This is achieved by reimplementing the parent methods in the child class.

To better understand polymorphism, let's look at an example of a class that can be based on the Shuttle class and transitively on the Rocket class as well.

## The ImprovedShuttle class

Our Shuttle class already improves the basic Rocket class, however, the information we receive from the Rocket class such as `get_distance` is very limited. This is because we currently only get information about the absolute distance, but we do not know the direction, which we need to face to get to that place the fastest.

Therefore, we will create an improved Shuttle, which will be based on the initial Shuttle and will provide better distance information such as angle in which we need to rotate. The formula used is based on taking arctangent of the 2-dimension distances and transforming from radians to degrees.

Here is what the ImprovedShuttle class looks like:

```
from math import atan, pi, sqrt

class ImprovedShuttle(Shuttle):
    # Improved Shuttle that provides better distance information
    #  such as angle.

    def __init__(self, x=0, y=0, flights_completed=0):
        super().__init__(x, y)
        self.flights_completed = flights_completed

    def get_distance(self, other_rocket):
        # Calculates the distance from this rocket to another rocket,
        #  the angle to rotate to face the other rocket,
        #  and returns those values.
        distance = super().get_distance(other_rocket)
        angle = atan((other_rocket.y - self.y) / (other_rocket.x - self.x)) * (180 / pi)
        return distance, angle

improvedShuttle = ImprovedShuttle(10,0,3)
otherShuttle = ImprovedShuttle(13, 3)

# Show the distance between them.
distance, angle = improvedShuttle.get_distance(otherShuttle)
print(f"The shuttles are {distance:.6f} units apart.")
print(f"The angle the initial shuttle needs to rotate in case it needs to go to the other
```

As you can see in the example above, since ImprovedShuttle inherits Shuttle and Shuttle inherits Rocket, then transitively ImprovedShuttle is a child of Rocket class and has access to the parent `get_distance` method. It is possible to access that parent method by making a `super().get_distance()` call.

As a result, class ImprovedShuttle has ***overridden*** Rocket's get_distance. This means that it has reimplemented the parent's method.

It is important to mention that it is not necessary to override (reimplement) every method in the parent class when using inheritance, but if needed, it is possible.

> Note: ImprovedShuttle's get_distance() now returns two outputs, while the parent class only returns one. Imagine you are looping a list containing a mix of Rockets and ImprovedShuttles to store their distance in an array (with as many elements as the

> length of the lists); this difference in the output may require some extra lines of code to handle potential problems.

# Exercise

**Exercise**

It is time to practice what you have learned so far about classes, inheritance, and polymorphism. Given the predefined Person class, create a Student and Teacher classes that inherit from People and have the additional following requirements

1. Create a class Student:
    1. Make class Student inherit class Person;
    2. Add parameters `start year` and `GPA grade` in the `__init__` method and reuse the parent constructor for the other attributes;
    3. Create a method `change_grade`, which sets a new grade for the student;
    4. Override the `introduce_yourself` method to account for the 2 new fields (start year and GPA grade). In your overriding try to reuse the parent `introduce_yourself` method implementation by calling `super()`;
2. Create a class Teacher:
    1. Make class Teacher inherit class Person;
    2. Add an attribute called `students` of type `set` in the `__init__` method and reuse the parent constructor for the other attributes. Remember that a set is a collection of elements that contains no duplicates. A set can be initialised in multiple ways. For example, `my_set = set()` or `my_set = {}`;
    3. Create a method `add_student`, which adds a student to the student set of a teacher. Elements to a set can be added using the method `add()`. For example, `my_set.add(3)`;
    4. Create a method `print_classroom`, which prints the introductions of all the students in the classroom of the teacher. (Hint: call `introduce_yourself` on every Student object in the set).
3. Similarly to the previous exercise, show your new classes are working properly by creating objects for each of them and calling their respective methods. Furthermore, add the necessary documentation for the classes and methods.

```
class Person():
    def __init__(self, name, age, country_of_origin):
        """Constructor to initialise a Person object.

        Keyword arguments:
        name -- the name of the person
        age -- the age of the person
        country_of_origin -- the country the person was born
        """
        self.__name = name
        self.age = age
        self.country_of_origin = country_of_origin

    def introduce_yourself(self):
        """Prints a brief introduction of the person."""
        print(f"Hello, my name is {self.__name}, I am {self.age} and I am from {self.coun

    def get_name(self):
        """Gets the name of a person."""
        return self.__name
```

```
"""Use this code cell to type your answer."""
```

### Solution

**If you have done the exercise correctly, the following output should be printed when you run the cell above.**

```
Showing how class Student is defined:
New GPA grade of student is 7.2.

Showing how class Teacher is defined:
Hello, my name is Joe, I am 42 and I am from Germany.
The classroom consists of the following students:
Hello, my name is Mike, I am 22 and I am from Italy.
My GPA grade is 7.2 and I started studying in year 2021.
```

## Ful working out

```
class Student(Person):
    def __init__(self, name, age, country_of_origin, start_year, gpa_grade):
        """Constructor to initialise a Student object.

        Keyword arguments:
        name -- the name of the student
        age -- the age of the student
        country_of_origin -- the country the student was born
        start_year -- the year the education of the student began
        gpa_grade -- current gpa_grade of a student
        """
        super().__init__(name, age, country_of_origin)
        self.start_year = start_year
        self.gpa_grade = gpa_grade

    def introduce_yourself(self):
        """Prints a brief introduction of the student."""
        super().introduce_yourself()
        print(f"My GPA grade is {self.gpa_grade} and I started studying in year

    def change_grade(self, new_grade):
        """Modifies current GPA grade of the student."""
        self.gpa_grade = new_grade

class Teacher(Person):
    def __init__(self, name, age, country_of_origin):
        """Constructor to initialise a Teacher object.

        Keyword arguments:
        name -- the name of the teacher
        age -- the age of the teacher
        country_of_origin -- the country the teacher was born
        """
        super().__init__(name, age, country_of_origin)
        self.students = set()

    def add_student(self, student):
        """Adds a student to the classroom of a teacher.

        Keyword arguments:
        studnet -- student to add to the classroom
        """
        self.students.add(student)

    def print_classroom(self):
        """Prints the classroom of a teacher."""
        print("The classroom consists of the following students:")
        for student in self.students:
            student.introduce_yourself()


print("Showing how class Student is defined:")
student = Student("Mike", 22, "Italy", "2021", 7.3)

student.change_grade(7.2)
print(f"New GPA grade of student is {student.gpa_grade}.")
print()

print("Showing how class Teacher is defined:")
teacher = Teacher("Joe", 42, "Germany")
teacher.introduce_yourself()
teacher.add_student(student)
```

```
teacher.print_classroom()
```

# Documentation: Good Practices

The Python Organization publishes best practices for writing Python code, which covers important aspects like documentation. This section illustrates a few concepts that are important of OOP.

If you would like to know where these concepts come from, try having a look at PEP 8, which is the style guide for writing Python code. Another document, PEP 257, covers conventions for writing docstrings. PEP 257 is covered here, as it has more relevant material related to documentation of classes and methods.

## Naming conventions

Class names should be written in *CamelCase*, with an initial capital letter and any new word capitalized. There should be no underscores in your class names. For example, if you have a super cool class, you should name it `ASuperCoolClass`.

Method names should always have an initial lowercase letter, similar to the regulations on naming functions. Furthermore, the first argument of every class method should be the keyword `self`. For example, the method signature (the same as function signature - the function name and function arguments) of a method `move_up()` should be `move_up(self):`. Nevertheless, if a method name contains more than 1 word, then the words should be separated by underscore `_`. For instance, if you have an important method, you should name it `important_method`.

## Docstrings

A docstring is a string literal that appears at the start of classes/methods.

By convention, docstrings begin and end with 3 quotation marks: `"""docstring"""` and should be placed right below the signature of a method or the class signature. A rule of thumb is to have 1 line explaning what a method does, followed by 1 blank line, followed by zero/one/multiple lines explaning what each of the parameter does:

```
```

```
class Rocket():
    """Rocket simulates a rocket ship for a game,
    or a physics simulation.
    """

    def __init__(self, x=0, y=0):
        """Constructor to initialise a Rocket object.

        Keyword arguments:
        x -- x coordinate (default 0)
        y -- y coordinate (default 0)
        """
        self.x = x
        self.y = y

    def move_rocket(self, x_increment=0, y_increment=1):
        """Moves the rocket according to the paremeters given.

        Keyword arguments:
        x_increment -- units to move in x dimension (default 0)
        y_increment -- units to move in y dimension (default 1)
        """
        self.x += x_increment
        self.y += y_increment

    def get_distance(self, other_rocket):
        """Calculates the distance from this rocket to another rocket
        and returns that value.

        Keyword arguments:
        other_rocket -- the other rocket, which distance to compare to
        """
        distance = sqrt((self.x-other_rocket.x)**2+(self.y-other_rocket.y)**2)
        return distance

# Check the documentation of Rocket class
help(Rocket)
```

Note that the *self* argument is not explained the docstrings, because its use is implicitly known.

# Week 1.8: SymPy

> **Attribution**
>
> This chapter reuses material from *Python for Engineers*. Find out more here.

This week you'll learn how to use SymPy, a python package which does symbolical math for you!

You can use this to solve the more elaborate mathematical procedures, or check you hand-written math!

# SymPy

## Contents


- Why SymPy?
- First steps
- Variables
- Expressions
- Functions
- Evaluating expressions
- Some examples
- Gotchas


SymPy is a Python library for symbolic mathematics. You can use it as a tool to evaluate basic and complicated math operations, whenever they become tedious or error prone.

For example, SymPy is able to solve the differential equation $y'' - y = e^t$ for you!

```
import sympy as sym
y = sym.Function('y')
t = sym.symbols('t')
sym.dsolve(sym.Eq(y(t).diff(t, t) - y(t), sym.exp(t)), y(t))
```

$$y(t) = C_2 e^{-t} + \left(C_1 + \frac{t}{2}\right) e^t$$

On this page, you'll get an introduction of the SymPy-basics. This introduction is derived from the introduction into SymPy from Jason Moore (CC-BY licensed) (Moore, 2023) and the SymPy Tutorial (Copyright (c) 2006-2023 SymPy Development Team) (SymPy Development Team, 2023).

# Why SymPy?

… and not Wolfram Alpha, Mathematica, Maple, GeoGebra, your fancy calculator, or …?

> First off, SymPy is completely free. It is open source, and licensed under the liberal BSD license, so you can modify the source code and even sell it if you want to. This contrasts with popular commercial systems like Maple or Mathematica that cost hundreds of dollars in licenses.
>
> Second, SymPy uses Python. Most computer algebra systems invent their own language. Not SymPy. SymPy is written entirely in Python, and is executed entirely in Python. This means that if you already know Python, it is much easier to get started with SymPy, because you already know the syntax (and if you don't know Python, it is really easy to learn). We already know that Python is a well-designed, battle-tested language. The SymPy developers are confident in their abilities in writing mathematical software, but programming language design is a completely different thing. By reusing an existing language, we are able to focus on those things that matter: the mathematics.
>
> Another computer algebra system, Sage also uses Python as its language. But Sage is large, with a download of over a gigabyte. An advantage of SymPy is that it is lightweight. In addition to being relatively small, it has no dependencies other than Python, so it can be used almost anywhere easily. Furthermore, the goals of Sage and the goals of SymPy are different. Sage aims to be a full featured system for mathematics, and aims to do so by compiling all the major open source mathematical systems together into one. When you call some function in Sage, such as integrate, it calls out to one of the open source packages that it includes. In fact, SymPy is included in Sage. SymPy on the other hand aims to be an independent system, with all the features implemented in SymPy itself.
>
> A final important feature of SymPy is that it can be used as a library. Many computer algebra systems focus on being usable in interactive environments, but if you wish to automate or extend them, it is difficult to do. With SymPy, you can just as easily use it in an interactive Python environment or import it in your own Python application. SymPy also provides APIs to make it easy to extend it with your own custom functions. (SymPy Development Team, 2023)

# First steps

Let's take an example. Say we wanted to use the built-in Python functions to compute square roots. We might do something like this:

```
import math
math.sqrt(8)
```

$$2.82842712474619$$

Here we got an approximate result, which is not the exact square root of 8. If all we cared about was the decimal form of the square root of 8, we would be done.

But suppose we want to go further. Recall that $\sqrt{8} = \sqrt{4 \cdot 2} = 2\sqrt{2}$ . We would have a hard time deducing this from the above result. This is where symbolic computation comes in. With a symbolic computation system like SymPy, square roots of numbers that are not perfect squares are left unevaluated by default

We'll import SymPy as follows:

```
import sympy as sym
```

Since SymPy works with mathematical symbols it's nice to view SymPy objects in a format that is similar to the math in a textbook. Executing `sym.init_printing()` at the beginning of your Jupyter Notebook will ensure that SymPy objects render as typeset mathematics.

```
sym.init_printing()
```

Now we can apply SymPy to our simple root problem:

```
sym.sqrt(8)
```

$$2\sqrt{2}$$

It's not only exact, but also simplified!

# Variables

The use of computer algebra systems like SymPy is much more powerful, as they're able to compute expressions with variables.

We need to define variable explicitly, so that the correct object is created. For example, let's create two variables `x` and `y` representing $x$ and $y$:

```
x, y = sym.symbols('x y')
```

**Test yourself!**

Review the SymPy documentation and create symbols `q1`, `q2`, … `q10` using `sym.var()` without providing a full list of `q1, q2, q3, q4, q5, q6, q7, q8, q9, q10`. Tip: `sym.var()` is an extension to `sym.symbols`.

Click 🚀 –> **Live Code** to activate live coding!

```
sym.var(    )
```

**Answer**

```
sym.var('q1:11')
```

# Expressions

Variables alone have little meaning, we need expressions to do some proper math. Let's define a symbolic expression, representing the mathematical expression $x + 2y$. The most basic way to construct expressions is with the standard Python operators `+`, `-`, `*`, `/`, and `**`. For example:

```
expr = x + 2*y
expr
```

$x + 2y$

Note that we wrote `x + 2*y` just as we would if `x` and `y` were ordinary Python variables. But in this case, instead of evaluating to something, the expression remains as just $x + 2y$. Now let us play around with it:

```
expr + 1
```

$x + 2y + 1$

```
expr - x
```

$2y$

Notice something in the above example. When we typed `expr - x`, we did not get $x + 2y - x$, but rather just $2y$. The $x$ and the $-x$ automatically canceled one another. This is similar to how $\sqrt{8}$ automatically turned into $2\sqrt{2}$ above. This isn't always the case in SymPy, however; consider this example:

```
x*expr
```

$$x\left(x + 2y\right)$$

Aside from obvious simplifications like $x - x = 0$ and $\sqrt{8} = 2\sqrt{2}$, most simplifications are not performed automatically. A general formula to simplify formulas is `sym.simplify`, which attempts to apply all of these functions in an intelligent way to arrive at the simplest form of an expression:

```
expr2 = sym.sin(x)**2+sym.cos(x)**2
expr2
```

$$\sin^2\left(x\right) + \cos^2\left(x\right)$$

```
sym.simplify(expr2)
```

$$1$$

Only attempt simplification on expressions that are several lines of text. Larger expressions become increasingly computationally intensive to simplify and there is generally no need to do so.

In many cases, our expressions include a left- and right-hand-side. We cannot use `==` for that (`==` checks the similarity of the python-object), but we can use the `sym.Eq` function to create the symbolic equality $x + 2y = 3$:

```
sym.Eq(expr,3)
```

$$x + 2y = 3$$

When setting up these expressions, watch out! Be careful with numbers, as SymPy may not intepret them as expected. For example:

```
1/2*x
```

$$0.5x$$

Python does the division before it is multiplied by `x`, thus a floating point value is created. To fix this you can use the `sym.S()` function to "sympify" numbers:

```
sym.S(1)/2*x
```

$$\frac{x}{2}$$

Or you can ensure the symbol comes first in the division operation:

```
x/2
```

$$\frac{x}{2}$$

**Test yourself!**

Create an expression for the normal distribution function:

$$\frac{1}{\sqrt{2\pi\sigma}}e^{\frac{(x-\mu)^2}{2\sigma^2}}$$

All variables have been defined with:

```
sym.var('x, sigma, mu')
```

Click 🚀 –> **Live Code** to activate live coding!

```
expr =
```

# Functions

You will also work with undefined mathematical functions in addition to variables. These will play an important role in setting up differential equations, where you typically don't know the function, but only its derivative(s). You can create arbitrary functions of variables. In this case, you make a function of $x$. First create the function name:

```
f = sym.Function('f')
```

Now you can create functions of one or more variables like so:

```
f(x)
```

$f(x)$

The same `UndefinedFunction` can be used to create multivariate functions:

```
f(x,y)
```

$f(x, y)$

**Test yourself!**

Create a function $H(x, y, z)$

All variables have been defined with:

```
x, y, z = sym.symbols('x, y, z')
```

Click 🚀 –> **Live Code** to activate live coding!

```
function =
```

# Evaluating expressions

SymPy expressions can be evaluated numerically in several ways. Let's say you have an expression defined as follows:

```
a, omega, b, t = sym.symbols('a, omega, b, t')
f = sym.Function('f')
expr3 = a*sym.sin(omega) + sym.Abs(f(t))/sym.sqrt(b)
expr3
```

$$a \sin(\omega) + \frac{|f(t)|}{\sqrt{b}}$$

And you have some values for which you want to evaluate of the expression:

```
values = {omega: sym.pi/4, a: 2, f(t): -12, b: 25}
```

You can evalute the expression using `sym.subs`:

```
expr3.subs(values)
```

$$\sqrt{2} + \frac{12}{5}$$

Notice how the square root and fraction do not automatically reduce to their decimal equivalents. To do so, you must use the `evalf()` method:

```
expr3.subs(values).evalf()
```

$$3.81421356237310$$

To obtain machine precision floating point numbers directly and with more flexibility, it is better to use the `sym.lambdify()` function to convert the expression to a Python function. When using `sym.lambdify()`, all symbols and functions should be converted to numbers, so first identify what symbols and functions make up the expression.

```
eval_expr3 = sym.lambdify((omega, a, f(t), b), expr3)
```

`sym.lambdify()` generates a Python function and, in this case, we store that function in the variable `eval_expr3`. You can see what the inputs and outputs of the function are with `help()`:

```
help(eval_expr3)
```

```
Help on function _lambdifygenerated:

_lambdifygenerated(omega, a, _Dummy_38, b)
    Created with lambdify. Signature:

    func(omega, a, f, b)

    Expression:

    a*sin(omega) + Abs(f(t))/sqrt(b)

    Source code:

    def _lambdifygenerated(omega, a, _Dummy_38, b):
        return a*sin(omega) + abs(_Dummy_38)/sqrt(b)


    Imported modules:
```

This function operates on and returns floating point values. However, it also support arrays of floats. For example:

```
eval_expr3(3.14/4, 2, -12, [25, 26, 27])
```

```
array([3.81365036, 3.76704398, 3.72305144])
```

**Test yourself!**

Create a symbolic expression representing Newton's Law of Universal Gravitation. Use `sym.lambdify()` to evaluate the expression for two mass of 5.972E24 kg and 80 kg at a distance of 6371 km apart to find the gravitational force in Newtons. $G$ equals $6.67430 \cdot 10^{-11}$

All variables have been defined with:

```
G, m_1, m_2, r = sym.symbols('G, m_1, m_2, r')
```

Click 🚀 –> **Live Code** to activate live coding!

```
answer =
```

# Some examples

SymPy has many possibilities. Check out the documentation if you're curious. Some examples are shown below.

Take the derivative of $\sin(x)e^x$

```
sym.diff(sym.sin(x)*sym.exp(x),x)
```

$e^x \sin(x) + e^x \cos(x)$

Compute $\int (e^x \sin(x) + e^x \cos(x)) dx$.

```
sym.integrate(sym.exp(x)*sym.sin(x) + sym.exp(x)*sym.cos(x), x)
```

$e^x \sin(x)$

Compute $\int_{-\infty}^{\infty} \sin(x^2) dx$.

```
```

```
sym.integrate(sym.sin(x**2), (x, -sym.oo, sym.oo))
```

$$\frac{\sqrt{2}\sqrt{\pi}}{2}$$

Find $\lim_{x \to 0} \frac{\sin(x)}{x}$.

```
sym.limit(sym.sin(x)/x, x, 0)
```

$$1$$

Solve $x^2 - 2 = 0$.

```
sym.solve(sym.Eq(x**2 - 2,0), x)
```

$$\left[-\sqrt{2}, \sqrt{2}\right]$$

Find the eigenvalues of $\begin{bmatrix} 1 & 2 \\ 2 & 2 \end{bmatrix}$.

```
sym.Matrix([[1, 2], [2, 2]]).eigenvals()
```

$$\left\{\frac{3}{2} - \frac{\sqrt{17}}{2} : 1, \ \frac{3}{2} + \frac{\sqrt{17}}{2} : 1\right\}$$

Print $\int_0^{\pi} \cos^2(x) dx$ using $\LaTeX$.

```
sym.latex(sym.Integral(sym.cos(x)**2, (x, 0, sym.pi)))
```

```
'\\int\\limits_{0}^{\\pi} \\cos^{2}{\\left(x \\right)}\\, dx'
```

# Gotchas

Trial and error is an approach for many programmers, but you may want to prevent frequently made mistakes. Take a look at SymPy's documentation on gotchas to accelerate your learning!

> **Attribution**
>
> Written by Tom van Woudenberg
>
> This page reuses CC BY content from Lanzafame *et al.* (n.d.). Find out more here

# Gurobi

## Contents


- Before Starting
- Installation Instructions


Gurobi is an optimizer written in Python. It's not open-source, but an academic license is available. This page shows how to activate the license. General instructions are available on their support website as well, if additional information is needed.

## Before Starting

The software is easy to install, but keep in mind:

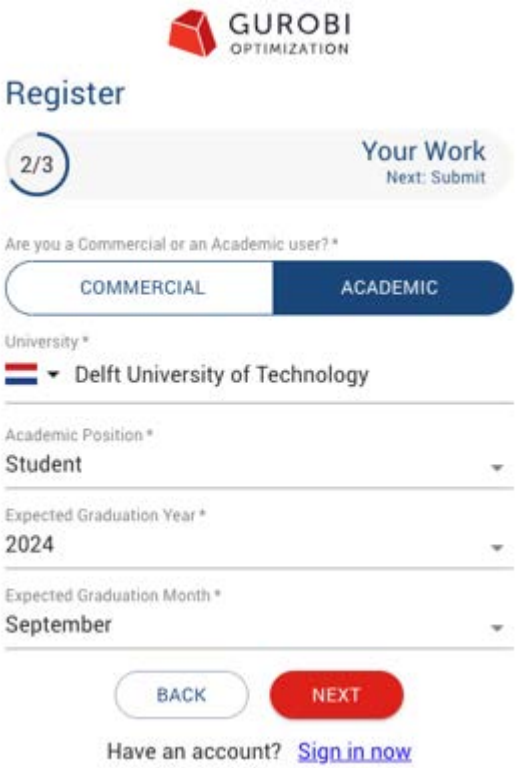
1. The installer will ask you to restart your computer
2. To use Gurobi you need to have a TU Delft IP address. This means if you are *not* working on campus, you will have to use a VPN to activate the license, described below.

|**How to get a VPN connection with TU Delft IP Address** |The software eduVPN is described here. After reading the description, click on the "manual" button and select "Installing eduVPN for students," which will take you to the the download page.

## Installation Instructions

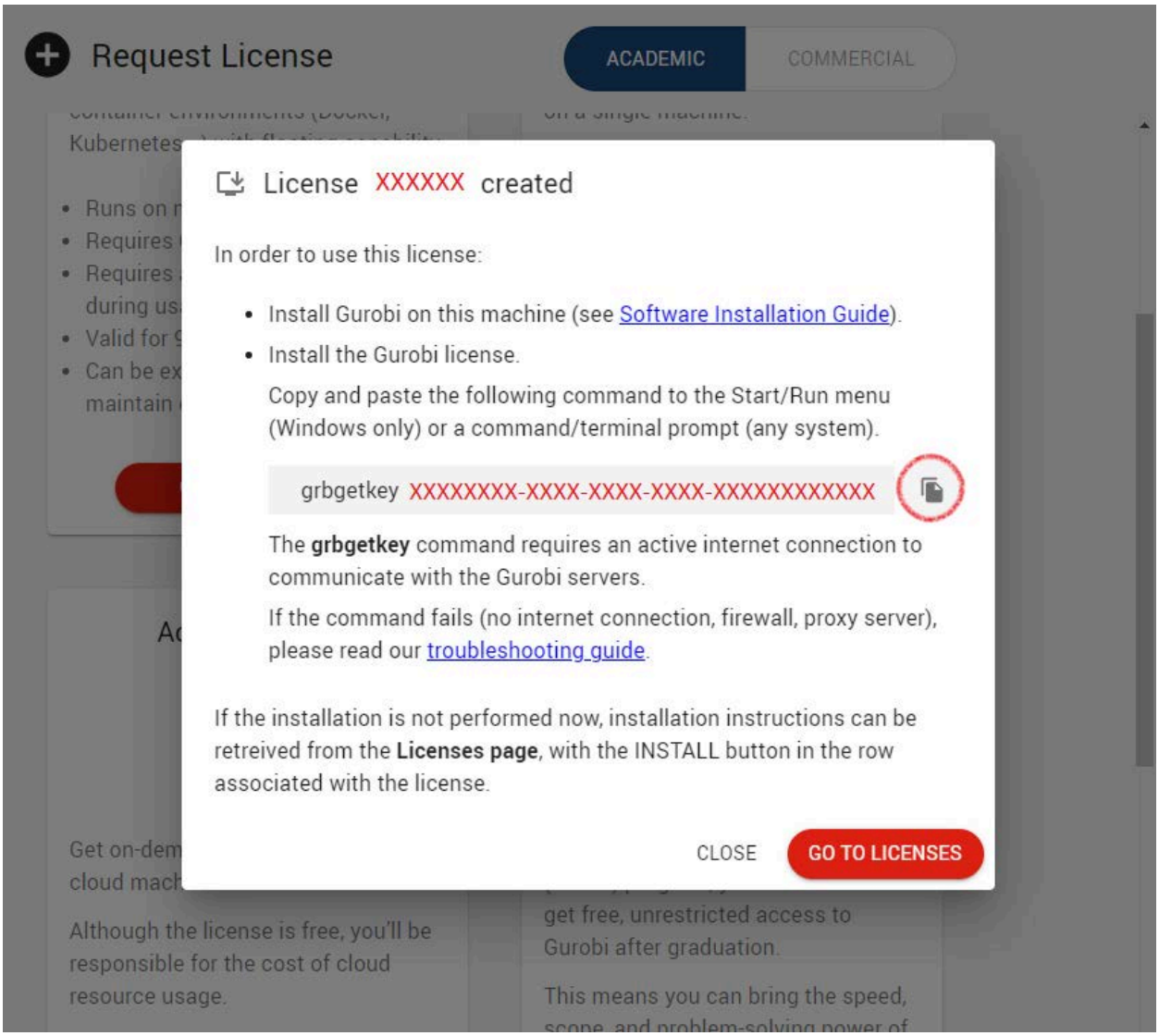
Please do the following steps:

1. Connect your computer to the TU Delft network by connecting to Eduroam or the TU Delft VPN.
2. Register on the Gurobi website as a student. Set up your work as 'academic user', with 'student' as 'academic position'. You will be asked to verify your email address.




3. Download and install Gurobi optimizer from the download page for your operating system.
4. If you are not on the TU Delft campus, this is when you should check to make sure your VPN connection is activated.
5. Visit the License Request section of your User Portal, find the “Named-User Academic” block and select “GENERATE NOW.” Once accepting the agreement, you should be

immediately able to copy the key, as shown in the image:




6. We will open a command-line interface to set up the key (it doesn't matter which one you use, for example, the system default, Anaconda prompt or git bash; on **Windows** try `Win`+`R` then type `cmd` then press `enter`; on **Mac** open Terminal). For both platforms, all you should have to do is copy the license key, which will paste the following text: `grbgetkey xxxxxxxx`, where you should see your key instead of the `x`'s.
7. Next, you should see a prompt that looks like this:

```
info : grbgetkey version 11.0.0, build v11.0.0rc2
info : Platform is <PLATFORM> - <SPECIFICATIONS>
info : Contacting Gurobi license server...
info : License file for license ID XXXXXX was successfully retrieved
info : License expires at the end of the day on YYYY-MM-DD
info : Saving license file...

In which directory would you like to store the Gurobi license key file?
[hit Enter to store it in LICENSE_PATH/gurobi]:
```

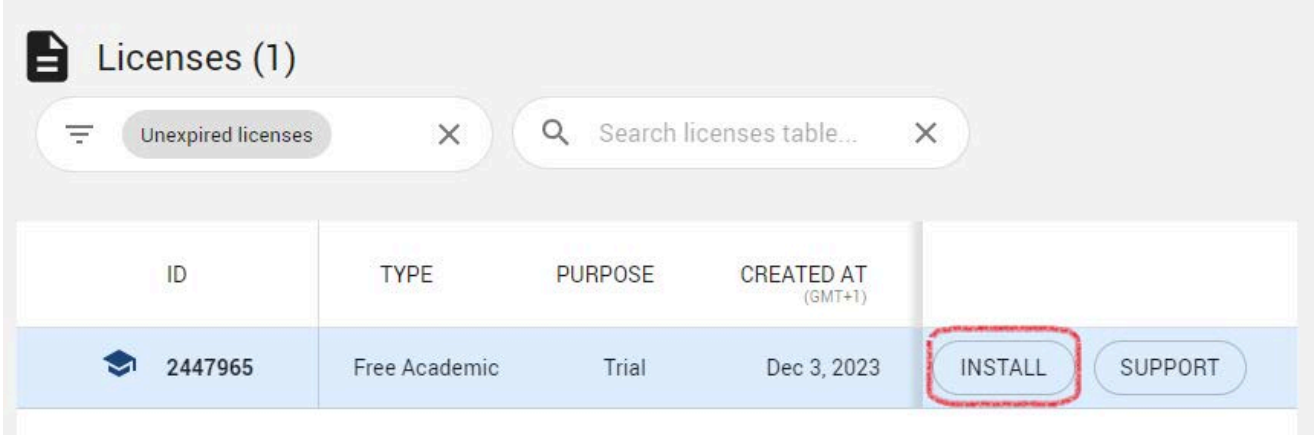
8. You will be prompted to confirm the file location; it is strongly recommended that you accept the default location for your system by hitting `Enter`. If you need to store the license file somewhere other than the default location, please see Where do I place the

Gurobi license file (gurobi.lic)? for important details. If successful, the following message will appear:

```
info : License XXXXXX written to file LICENSE_PATH/gurobi.lic
```

9. To test the installation, execute `gurobi` in your command-line interface and hit `enter`. Depending on your operating system configuration, you may need to try `gurobi.sh` or `gurobi.bat` as well. If this does not work, there are additional instructions here, or you can also get help from an instructor.
10. Now you are ready to go!

|If you need to find your license later, after it was generated, you can find it via the User Portal under "Review your current licenses." Clicking the "Install" button, as shown below, will bring you to the same view as shown above in Step 4 to copy the license key.



# Vectors and matrices

## Contents


- 1.1. Matrix
- 1.2. Vector
- 1.3. Matrix operations
- 1.4. Special matrices
- 1.5. Matrix Transpose
- 1.6. Trace of a matrix
- 1.7. Linear Combination
- 1.8. Linear Dependency
- 1.9. Rank of a matrix
- 1.10. Singular matrices
- 1.11. Inverse of a matrix


## 1.1. Matrix

A matrix is a rectangular array of different numbers. If $A$ is a matrix, it can be written, in a generic form as:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix},$$

where $a_{ij} = (A)_{ij}$ denotes the $(i, j)$th element of $A$. The size of a matrix is defined as its number of rows times the number of columns. In the above example, the size of the matrix $A$ is $m \times n$.

# 1.2. Vector

An $m \times 1$ matrix

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

is called an $m$-dimensional column vector. A $1 \times n$ matrix is called an $n$-dimensional row vector. When we speak of vectors, we usually refer to column vectors, unless otherwise stated.

# 1.3. Matrix operations

*Summation*: The sum of two matrices $A$ and $B$ is defined if they have the same size. Then $A + B = (a_{ij} + b_{ij})$. The product of a scalar $\alpha$ and a matrix $A$ is $\alpha A = A\alpha = (\alpha a_{ij})$.

*Multiplication*: The product $AB$ of two matrices $A$ and $B$ is defined only if the number of columns of matrix $A$ equals the number of rows of matrix $B$. Thus if $A$ is of size $m \times n$ and $B$ is of size $n \times o$, then $C = AB$ is of size $m \times o$, which has its $(ij)$th entry $c_{ij}$ given by $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$. Note that in general $AB \neq BA$.

# 1.4. Special matrices

*Square matrices*: Matrices with the same number of columns and rows are called square matrices. Let $A$ be an $m \times n$ matrix. If $m = n$, then $A$ is a square matrix.

*Symmetric matrices*: A square matrix $A$ is called symmetric when $A^T = A$. For example, the following matrix is a symmetric matrix:

$$A = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 \\ 2 & 3 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{bmatrix}.$$

In other words, in symmetric matrices, $a_{ij} = a_{ji}$.

*Diagonal matrices*: The diagonal entries of an $m \times m$ matrix $A$, with entries $(A)_{ij} = a_{ij}$, are $a_{11}, a_{22}, \ldots, a_{mm}$. Matrix $A$ is called a diagonal matrix if all other entries are equal to zero. Such a matrix will be denoted as $A = \text{diag}(a_{11}, a_{22}, \ldots, a_{mm})$. For example, the following matrix is a diagonal matrix:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}.$$

*Identity matrices*: A diagonal matrix is called an identity matrix if all of its diagonal entries equal $1$. An identity matrix of size $m \times m$ will be written as $A = I_m$ or sometimes simply as $A = I$. Examples of identity metrices are:

$$I_1 = 1, \quad I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

*Zero matrix*: A matrix of which all its entries equal zero is called the zero-matrix and is written as $A = 0$.

# 1.5. Matrix Transpose

The transpose of a matrix $A$ is the matrix $A^T$ obtained by interchanging the rows and columns of $A$. Therefore, the $(i, j)$th entry of $A^T$ is $a_{ji}$. If the size of $A$ is $m \times n$, then its transpose $A^T$ is a matrix of size $n \times m$. Let $\alpha$ and $\beta$ be scalars and $A$ and $B$ be matrices of appropriate size. Then the following properties are hold:

$$\begin{array}{llll} (a) & (\alpha A)^T = \alpha A^T & (c) & (\alpha A + \beta B)^T = \alpha A^T + \beta B^T \\ (b) & (A^T)^T = A & (d) & (AB)^T = B^T A^T \end{array}.$$

Check yourself that for a symmetric matrix $W$ we have $W = W^T$. Consequently, also $\left(\mathrm{A}^T W \mathrm{A}\right)$ and $\left(\mathrm{A}^T W \mathrm{A}\right)^{-1}$ are symmetric.

# 1.6. Trace of a matrix

The trace is a function that is only defined on square matrices. The trace of an square matrix $A$ of size $m \times m$ , denoted as $\text{trace}(A)$, is defined to be the sum of all its diagonal entries, $\text{trace}(A) = \sum_{i=1}^{m} a_{ii}$. For example:

$$\text{trace}\left( \begin{bmatrix} 4 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 1 & 2 \end{bmatrix} \right) = 4 + 3 + 2 = 9.$$

# 1.7. Linear Combination

Assume $x_1,\ x_2, \ldots, x_n$ are $n$ vectors with the size of $m \times 1$ each, and $\alpha_1,\ \alpha_2, \ldots \alpha_n$, are $n$ scalars, then $\sum_{i=1}^{n} \alpha_i x_i$ is called a linear combination of $x_1, \ldots, x_n$. In other words, if we multiply vectors by scalars, and add or subtract them from each other, the result is called the linear combination of the vectors.

# 1.8. Linear Dependency

Vectors $x_i$, $i = 1, \ldots, n$, are said to be linear dependent if there exist scalars $\alpha_i$, not all equal to zero, such that $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0$. An alternative but equivalent definition is: a set of vectors are linear dependent if at least one of the vectors can be written as linear combination of the others. If not, we say that the vectors $x_i$ are linear independent. The vectors $x_1, \ldots, x_n$ are linear independent if and only if:

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n = 0 \Rightarrow \alpha_1 = \ldots = \alpha_n = 0$$

# 1.9. Rank of a matrix

The maximum number of linearly independent column vectors of a matrix $A$ is called the rank of $A$, and it is denoted as $\text{rank}(A)$. The maximum number of linearly independent column vectors of a matrix always equals its maximum number of linearly independent row vectors. A matrix $A$ of size $m \times n$ is said to have full row rank if $\text{rank}(A) = m$ and full column rank if $\text{rank}(A) = n$ . The matrix is said to have a rank deficiency if $\text{rank}(A) < \min(m, n)$.

# 1.10. Singular matrices

Square matrices with a rank deficiency are called singular. Alternatively, if a square matrix of size $m \times m$ has a rank equal to $m$, the matrix is called nonsingular.

# 1.11. Inverse of a matrix

Let $A$ be a square $m \times m$ and nonsingular matrix. Then there exists a unique matrix $A^{-1}$, called the inverse of $A$, such that

$$AA^{-1} = A^{-1}A = I_m.$$

It can be shown that $(A^{-1})^{-1} = A$. If $\alpha$ is a nonzero scalar, then $(\alpha A)^{-1} = \frac{1}{\alpha}A^{-1}$. Note that, singular matrices (or matrices with rank deficiency) are not invertible.

# Vector spaces

## Contents


- 2.1. Vector space and subspace
- 2.2. Span
- 2.3. Basis and dimension of a vector space
- 2.4. Column Space (or Range Space) of a Matrix
- 2.5. Inner product of two vectors
- 2.6. Norm or length of a vector
- 2.7. Distance between two vectors
- 2.8. Angle between two vectors and orthogonality


## 2.1. Vector space and subspace

A set $\mathcal{W}$ is called a vector space if its elements are vectors and the sum of two elements of $\mathcal{W}$ is again an element of $\mathcal{W}$, and the product of an element of $\mathcal{W}$ with a scalar is an element of $\mathcal{W}$.
A subset of a vector space which itself is a vector space is called a subspace.

## 2.2. Span

Let $a_i \in \mathcal{W}$, $i = 1, \ldots, n$, where $\mathcal{W}$ is a vector space. The set of all linear combinations of $a_1, \ldots, a_n$, denoted as $\{a_1, \ldots, a_n\}$, is a subspace of $\mathcal{W}$: $\{a_1, \ldots, a_n\} \subset \mathcal{W}$.

If every vector of a vector space $\mathcal{V}$ can be written as a linear combination of $a_1, \ldots, a_n$, then $a_1, \ldots, a_n$ is said to span $\mathcal{V}$: $\mathcal{V} = \{a_1, \ldots, a_n\}$.

## 2.3. Basis and dimension of a vector space

A basis of a vector space $\mathcal{W}$ is a set of linear independent vectors which span $\mathcal{W}$.

Every vector space contains a basis and every vector can be written as a unique linear combination of the vectors of a basis.

The dimension of a vector space $\mathcal{W}$, denoted as $\dim \mathcal{W}$, is the number of vectors of a basis of $\mathcal{W}$. In an $n$-dimensional vector space $\mathcal{W}$, every linear independent set of $n$ vectors is a basis of $\mathcal{W}$.

As an example, the three-dimensional space $\mathbb{R}^3$ is a vector space, and one possible basis for $\mathbb{R}^3$ is a set of the following unit vectors:

$$\begin{bmatrix}1\\0\\0\end{bmatrix}, \quad \begin{bmatrix}0\\1\\0\end{bmatrix}, \quad \begin{bmatrix}0\\0\\1\end{bmatrix}.$$

That is all the vectors in $\mathbb{R}^3$ can be written as linear combination of these three unit vectors. For example, the arbitrary three dimentional vector $\begin{bmatrix}4\\3\\5\end{bmatrix}$ can be written as the linear combination:

$$\begin{bmatrix}4\\3\\5\end{bmatrix} = 4\begin{bmatrix}1\\0\\0\end{bmatrix} + 3\begin{bmatrix}0\\1\\0\end{bmatrix} + 5\begin{bmatrix}0\\0\\1\end{bmatrix}.$$

We can say the three vectors $\begin{bmatrix}1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\end{bmatrix}$, and $\begin{bmatrix}0\\0\\1\end{bmatrix}$ span the vector space $\mathbb{R}^3$.

# 2.4. Column Space (or Range Space) of a Matrix

The column space (or range space) of a matrix $A$ of size $m \times n$, is the subspace of $\mathbb{R}^m$ which is spanned by the column vectors of $A$. The range space of $A$ is denoted as $\mathcal{R}(A)$. The dimension of $\mathcal{R}(A)$, equals the maximum number of linear independent column vectors of $A$.

For example, let $A = \begin{bmatrix}1 & 1\\1 & 2\\1 & 3\end{bmatrix}$. Then the $\mathcal{R}(A)$ is a subspace in $\mathbb{R}^3$, and its elements can be written as linear combination of the two column vectors of $A$:

$$\begin{bmatrix}1\\1\\1\end{bmatrix}, \text{ and } \begin{bmatrix}1\\2\\3\end{bmatrix}.$$

In this example, the dimension of $\mathcal{R}(A)$ is 2 becouse $A$ has two independent columns.

# 2.5. Inner product of two vectors

Inner product of two vectors $x$ and $y$ denoted by $(x, y)$, is defined as:

$$(x, y) = x^T y.$$

For example the inner product of the two vectors $x = \begin{bmatrix}3\\2\\1\\1\end{bmatrix}$ and $y = \begin{bmatrix}1\\2\\3\\4\end{bmatrix}$ is calculated as:

$$(x, y) = x^T y = \begin{bmatrix}3 & 2 & 1 & 1\end{bmatrix} \begin{bmatrix}1\\2\\3\\4\end{bmatrix} = (3 \times 1) + (2 \times 2) + (1 \times 3) + (1 \times 4) = 14.$$

# 2.6. Norm or length of a vector

The length or norm of a vector $x$, denoted as $||x||$, is defined as:

$$||x|| = \sqrt{(x, x)} = \sqrt{x^T x}.$$

For example, the length of the vector $x = \begin{bmatrix}3\\2\\1\\1\end{bmatrix}$ is computed as:

$$||x|| = \sqrt{(3 \times 3) + (2 \times 2) + (1 \times 1) + (1 \times 1)} = \sqrt{15}$$

# 2.7. Distance between two vectors

The distance between two vectors $x$ and $y$ is defined as the norm of $u - v$, and is given as $||u - v||$.

# 2.8. Angle between two vectors and orthogonality

The angle between two vectors $x$ and $y$

is defined to be the number $\theta \in [0, \pi]$ such that:

$$\cos(\theta) = \frac{(x, y)}{\|x\|\|y\|}.$$

Two vectors are said to be orthogonal (or normal to each other) if the angle between them is $\pi/2$, or in other words when their inner product equals zero: $(x, y) = x^T y = 0$ .

# System of linear equations

## Contents


- 3.1. Properties of linear systems of equations
- 3.2. Consistent systems: at least one solution exists
- 3.3. Unique solution
- 3.4. Determined, overdetermined and underdetermined systems
- 3.5. Redundancy


Vectors and matrices play an important role in describing and solving systems of linear equations. Let a linear system of $m$ equations in $n$ unknown parameters $x_i$, $i = 1, \ldots, n$, be given as

$$\begin{aligned} y_1 &= a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n \\ y_2 &= a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \\ \vdots & \qquad\qquad \vdots \\ y_m &= a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \end{aligned}$$

This system of linear equations can be written in a matrix form as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

By introducing

$$\mathrm{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \ \mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}, \ \mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix},$$

the linear system can be written in the compact matrix form $\mathrm{y} = \mathrm{Ax}$.

Note that the right-hand side of the system $\mathrm{y} = \mathrm{Ax}$ can be re-written as the linear combinations of the column vectors of $\mathrm{A}$:

$$\mathrm{Ax} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} x_1 + \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} x_2 + \cdots + \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} x_n.$$

The linear system of equations is solved, once it is known which linear combination(s) of the columns of $\mathrm{A}$ produces $\mathrm{y}$.

Example: Assume the following system of three equations with two unknowns:

$$\begin{aligned} 1 &= x_1 - x_2 \\ -1 &= x_1 - 3x_2 \\ 3 &= x_1 + x_2 \end{aligned}$$

These three equations can be written in a matrix form as

$$\begin{bmatrix} 1 \\ -1 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 1 & -3 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The solution of this system is $x_1 = 2$ and $x_2 = 1$. We can see that the particular linear combination of columns of $\mathrm{A}$ using $x_1 = 2$ and $x_2 = 1$ produces the vector $\begin{bmatrix} 1 \\ -1 \\ 3 \end{bmatrix}$:

$$\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} 2 + \begin{bmatrix} -1 \\ -3 \\ 1 \end{bmatrix} 1 = \begin{bmatrix} 1 \\ -1 \\ 3 \end{bmatrix}.$$

# 3.1. Properties of linear systems of equations

Consider a linear system of $m$ equations in $n$ unknowns:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

In compact form this is written as $\mathrm{y} = \mathrm{Ax}$.

Such a system may or may not have a solution, and if a solution exists, this solution may or may not be unique.

# 3.2. Consistent systems: at least one solution exists

If a solution exists, the system of equations is called consistent. A solution exists if and only if $y$ can be written as a linear combination of the column vectors of matrix $\mathrm{A}$:

$$\mathrm{y} = \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} x_1 + \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} x_2 + \cdots + \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} x_n$$

In that case $\mathrm{y}$ is an element of the range space of $\mathrm{A}$: $\mathrm{y} \in \mathcal{R}(\mathrm{A})$.

For a consistent system of equations it is always possible to find a solution $x$ such that $\mathrm{y} = \mathrm{Ax}$. If it is not possible to find a solution, the system is called inconsistent.

# Consistency is guaranteed if $\mathrm{rank}(\mathrm{A}) = m$

Explanation: $\mathrm{y} \in \mathbb{R}^m$ and the system is consistent if $\mathrm{y} \in \mathcal{R}(\mathrm{A})$, hence consistency is guaranteed if $\mathcal{R}(\mathrm{A}) = \mathbb{R}^m$. This means that the columns of $\mathrm{A}$ must span the complete space of reals $\mathbb{R}^m$, which is true if $\mathrm{rank}(\mathrm{A}) = m$.

If $\mathrm{rank}(\mathrm{A}) < m$, the system may or may not be consistent, this depends then on the actual entries of the vector $\mathrm{y}$.

# 3.3. Unique solution

A consistent system has a unique solution if and only if the column vectors of matrix $\mathrm{A}$ are independent, i.e. if $\text{rank}(\mathrm{A}) = n$.

This can be seen as follows: assume that $\mathrm{x}$ and $\mathrm{x}' \neq \mathrm{x}$ are two different solutions. Then $\mathrm{Ax} = \mathrm{Ax}'$ or $\mathrm{A}(\mathrm{x} - \mathrm{x}') = 0$. But this can only be the case if some of the column vectors of $\mathrm{A}$ are linear dependent, which contradicts the assumption of full column rank.

# 3.4. Determined, overdetermined and underdetermined systems

A system of equations $\mathrm{y} = \mathrm{Ax}$ with $\text{rank}(\mathrm{A}) = m = n$ is consistent and has a unique solution: $\hat{\mathrm{x}} = \mathrm{A}^{-1}\mathrm{y}$. Such a system is called *determined*.

A system is *underdetermined* if $\text{rank}(\mathrm{A}) < n$, i.e. if it does not have a unique solution.

A system is *overdetermined* if $\text{rank}(\mathrm{A}) < m$, i.e. the system may or may not be consistent.

# 3.5. Redundancy

The redundancy of a system of equations is equal to $m - \text{rank}(\mathrm{A})$.

If we restrict ourselves to systems of observation equations that are of full column rank: $\text{rank}(\mathrm{A}) = n$, the system can either be

*Determined systems*: $\text{rank}(\mathrm{A}) = n = m$, the redundancy is equal to 0

*Overdetermined systems*: $\text{rank}(\mathrm{A}) = n < m$, the redundancy is equal to $m - n > 0$

# Multivariate differentiation

## Contents


- 4.1. Gradient vector
- 4.2. Gradient of linear functions
- 4.3. Gradient of quadratic functions
- 4.4. Jacobian matrix


## 4.1. Gradient vector

For a multivariate function $f(\mathrm{x})$ where $\mathrm{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, the vector with enteries as partial derivatives with respect to $x_1, x_2, \ldots,$ and $x_n$ is called the gradient vector and it is denoted as:

$$\partial_{\mathrm{x}} f = \begin{bmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{bmatrix}.$$

For example, let's assume $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $f(\mathrm{x}) = x_1^2 + 3x_2 + \cos(x_3)$. Then the gradient vector of $f(\mathrm{x})$ is:

$$\partial_{\mathrm{x}} f = \begin{bmatrix} 2x_1 \\ 3 \\ -\sin(x_3) \end{bmatrix}.$$

# 4.2. Gradient of linear functions

If $\mathrm{b}$ and $\mathrm{x}$ are the column vectors with the same size, the gradient of $f(\mathrm{x}) = \mathrm{b}^T\mathrm{x} = \mathrm{x}^T\mathrm{b}$ with respect to $\mathrm{x}$ is derived as:

$$\partial_{\mathrm{x}} f = \mathrm{b}.$$

If $\mathrm{b}$ is a $m \times 1$ vector, $\mathrm{x}$ is a $n \times 1$, and $\mathrm{A}$ is a matrix of size $m \times n$, the gradient of $f(\mathrm{x}) = \mathrm{b}^T\mathrm{A}\mathrm{x} = \mathrm{x}^T\mathrm{A}^T\mathrm{b}$ with respect to $\mathrm{A}$ is derived as:

$$\partial_{\mathrm{x}} f = \mathrm{A}^T\mathrm{b}.$$

# 4.3. Gradient of quadratic functions

If $\mathrm{x}$ is the column vector of size $n \times 1$, and $\mathrm{B}$ is an square matrix of size $n \times n$, the gradient of $f(\mathrm{x}) = \mathrm{x}^T\mathrm{B}\mathrm{x} = \mathrm{x}^T\mathrm{B}^T\mathrm{x}$ with respect to $\mathrm{x}$ is derived as:

$$\partial_{\mathrm{x}} f = (\mathrm{B} + \mathrm{B}^T)\mathrm{x}.$$

Note that if $\mathrm{B}$ is a symmetric matrix, then $\mathrm{B}^T = \mathrm{B}$, and so the gradient of $f(\mathrm{x}) = \mathrm{x}^T\mathrm{B}\mathrm{x} = \mathrm{x}^T\mathrm{B}^T\mathrm{x}$ with respect to $\mathrm{x}$ is derived as:

$$\partial_{\mathrm{x}} f = 2\mathrm{B}\mathrm{x}.$$

# 4.4. Jacobian matrix

Assume $m$ number of different multivariate functions as $f_1(\mathrm{x}), f_2(\mathrm{x}), \ldots, f_m(\mathrm{x})$, where $\mathrm{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$. If we collect all the $f_i(\mathrm{x})$ functions in a column vector $F(\mathrm{x})$ as

$$F(\mathrm{x}) = \begin{bmatrix} f_1(\mathrm{x}) \\ \vdots \\ f_m(\mathrm{x}) \end{bmatrix},$$

$F(\mathrm{x})$ is itself a $m$-vector function of an $n$-vector $\mathrm{x}$. For such a vector multivariate function, the $m \times n$ matrix of partial derivatives $\partial f_i / \partial x_j$, denoted as $J_{\mathrm{x}}$, is called Jacobian matrix of $F$ defined as:

$$J_{\mathrm{x}} = \partial_{\mathrm{x}^T} F = \begin{bmatrix} \partial f_1/\partial x_1 & \dots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \dots & \partial f_2/\partial x_n \\ \vdots & & \vdots \\ \partial f_m/\partial x_1 & \dots & \partial f_m/\partial x_n \end{bmatrix}.$$

For example, let $\mathrm{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $F(\mathrm{x}) = \begin{bmatrix} x_1 + x_2 + x_3 \\ x_1^2 + 2x_2 \\ x_1^2 + x_2^3 + \cos(x_3) \end{bmatrix}$. Then the Jacobian matrix of $F(\mathrm{x})$ is derived as:

$$J_{\mathrm{x}} = \partial_{\mathrm{x}^T} F = \begin{bmatrix} 1 & 1 & 1 \\ 2x_1 & 2 & 0 \\ 2x_1 & 3x_2^2 & -\sin(x_3) \end{bmatrix}.$$

# Taylor series

## Contents


- 5.1. Taylor's theorem for approximating functions of 1 variable
- 5.2. First-order Taylor polynomial for linearizing a function of $n$ variables


## 5.1. Taylor's theorem for approximating functions of 1 variable

Taylor's theorem can be used to approximate a function $f(x)$ with the so called $p$-th order Taylor polynomial:

$$f(x) \approx f(x_0) + \frac{\partial}{\partial x} f(x_0)(x - x_0) + \frac{1}{2!} \frac{\partial^2}{\partial x^2} f(x_0)(x - x_0)^2 + \ldots + \frac{1}{p!} \frac{\partial^p}{\partial x^p} f(x_0)(x -$$

where it is required that the function $f : \mathbb{R} \mapsto \mathbb{R}$ is $p$-times differentiable at the point $x_0 \in \mathbb{R}$.

The approximation error is equal to

$$R_p(x) = f(x) - P_k(x)$$

and is called the remainder term.

Example:

A linear approximation (also called linearization) of $f(x) = \cos(x)$ at $x_0$ is obtained by the 1st order Taylor polynomial as:

$$f(x) \approx \cos x_0 - \sin x_0 \cdot (x - x_0)$$

# 5.2. First-order Taylor polynomial for linearizing a function of $n$ variables

For linearizing non-linear functions of $x$ being a vector with $n$ variables, we need the first-order Taylor polynomial, which is then given by:

$$f(x) \approx f(x_0) + \partial_{x_1} f(x_0) \Delta x_0 + \partial_{x_2} f(x_0) \Delta x_0 + \ldots + \partial_{x_n} f(x_0) \Delta x_0$$

where $\Delta x_0 = (x - x_0)$ and we need the $n$ partial derivatives of function $f$ evaluated at $x_0$.

# Probability axioms

The discussion about what probability is, as well as the interpretation of what it means, has been going on for centuries, and we won't cover it here. Lukcily, most modern perspectives on probability are compatible and can be traced back to a few key fundamental concepts, known as the axioms of probability. These three axioms are summarized here:

1. The probability of any event is $[0, 1]$.
2. The set of all possible outomes has probability 1.0.
3. If two events are mutually exclusive, the probability of both events (the union) is the sum of their probabilities: $P[A \cup B] = P(A) + P(B)$

While they may seem simple, these axioms are actually precise mathematical statements that provide the basis for a number of theorems and proofs which allow us to apply probability theory to a wide range of applications.

# Random variables

From a previous course on probability and statistics you probably remember the use of Venn diagrams to aid our understanding of arithmetic involving probability. The box represented the sample space, with circles (and their relative size) representing various events and their probability of occurrence. The concept of a random variable is nothing more than a mapping of this sample space to a number line, which allows us to combine probability theory with calculus. We use a capital letter to denote a random variable and realizations of that random variable are described with a lower case letter. For example, consider a discrete random variable, $X$, which can take 1, 2 or 3 as possible outcomes. We can write the probability of each event mathematically as:

$$p_X(x_i) = P(X = x_i)$$

Where $i = 1, 2, 3$ and $P$ is mathematical notation for describing the event in the parenthesis. The function describes the probability for all outcomes of $P$ (i.e., the sample space), which, as implied by the axioms, should sum to 1.

This simple example is actually a discrete random variable, because the values of $X$ take on a finite number of values. For most of this course, however, we will work with continuous random variables, which can take on an infinite set of values. A key characteristic of continuous variables is that the probaility of an event must be defined over an interval. For example, we might be interested in the probability that $X$ takes a value between 3 and 4, which we will see in the next sections.

The mapping of a sample space to a number line combined with a (mathematical) specification of probability describes how probability is distributed across all events in the sample space. For this reason we use the term *probability distribution* to describe the mathematical functions defining probability for outcomes of a random variable, regardless of whether it is discrete or continous.

# Expectation (mean) and variance

## Contents


- 8.1. Mean or expectation
- 8.2. Variance


## 8.1. Mean or expectation

The mean is the first moment of a probability distribution and is also referred to as the *expectation* of the random variable $X$, and is defined as:

$$\mathbb{E}(X) = \mu_X = \sum_{i=1}^{N} x_i p_X(x_i)$$

for discrete distributions, where $p_Y(x_i)$ are the probability masses for all possible outcomes $x_i$, $i = 1, \ldots, N$.

For continuous distributions the equivalent is:

$$\mathbb{E}(X) = \mu_X = \int_{-\infty}^{\infty} x f_X(x) dx$$

where $f_X(x)$ is the probability density function.

We will often use the notation $\mathbb{E}(X) = \mu_X$.

The empirical or sample mean based on $m$ outcomes (or: realizations) $x_i$ can be computed as:

$$\hat{\mu}_X = \frac{1}{m}\sum_{i=1}^{m} x_i$$

where we use the ^-symbol to indicate it is an *estimate* of the mean based on a number of realizations.

# 8.2. Variance

The outcomes or realizations of a random variable will by definition inhibit a certain spread; they will fluctuate around the mean. The variance or dispersion of a random variable is a measure of these fluctuations around the mean, and is defined by:

$$\begin{aligned}\mathbb{D}(X) = \sigma_X^2 &= \mathbb{E}((X-\mu_X)^2) \\ &= \int_{-\infty}^{\infty} (x-\mu_X)^2 f_X(x) dx\end{aligned}$$

Hence, the variance equals the expectation of the squared deviations from the mean value. The variance is the *second central moment*.

Based on the formula for the sample mean, we can also find the expression for the sample variance:

$$\hat{\sigma}_X^2 = \frac{1}{m-1}\sum_{i=1}^{m} (x_i - \hat{\mu}_X)^2$$

Note that here we divide by $m-1$ instead of $m$. The reason is that otherwise you would get a sample variance which on average would deviate from the true value. In other words, by dividing by $m-1$ we guarantee that if you would repeatedly determine the sample variance based on a new set of $m$ observations, the average of these sample variances converges to the true variance. For large $m$ this of course does not matter much.

The standard deviation $\sigma_X$ of random variable $X$ is given by the square root of its variance.

# Probability Tables

Sometimes we need to compute a probability, or evaluate the inverse CDF of a distribution, but we don't have a computer handy. Or perhaps it's easier to simply look up a value in a table rather than firing up a Python environment. Tables for useful distributions are provided in the Sections of this Chapter, along with some interactive code examples to help you understand what the table is telling you, as well as illustrate the methods that are implemented in commonly used Python packages like `scipy.stats`.

**MUDE exam information**

You will get one or more of the following tables on the exam, depending on the question. These pages are intended for you to understand how the value in the table is computed, and to have practice reading from them. Keep in mind that sometimes the table is formatted in a slightly different way. *It might even be possible that the table on the exam gives you probability for a different part of the distribtuion (e.g., a different tail) than those illustrated here!*

# Table: Standard Normal Distribution

## Contents


- Table of Values


The table shows one-sided (**right-hand**) probabilities $\alpha$ as function of the critical value $k_\alpha$, i.e. $\alpha = P(Z \geq k_\alpha)$. Note that this table gives a different value than the double-tailed illustration as described in Precision and confidence intervals.

To evalute the table for $k_\alpha$: find the value up to the first decimal is in the first column, then find the column in the first row that most closely matches the second decimal.

Example: $\alpha$ = 0.0250 for $k_\alpha$ = 1.96.

The cell below is set up to use interactively. To use it, click 🚀 –> **Live Code** on the top right corner of this screen and then wait until Python interaction is ready. The method `scipy.stats.norm` has already been imported as `norm`.

```
alpha = 0.0250

k_alpha = norm.ppf(1 - alpha)

print(f"For alpha = {alpha:.4f}, k_alpha = {k_alpha:0.4f}.")
print(f"The probability in the upper (right-hand) tail is {100*alpha:.1f}%.")

print(f"\nIf this k_alpha defined a confidence level, it would be {100*alpha*2:.1f}%,")
print(f" and the bounds would be {k_alpha:.2f} standard deviations from the mean.")
```

```
For alpha = 0.0250, k_alpha = 1.9600.
The probability in the upper (right-hand) tail is 2.5%.

If this k_alpha defined a confidence level, it would be 5.0%,
 and the bounds would be 1.96 standard deviations from the mean.
```

# Table of Values

*Table 9.1* Standard Normal Distribution, Upper-Right Tail

| $k_\alpha$ | **0.00** | **0.01** | **0.02** | **0.03** | **0.04** | **0.05** | **0.06** | **0.07** | **0.08** | **0.** |
|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 0.5000 | 0.4960 | 0.4920 | 0.4880 | 0.4840 | 0.4801 | 0.4761 | 0.4721 | 0.4681 | 0. |
| 0.1 | 0.4602 | 0.4562 | 0.4522 | 0.4483 | 0.4443 | 0.4404 | 0.4364 | 0.4325 | 0.4286 | 0. |
| 0.2 | 0.4207 | 0.4168 | 0.4129 | 0.4090 | 0.4052 | 0.4013 | 0.3974 | 0.3936 | 0.3897 | 0. |
| 0.3 | 0.3821 | 0.3783 | 0.3745 | 0.3707 | 0.3669 | 0.3632 | 0.3594 | 0.3557 | 0.3520 | 0. |
| 0.4 | 0.3446 | 0.3409 | 0.3372 | 0.3336 | 0.3300 | 0.3264 | 0.3228 | 0.3192 | 0.3156 | 0. |
| 0.5 | 0.3085 | 0.3050 | 0.3015 | 0.2981 | 0.2946 | 0.2912 | 0.2877 | 0.2843 | 0.2810 | 0. |
| 0.6 | 0.2743 | 0.2709 | 0.2676 | 0.2643 | 0.2611 | 0.2578 | 0.2546 | 0.2514 | 0.2483 | 0. |
| 0.7 | 0.2420 | 0.2389 | 0.2358 | 0.2327 | 0.2296 | 0.2266 | 0.2236 | 0.2206 | 0.2177 | 0. |
| 0.8 | 0.2119 | 0.2090 | 0.2061 | 0.2033 | 0.2005 | 0.1977 | 0.1949 | 0.1922 | 0.1894 | 0. |
| 0.9 | 0.1841 | 0.1814 | 0.1788 | 0.1762 | 0.1736 | 0.1711 | 0.1685 | 0.1660 | 0.1635 | 0. |
| 1.0 | 0.1587 | 0.1562 | 0.1539 | 0.1515 | 0.1492 | 0.1469 | 0.1446 | 0.1423 | 0.1401 | 0. |
| 1.1 | 0.1357 | 0.1335 | 0.1314 | 0.1292 | 0.1271 | 0.1251 | 0.1230 | 0.1210 | 0.1190 | 0. |
| 1.2 | 0.1151 | 0.1131 | 0.1112 | 0.1093 | 0.1075 | 0.1056 | 0.1038 | 0.1020 | 0.1003 | 0. |
| 1.3 | 0.0968 | 0.0951 | 0.0934 | 0.0918 | 0.0901 | 0.0885 | 0.0869 | 0.0853 | 0.0838 | 0. |
| 1.4 | 0.0808 | 0.0793 | 0.0778 | 0.0764 | 0.0749 | 0.0735 | 0.0721 | 0.0708 | 0.0694 | 0. |
| 1.5 | 0.0668 | 0.0655 | 0.0643 | 0.0630 | 0.0618 | 0.0606 | 0.0594 | 0.0582 | 0.0571 | 0. |
| 1.6 | 0.0548 | 0.0537 | 0.0526 | 0.0516 | 0.0505 | 0.0495 | 0.0485 | 0.0475 | 0.0465 | 0. |
| 1.7 | 0.0446 | 0.0436 | 0.0427 | 0.0418 | 0.0409 | 0.0401 | 0.0392 | 0.0384 | 0.0375 | 0. |
| 1.8 | 0.0359 | 0.0351 | 0.0344 | 0.0336 | 0.0329 | 0.0322 | 0.0314 | 0.0307 | 0.0301 | 0. |
| 1.9 | 0.0287 | 0.0281 | 0.0274 | 0.0268 | 0.0262 | 0.0256 | 0.0250 | 0.0244 | 0.0239 | 0. |

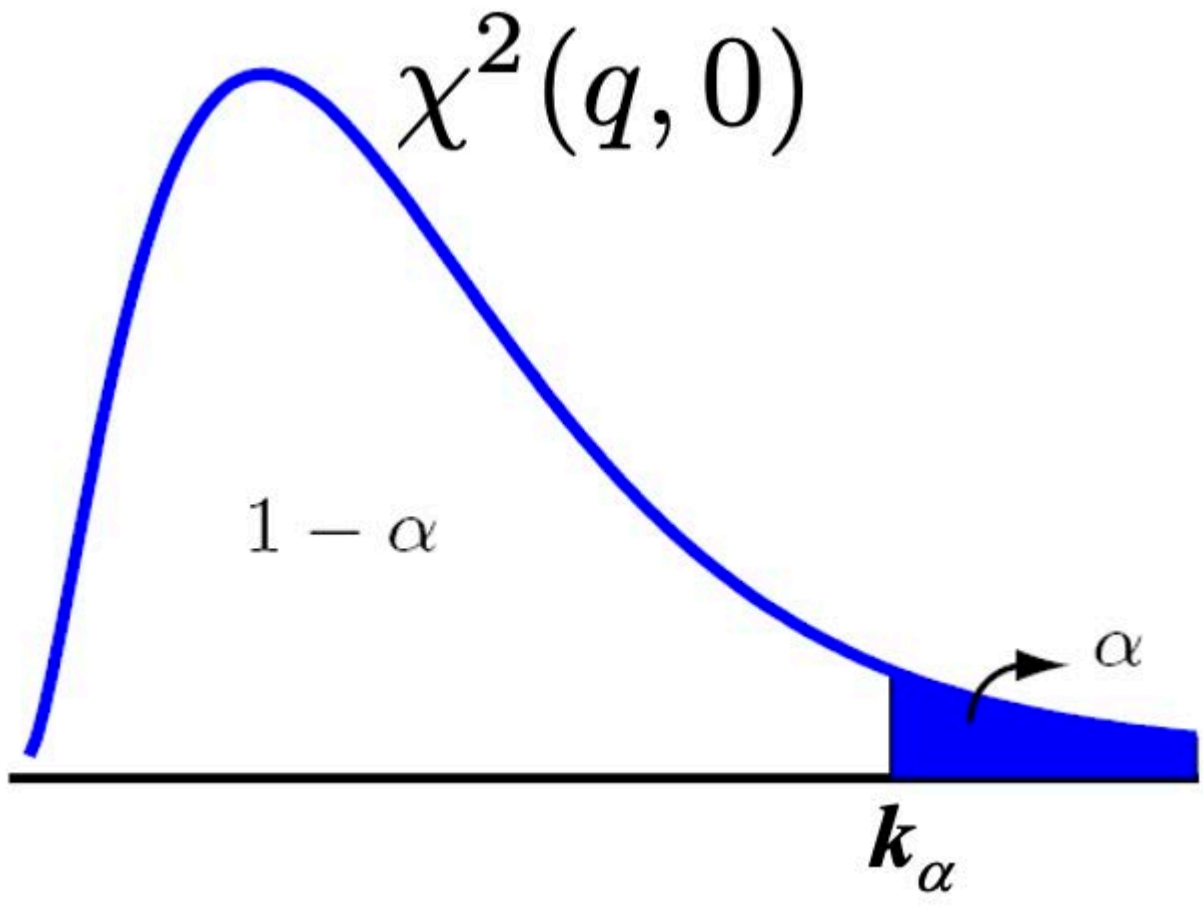
| $k_\alpha$ | **0.00** | **0.01** | **0.02** | **0.03** | **0.04** | **0.05** | **0.06** | **0.07** | **0.08** | **0.** |
|---|---|---|---|---|---|---|---|---|---|---|
| 2.0 | 0.0228 | 0.0222 | 0.0217 | 0.0212 | 0.0207 | 0.0202 | 0.0197 | 0.0192 | 0.0188 | 0. |
| 2.1 | 0.0179 | 0.0174 | 0.0170 | 0.0166 | 0.0162 | 0.0158 | 0.0154 | 0.0150 | 0.0146 | 0. |
| 2.2 | 0.0139 | 0.0136 | 0.0132 | 0.0129 | 0.0125 | 0.0122 | 0.0119 | 0.0116 | 0.0113 | 0. |
| 2.3 | 0.0107 | 0.0104 | 0.0102 | 0.0099 | 0.0096 | 0.0094 | 0.0091 | 0.0089 | 0.0087 | 0. |
| 2.4 | 0.0082 | 0.0080 | 0.0078 | 0.0075 | 0.0073 | 0.0071 | 0.0069 | 0.0068 | 0.0066 | 0. |
| 2.5 | 0.0062 | 0.0060 | 0.0059 | 0.0057 | 0.0055 | 0.0054 | 0.0052 | 0.0051 | 0.0049 | 0. |
| 2.6 | 0.0047 | 0.0045 | 0.0044 | 0.0043 | 0.0041 | 0.0040 | 0.0039 | 0.0038 | 0.0037 | 0. |
| 2.7 | 0.0035 | 0.0034 | 0.0033 | 0.0032 | 0.0031 | 0.0030 | 0.0029 | 0.0028 | 0.0027 | 0. |
| 2.8 | 0.0026 | 0.0025 | 0.0024 | 0.0023 | 0.0023 | 0.0022 | 0.0021 | 0.0021 | 0.0020 | 0. |
| 2.9 | 0.0019 | 0.0018 | 0.0018 | 0.0017 | 0.0016 | 0.0016 | 0.0015 | 0.0015 | 0.0014 | 0. |
| 3.0 | 0.0013 | 0.0013 | 0.0013 | 0.0012 | 0.0012 | 0.0011 | 0.0011 | 0.0011 | 0.0010 | 0. |
| 3.1 | 0.0010 | 0.0009 | 0.0009 | 0.0009 | 0.0008 | 0.0008 | 0.0008 | 0.0008 | 0.0007 | 0. |
| 3.2 | 0.0007 | 0.0007 | 0.0006 | 0.0006 | 0.0006 | 0.0006 | 0.0006 | 0.0005 | 0.0005 | 0. |
| 3.3 | 0.0005 | 0.0005 | 0.0005 | 0.0004 | 0.0004 | 0.0004 | 0.0004 | 0.0004 | 0.0004 | 0. |
| 3.4 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0.0003 | 0. |

# Table: Central Chi-Squared Distribution

## Contents


- Table of Values


The table shows one-sided (**right-hand**) probabilities $\alpha$ as function of the critical value $k_\alpha$ and the degrees of freedom, $q$, i.e. $\alpha = P(X \geq k_\alpha)$ for $\chi^2(q, 0)$.




To evaluate the table for $k_\alpha$: find the value of $q$ in the first column, then the column for $\alpha$ in the first row.

Example: $\alpha$ = 0.0100 and $q$ = 10 yield $k_\alpha$ = 23.2093.

The cell below is set up to use interactively. To use it, click 🚀 –> **Live Code** on the top right corner of this screen and then wait until Python interaction is ready. The method `scipy.stats.chi2` has already been imported as `chi2`.

```
alpha = 0.0100
q = 10

k_alpha = chi2.ppf(1 - alpha, q)

print(f"For alpha = {alpha:.4f} and q = {q} degrees of freedom, k_alpha = {k_alpha:0.4f},
print(f"The probability in the upper (right-hand) tail is {100*alpha:.1f}%.")
```

```
For alpha = 0.0100 and q = 10 degrees of freedom, k_alpha = 23.2093,
The probability in the upper (right-hand) tail is 1.0%.
```

# Table of Values

*Table 9.2* Central Chi-Squared Distribution

| $\alpha$ | **0.9990** | **0.9950** | **0.9900** | **0.9750** | **0.9500** | **0.9000** | **0.1000** | **0.0500** |
|---|---|---|---|---|---|---|---|---|
| $q$ | | | | | | | | |
| 1 | 0.0000 | 0.0000 | 0.0002 | 0.0010 | 0.0039 | 0.0158 | 2.7055 | 3.8415 |
| 2 | 0.0020 | 0.0100 | 0.0201 | 0.0506 | 0.1026 | 0.2107 | 4.6052 | 5.9915 |
| 3 | 0.0243 | 0.0717 | 0.1148 | 0.2158 | 0.3518 | 0.5844 | 6.2514 | 7.8147 |
| 4 | 0.0908 | 0.2070 | 0.2971 | 0.4844 | 0.7107 | 1.0636 | 7.7794 | 9.4877 |
| 5 | 0.2102 | 0.4117 | 0.5543 | 0.8312 | 1.1455 | 1.6103 | 9.2364 | 11.0705 |
| 6 | 0.3811 | 0.6757 | 0.8721 | 1.2373 | 1.6354 | 2.2041 | 10.6446 | 12.5916 |
| 7 | 0.5985 | 0.9893 | 1.2390 | 1.6899 | 2.1673 | 2.8331 | 12.0170 | 14.0671 |
| 8 | 0.8571 | 1.3444 | 1.6465 | 2.1797 | 2.7326 | 3.4895 | 13.3616 | 15.5073 |
| 9 | 1.1519 | 1.7349 | 2.0879 | 2.7004 | 3.3251 | 4.1682 | 14.6837 | 16.9190 |
| 10 | 1.4787 | 2.1559 | 2.5582 | 3.2470 | 3.9403 | 4.8652 | 15.9872 | 18.3070 |
| 11 | 1.8339 | 2.6032 | 3.0535 | 3.8157 | 4.5748 | 5.5778 | 17.2750 | 19.6751 |
| 12 | 2.2142 | 3.0738 | 3.5706 | 4.4038 | 5.2260 | 6.3038 | 18.5493 | 21.0261 |
| 13 | 2.6172 | 3.5650 | 4.1069 | 5.0088 | 5.8919 | 7.0415 | 19.8119 | 22.3620 |
| 14 | 3.0407 | 4.0747 | 4.6604 | 5.6287 | 6.5706 | 7.7895 | 21.0641 | 23.6848 |
| 15 | 3.4827 | 4.6009 | 5.2293 | 6.2621 | 7.2609 | 8.5468 | 22.3071 | 24.9958 |
| 16 | 3.9416 | 5.1422 | 5.8122 | 6.9077 | 7.9616 | 9.3122 | 23.5418 | 26.2962 |
| 17 | 4.4161 | 5.6972 | 6.4078 | 7.5642 | 8.6718 | 10.0852 | 24.7690 | 27.5871 |
| 18 | 4.9048 | 6.2648 | 7.0149 | 8.2307 | 9.3905 | 10.8649 | 25.9894 | 28.8693 |
| 19 | 5.4068 | 6.8440 | 7.6327 | 8.9065 | 10.1170 | 11.6509 | 27.2036 | 30.1435 |

| $\alpha$ | **0.9990** | **0.9950** | **0.9900** | **0.9750** | **0.9500** | **0.9000** | **0.1000** | **0.0500** |
|---|---|---|---|---|---|---|---|---|
| $q$ | | | | | | | | |
| 20 | 5.9210 | 7.4338 | 8.2604 | 9.5908 | 10.8508 | 12.4426 | 28.4120 | 31.4104 |
| 21 | 6.4467 | 8.0337 | 8.8972 | 10.2829 | 11.5913 | 13.2396 | 29.6151 | 32.6706 |
| 22 | 6.9830 | 8.6427 | 9.5425 | 10.9823 | 12.3380 | 14.0415 | 30.8133 | 33.9244 |
| 23 | 7.5292 | 9.2604 | 10.1957 | 11.6886 | 13.0905 | 14.8480 | 32.0069 | 35.1725 |
| 24 | 8.0849 | 9.8862 | 10.8564 | 12.4012 | 13.8484 | 15.6587 | 33.1962 | 36.4150 |
| 25 | 8.6493 | 10.5197 | 11.5240 | 13.1197 | 14.6114 | 16.4734 | 34.3816 | 37.6525 |
| 26 | 9.2221 | 11.1602 | 12.1981 | 13.8439 | 15.3792 | 17.2919 | 35.5632 | 38.8851 |
| 27 | 9.8028 | 11.8076 | 12.8785 | 14.5734 | 16.1514 | 18.1139 | 36.7412 | 40.1133 |
| 28 | 10.3909 | 12.4613 | 13.5647 | 15.3079 | 16.9279 | 18.9392 | 37.9159 | 41.3371 |
| 29 | 10.9861 | 13.1211 | 14.2565 | 16.0471 | 17.7084 | 19.7677 | 39.0875 | 42.5570 |
| 30 | 11.5880 | 13.7867 | 14.9535 | 16.7908 | 18.4927 | 20.5992 | 40.2560 | 43.7730 |
| 40 | 17.9164 | 20.7065 | 22.1643 | 24.4330 | 26.5093 | 29.0505 | 51.8051 | 55.7585 |
| 50 | 24.6739 | 27.9907 | 29.7067 | 32.3574 | 34.7643 | 37.6886 | 63.1671 | 67.5048 |
| 60 | 31.7383 | 35.5345 | 37.4849 | 40.4817 | 43.1880 | 46.4589 | 74.3970 | 79.0819 |
| 70 | 39.0364 | 43.2752 | 45.4417 | 48.7576 | 51.7393 | 55.3289 | 85.5270 | 90.5312 |
| 80 | 46.5199 | 51.1719 | 53.5401 | 57.1532 | 60.3915 | 64.2778 | 96.5782 | 101.8795 |
| 90 | 54.1552 | 59.1963 | 61.7541 | 65.6466 | 69.1260 | 73.2911 | 107.5650 | 113.1453 |
| 100 | 61.9179 | 67.3276 | 70.0649 | 74.2219 | 77.9295 | 82.3581 | 118.4980 | 124.3421 |

# Probability in Python

## Contents


- 10.1. Key Concepts: Probability Theory
- 10.2. Key Concepts: `scipy.stats`
- 10.3. Interactive Examples


This page gives a crash course in probability calculations in Python using continuous parametric distributions of `scipy.stats`.

> **Tip**
>
> Activate Python on this page using the rocket icon (🚀) above. Below the explanatory text there are a series of code cells that illustrate key aspects of `scipy.stats`. Make sure you experiment with the code cells, don't just run and read them. For example, try making the following changes:
>
> 1. Change parameter values
> 2. Change distribution types
> 3. Check the type of all objects
> 4. See if you can find alternative ways of writing code that generates the same output

## 10.1. Key Concepts: Probability Theory

This book is primarily concerned with the use of continuous parametric distributions to make computations related to probability. A *distribution* can thus be defined completely with the PDF or the CDF of a univariate random variable $X$

$$f_X(x;\theta_1,\theta_2) \qquad F_X(x;\theta_1,\theta_2)$$

where $\theta_1$ and $\theta_2$ denote (arbitrary) parameters of the distribution, which describe the shape of the PDF and CDF, respectively. Note the following about the distribution parameters::

1. In theory there is no limit on the number of parameters to define a distribution, although for practical purposes we consider mostly 2-parameter distributions.
2. The parameters of a distribution are usually formulated in a specific way to emphasize a particular physical interpretation; for example, the moments of the random variable or location, scale and shape of the distribution.
3. A common parameterization is location, scale and shape. This allows standardization of the distribution, for example, the location parameter translates the distribution, whereas the scale parameter stretches or compresses it.
4. For the Gaussian distribution, the parameters are typically the mean $\mu$ and $\sigma$, which are the moments of $X$ (first and second). These are also the location and scale parameters of a distribution.
5. For non-Gaussian distributions, there is no guarantee that the location, scale and shape parameters are equivalent to the moments or any other set of parameters used to define the distribution.

> **Tip**
>
> Always check that the implementation of a distribution in the software package you are using to make sure it is consistent with your expectations. This means checking the PDF and CDF as well as the definition of the parameters.

Once a distribution is defined, the list of calculations we make is relatively limited and includes the following:

1. Evaluate the CDF to obtain a probability $p = F_X(x)$
2. Use the inverse CDF to find the random variable value with a specific probability of non-exceedance $x = F_X^{-1}(p)$
3. Evaluate the PDF, which gives *density* (not probability!) as the output of $f_X(x)$
4. Find parameters of a distribution using method of moments or MLE that best fit a given set of data
5. Convert between different parameterizations, for example, find the location and scale parameters from the moments of a random variable
6. Generate a random sample of $n$ realizations of random variable $X$ that follows its distributions $\{x_1, x_2, \ldots, x_n\}$
7. Perform qualitative and quantitative goodness of fit analyses to validate a selected distribution

Refer to other chapters of this book if any of the topics above are not clear.

## Multivariate Distributions

Although the examples above are for univariate cases, the multivariate case is analogous, with the following differences:

1. More than one random variable is considered, meaning it can be more challenging to integrate the *density* function over a specific *regions of interest* $\Omega$
2. Additional parameters are required to define the dependence between marginal random variables. The most simple model is a multivariate Gaussian distribution, which adds one parameter for every additional random variable beyond the second, the Pearson correlation coefficient $\rho_{i,j}$

# 10.2. Key Concepts: `scipy.stats`

Commonly used distributions are implemented in `scipy.stats`, a module in the Python package Scipy. The module uses Object Oriented Programming to elegantly create a large number of distributions that behave in a predictable and numerically efficient way. This is accomplished via the *parent class* `rv_continuous`, which defines a number of key *methods*, for example, `pdf`, `cdf`, `ppf`, `rvs` and `fit`, just to name a few. A specific distribution is created by simply defining the PDF of and *instantiating* it as a child class of `rv_continuous`. *Inheritance* ensures that the methods implemented in `rv_continuous` are available in *instances* of the child class (our distribution!). Why is this important to understand?

1. All distributions in `scipy.stats` can be used in exactly the same way
2. It will help you interpret the Python Traceback when you encounter errors
3. You will better be able to prevent and identify logical errors (e.g., distribution parameters specified in the wrong order)
4. It will help you understand how to search for information in the `scipy.stats` documentation

**Tip**

The documentation of `scipy.stats` can be intimidating, but this is primarily because it assumes the reader has a thorough knowledge of Python, probability theory *and* Object Oriented Programming. One strategy to help you find the information you need is to check two places: 1) the page of the specific distribution you are interested in (for example, Gumbel here), and 2) the page describing the *parent class* `rv_continuous` here.

To easily find these pages, try typing a few key words into a search engine; for example, Googling “scipy stats gumbel” returns the pages linked here as the top search results.

Due to the OOP implentation, the conventional usage of a `scipy.stats` distribution is

```
<distribution_name>.<method>(<argument_1, argument_2, ...>)
```

where `distribution_name` is both the name of the distribution as well as the Python object itself (typically an instance of the distribution class) and `method` is a method of the class, which is typically associated with a calculation to make using the distribution. This will be illustrated below. For now, note the *methods* in the table below that are common to *any* `scipy.stats` distribution.

| Name | Math | `scipy.stats` Method | Conventional Usage | Returns |
|---|---|---|---|---|
| PDF | $f_X(x)$ | `pdf` | `.pdf(x)` | density |
| CDF | $F_X(x)$ | `cdf` | `.cdf(x)` | probability |
| Inverse CDF | $F_X^{-1}(x)$ | `ppf` | `.ppf(x)` | random variable value |
| Generate Sample of $X$ | $\{x_1, x_2, \ldots, x_n\}$ | `rvs` | `.rvs(size=n)` | sample (Numpy array) |
| Fit with MLE | $\{shape, loc, scale\}$ | `fit` | `.fit(data)` | tuple of parameters |
| Fit with Method of Moments | $\{loc, scale\}$ | `fit_loc_scale` | `.fit_loc_scale(data)` | tuple of parameters |

For **mutivariate** distributions, methods in the table above generally work exactly the same, accounting for differences in number of parameters and dimensionality of the random variables. This is illustrated in the examples below.

**Tip**

Python packages like `scipy.stats` rely heavily on Object Oriented Programming (OOP) to implement probability and statistics methods, and only a brief hint is provided here. Refer to the OOP page of this book to brush up on this topic in case anything here is unclear, for example, terms like *instance*, *parent class* or *inheritance.*

# Standard Parameterization

A distribution can be defined without specifying any parameters due to all distributions being implmented in `scipy.stats` in a standardized_ form. This means the *default* values for the location and scale parameters are 0 and 1, respectively. Default values for shape

parameters, when used, depend on the distribution parameterization. This means, for example, that the following two lines of code produce equivalent results:

```
stats.gumbel_r().cdf(1)
stats.gumbel_r(loc=0, scale=1).cdf(1)
```

Not all distributions have a shape parameter; if this is the case, the arguments are simply location and scale. When a shape parameter is required, it is usually the first parameter provided in the return of a function or required as an input argument. This often leads to logical errors when a location parameter is provided first, but the method expects a shape parameter.

**Tip**

To avoid memorizing the specific order of input arguments for every distribution, or having to constantly check the documentation, it is a good idea to specify the keyword argument of the parameter to guarantee that it is interpreted correctly (else an error will occur).

For example, do this:

```
stats.gumbel_r(loc=xxxx, scale=yyyy)
```

don't do this:

```
stats.gumbel_r(xxxx, yyyy)
```

Note also that the following two lines of code produce equivalent results:

```
stats.gumbel_r(lscale=1, loc=0).cdf(1)
stats.gumbel_r(loc=0, scale=1).cdf(1)
```

# Two Modes of Usage

There are two common ways to use `scipy.stats` distributions:

1. Evaluating a method of the distribution object directly
2. Defining a distribution as a variable, creating a *frozen* distribution

## Using a Method Directly

This simultaneously specifies the distribution (i.e., the distribution type and its parameters) and evaluates one of the methods. For example, consider the following line of code:

```
stats.gumbel_r(loc=1, scale=2).cdf(1)
```

You should be able to identify each of the following aspects; try reading through the list out loud and identifying each piece of the code that corresponds to the item:

1. Using the `stats` module of the `scipy` package…
2. define the Gumbel right distribution…
3. with parameter values 1 and 2 for the location and scale, respectively…
4. and evaluate the CDF at a random variable value of 1.

## Frozen Distributions

The approach of evaluating a method directly can be efficient, but eventually becomes cumbersome if the distribution is used repeatedly, as the parameters need to be specified each time. In this case, one can create a *frozen* distribution. This creates and instance of the distribution class where the parameters fixed at the values specified. Consider the following lines of code:

```
my_dist = stats.gumbel_r(loc=1, scale=2)
my_dist.cdf(1)
```

This returns the same value as the example above, except the parameters to not need to be specified again when evaluating the CDF as they are already incorporated in `my_dist`. Checking the `type` of this object would indicate that it is a frozen distribution:

```
type(my_dist) ---> scipy.stats._distn_infrastructure.rv_continuous_frozen
```

**Tip**

Frozen distributions are an example of a `scipy.stats` feature that is difficult to find in the documentation. It's actually buried in the `rv_continuous` page. See if you can find the explanation here!

# 10.3. Interactive Examples

The concepts described above will now be put into practice.

```
import scipy.stats as stats
import numpy as np
import matplotlib.pyplot as plt
```

## General Usage

First let's create a toy data set to work with, which we will do by generating a random sample from the Gumbel right distribution. Note the following steps executed in the cell below:

1. Using the `stats` module of the `scipy` package…
2. Define the Gumbel right distribution…
3. Using default parameters…
4. Create a random sample of 1000 values…
5. and save it as an array `data`

```
data = stats.gumbel_r.rvs(size=1000)
print(f'Variable data is of type {type(data)} and has length {len(data)}.')
```

```
Variable data is of type <class 'numpy.ndarray'> and has length 1000
```

Note that you **must** specify the keywords argument, `size=n`, *not* this: `stats.gumbel_r.rvs(5)`. Try it in a cell to see why.

The cell above follows the "use the methods directly" approach. We can also define the distribution as a frozen distribution:

```
frozen_dist = stats.gumbel_r()
print(f'Variable frozen_dist is of type {type(frozen_dist)}.')
```

```
Variable frozen_dist is of type <class 'scipy.stats._distn_infrastructure.rv_continuous_f
```

We can use methods of the Numpy array class (and object `data`) to confirm that the moments of the data match the moments of the distribution from which it came.

```
```

```
print('The mean and standard deviation of the...')
print(f'sample (data): {np.mean(data):6.3f} and {np.std(data):6.3f}.')
print(f'distribution:  {frozen_dist.mean():6.3f} and {frozen_dist.std():6.3f}.')
```

```
The mean and standard deviation of the...
sample (data):  0.570 and  1.277.
distribution:   0.577 and  1.283.
```

As expected, the values are nearly identical; the small differences would decrease as the sample size goes towards infinity.

Let's check what the parameters of the distribution would be if we fit the Gumbel right distribution to the data.

```
parameters = stats.gumbel_r.fit(data)
print(f'Parameters are type {type(parameters)}')
print(f'Parameter values are {parameters[0]:6.3f} and {parameters[1]:6.3f}.')
```

```
Parameters are type <class 'tuple'>
Parameter values are -0.004 and  0.981.
```

The values, which are provided in a tuple, represent the location and scale of the distribution and are close to 0 and 1. This is logical because we did not specify any parameters when creating the sample in the first place, thus the *default* values are 0 and 1, respectively.

We can specify alternative parameters and confirm that the location and scale have changed:

```
data_2 = stats.gumbel_r(loc=1, scale=2).rvs(size=1000)
parameters_2 = stats.gumbel_r.fit(data_2)
print(f'Parameters are type {type(parameters_2)}')
print(f'Parameter values are {parameters_2[0]:6.3f} and {parameters_2[1]:6.3f}.')
```

```
Parameters are type <class 'tuple'>
Parameter values are  0.955 and  2.016.
```

That was easy! Note that there is a handy Python feature called **tuple unpacking** which can be used to easily enter the parameters of a distribution as input arguments. Given a tuple `my_tuple`, the operator `*` "unpacks" all of the values from the tuple individually. For example, the code

```
my_tuple = (1, 'two', 4)
*my_tuple
```

returns `1 two 4`. This can be very useful when working with distributions, as we can do the following (illustrated for the CDF):

```
my_parameters = stats.<dist>.fit(data)
stats.<dist>.cdf(x, *my_parameters)
```

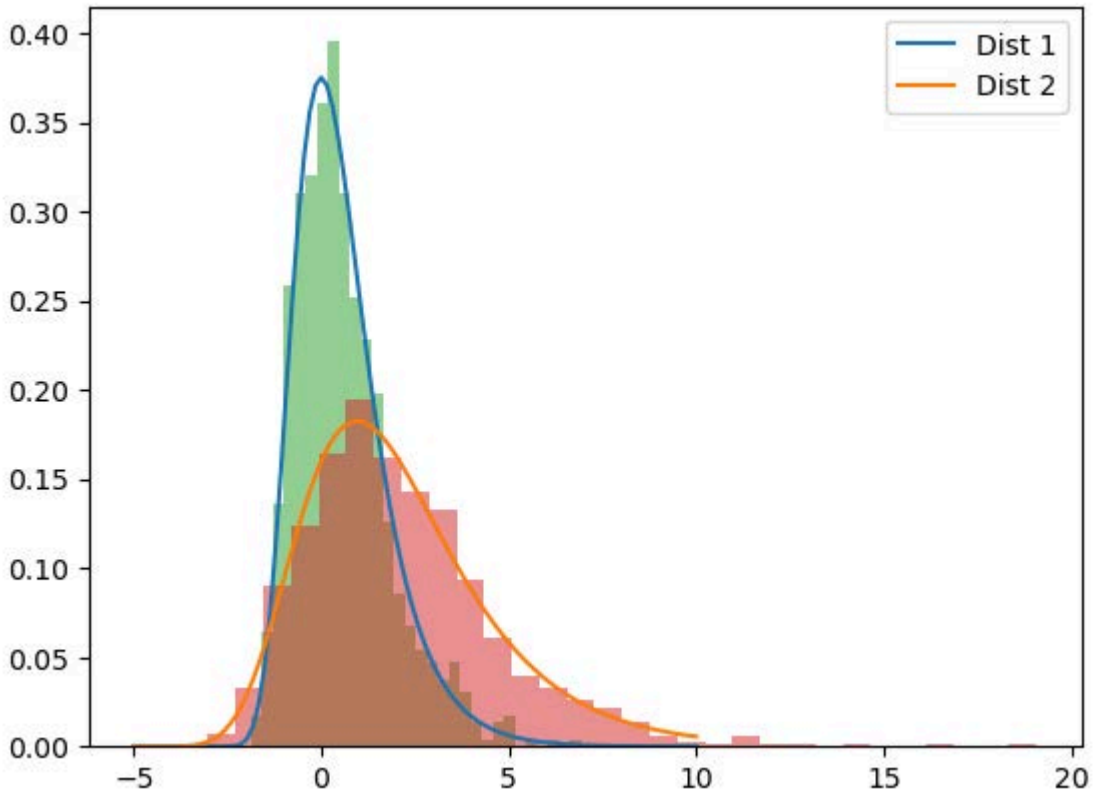
which evaluates the CDF at random variable values `x` given the parameters that fit the `data` (using MLE by default) for a generic distribution `<dist>`. Try it out in the following cell:

```
my_tuple = (1, 'two', 4)
print(*my_tuple)

stats.gumbel_r.cdf(2, *parameters_2)
print(f'The CDF at 2 is {stats.gumbel_r.cdf(2, *parameters_2):6.3f}.')
```

```
1 two 4
The CDF at 2 is  0.551.
```

Now that we have two distributions, let's plot them, along with the data, using tuple unpacking to easily specify the parameters:

```
x = np.linspace(-5, 10, 100)
pdf_fitted = stats.gumbel_r.pdf(x, *parameters)
pdf_fitted_2 = stats.gumbel_r.pdf(x, *parameters_2)
plt.plot(x, pdf_fitted, label='Dist 1')
plt.plot(x, pdf_fitted_2, label='Dist 2')
plt.hist(data, density=True, bins=30, alpha=0.5)
plt.hist(data_2, density=True, bins=30, alpha=0.5)
plt.legend()
plt.show()
```




## Illustration of a Common Error

It is easy to make mistakes when using `scipy.stats`. One common mistake is using the result of a `fit` method as if it were a distribution object. For example, defining the variable below and using the (non-existent) `cdf` method.

```
not_a_distribution = stats.gumbel_r.fit(data)
not_a_distribution.cdf()
```

If you cannot tell why this code is problematic, copy it into a cell and execute it to see what goes wrong! Hint: check `type(not_a_distribution)`.

# Comparing Math and Code

This section explicitly illustrates common mathematical expressions and the equivalent operations using Python.

## Univariate Case

Consider a random variable $X$ with moments (mean and standard deviation) $\mathbb{E}(X) = 5$ and $\mathbb{V}\text{ar}(X) = 2$ and we would like to fit a Gumbel right distribution using the method of moments. We can do this analytically as follows:

$$\mathbb{V}ar(X) = \frac{\pi^2}{6}\beta^2 \quad \rightarrow \quad \beta = \sqrt{\frac{6\mathbb{V}ar(X)}{\pi^2}} = \sqrt{\frac{6 \cdot 4}{\pi^2}} = 4.100$$

$$\mathbb{E}(X) = \mu + \lambda\beta \quad \rightarrow \quad \mu = \mathbb{E}(X) - \lambda\beta = 5 - 0.5772 \cdot 4.100 = 1.559$$

```
var = 4
exp = 5
beta = np.sqrt(6*var/np.pi**2)
mu = exp - 0.5772*beta
print(f'Calculated mu and beta are {mu:6.3f} and {beta:6.3f}.')
```

```
Calculated mu and beta are  4.100 and  1.559.
```

By comparing the PDF $f_X(x)$ and CDF $F_X(x)$ in the book with the documentation of `scipy.stats`, we can see that the function and parameter definitions are consistent. Furthermore, by examining the PDF and CDF we can confirm that the parameters $\mu$ and $\beta$ are the location and scale parameters, respectively. Thus we can define the distribution as follows. Printing the values of the mean and standard deviation of the distribution confirm that we have correctly defined the function!

```
dist = stats.gumbel_r(loc=mu, scale=beta)
print(f'The mean and standard deviation of this distribution are {dist.mean():6.3f} and {
```

```
The mean and standard deviation of this distribution are  5.000 and  2.000.
```

There is an alternative way to (approximately) define the distribution based on the method of moments, even if you have neither data nor the analytic expression for the parameters (or perhaps you are lazy and don't want to solve the equations by hand, as done above). This approach involves the following steps:

1. Create an arbitrary sample of points with the desired mean and standard deviation (this is most easily done with `stats.norm.rvs()`)
2. Use the `fit_loc_scale` method to find the location and scale parameters of any distribution
3. Freeze the distribution with desired parameter values
4. Print the mean and standard deviation of the frozen distribution to verify that the distribution was defined with the correct moments

```
```

```
sample =  stats.norm.rvs(loc=5, scale=2, size=1000)
parameters = stats.gumbel_r.fit_loc_scale(sample)
dist = stats.gumbel_r(*parameters)
print(f'Parameters are type {type(parameters)}')
print(f'Parameter values are {parameters[0]:6.3f} and {parameters[1]:6.3f}.')
print(f'The moments sample are {np.mean(sample):6.3f} and {np.std(sample):6.3f}.')
print(f'The moments of the distribution are {dist.mean():6.3f} and {dist.std():6.3f}.')
```

```
Parameters are type <class 'tuple'>
Parameter values are  4.063 and  1.555.
The mean and standard deviation of the sample are  4.961 and  1.995.
The mean and standard deviation of the distribution are  4.961 and  1.995.
```

When using this approach, note that the accuracy of the fitted parameters is limited by the accuracy of the artificially created sample. For more precise results, one should use the analytic expressions for the parameters. Also, don't forget to use `fit_loc_scale` to enforce that the method of moments is used, instead of simply `fit`, which uses the MLE approach!

Next we will compute a probability of the previously found distribution, beginning with:

$$p_1 = P[X \leq 3] = F_X(3)$$

followed by the complement

$$p_2 = P[X > 3] = 1 - P[X \leq 3] = 1 - F_X(3)$$

We can also use the inverse CDF to find the random variable value $x$ that has probability of non-exceedance equal to $p_1$

$$x = F_X^{-1}(p_1) = 3$$

which is the same value $x$ that has probability of exceedance $p_2$

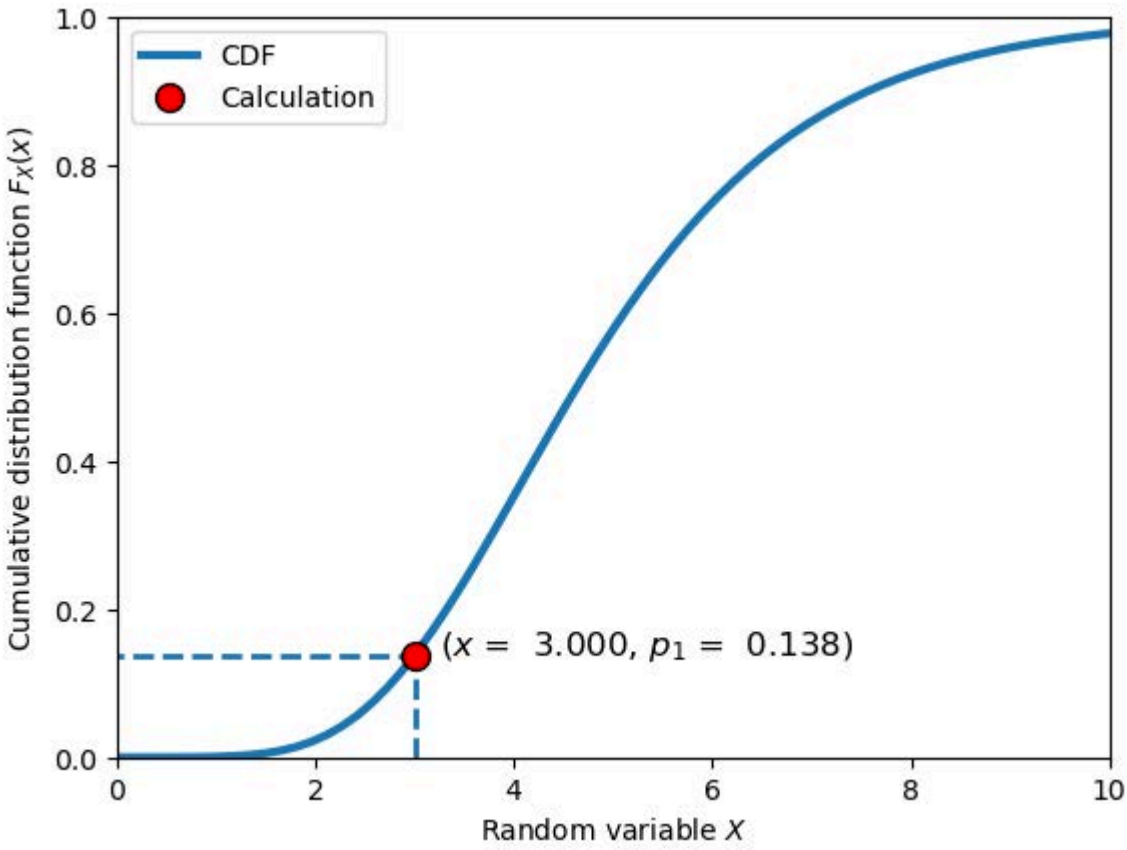
$$x = F_X^{-1}(1 - p_2) = 3$$

```
p1 = dist.cdf(3)
p2 = 1 - dist.cdf(3)
x_of_p1 = dist.ppf(p1)
x_of_p2 = dist.ppf(1 - p2)
print(f'The probability of being less than 3 is {p1:6.3f}.')
print(f'The probability of being greater than 3 is {p2:6.3f}.')
print(f'The x value for p1 is {x_of_p1:6.3f}.')
print(f'The x value for p2 is {x_of_p2:6.3f}.')
```

```
The probability of being less than 3 is  0.138.
The probability of being greater than 3 is  0.862.
The x value for p1 is  3.000.
The x value for p2 is  3.000.
```

Let's illustrate what this looks like by plotting the calculated values and the CDF:

```
# create a plot of the cdf with the value calculated above
x = np.linspace(-5, 10, 100)
F_x = dist.cdf(x)
plt.plot(x, F_x, label='CDF', linewidth=3)
plt.vlines(x_of_p1, 0, p1, linestyles='--', linewidth=2)
plt.hlines(p1, -5, x_of_p1, linestyles='--', linewidth=2)
plt.plot(x_of_p1, p1, 'ro', label='Calculation',
         markeredgecolor='black', markersize=10)
plt.xlim([0, 10])
plt.ylim([0, 1])
plt.legend()
plt.xlabel('Random variable $X$')
plt.ylabel('Cumulative distribution function $F_X(x)$')
plt.text(x_of_p1+.25, p1,
         f'($x$ = {x_of_p1:6.3f}, $p_1$ = {p1:6.3f})', fontsize=12,)
plt.show()
```




# Multivariate Gaussian

Now a quick example with the Multivariate Gaussian distribution. In this example, we wish to define the distribution $f_{X_1,X_2}(x_1, x_2)$, for which we need to specify the *marginal*

distributions and the *correlation coefficient* (or *covariance*). As the marginal distributions of the Multivariate Gaussian are also Gaussian, this is most easily done with the mean vector and covariance matrix (described elsewhere in the book).

Let's consider the situation where:

$$X_1 \sim N(\mu = 8, \sigma = 2)$$
$$X_2 \sim N(\mu = 5, \sigma = 1)$$

with correlation

$$\rho_{X_1,X_2} = 0.5$$

```
mean_vector = [8, 5]

std_x1 = 2
std_x2 = 1
correlation_coefficient = 0.5
covariance_matrix = [[std_x1**2, correlation_coefficient*std_x1*std_x2],
                     [correlation_coefficient*std_x1*std_x2, std_x2**2]]

bivariate_gaussian = stats.multivariate_normal(mean=mean_vector, cov=covariance_matrix)

print(f'Mean vector: {mean_vector}')
print(f'Covariance matrix: {covariance_matrix}')
```

```
Mean vector: [8, 5]
Covariance matrix: [[4, 1.0], [1.0, 1]]
```

Now we can use the distribution to create a sample, then check that it matches our expectation of the moments of marginal distributions. Note that *when sampling from a multivariate distribution we sample from the multivariate object, not the marginal distributions separately!*

```
sample = bivariate_gaussian.rvs(size=1000)
print(f'The sample has shape {sample.shape}.')
print(f'The sample mean of X1 is {np.mean(sample[:,0]):6.3f}.')
print(f'The sample mean of X2 is {np.mean(sample[:,1]):6.3f}.')
```

```
The sample has shape (1000, 2).
The sample mean of X1 is  7.958.
The sample mean of X2 is  4.969.
```

Computing a probability can be accomplished with the multivariate CDF:

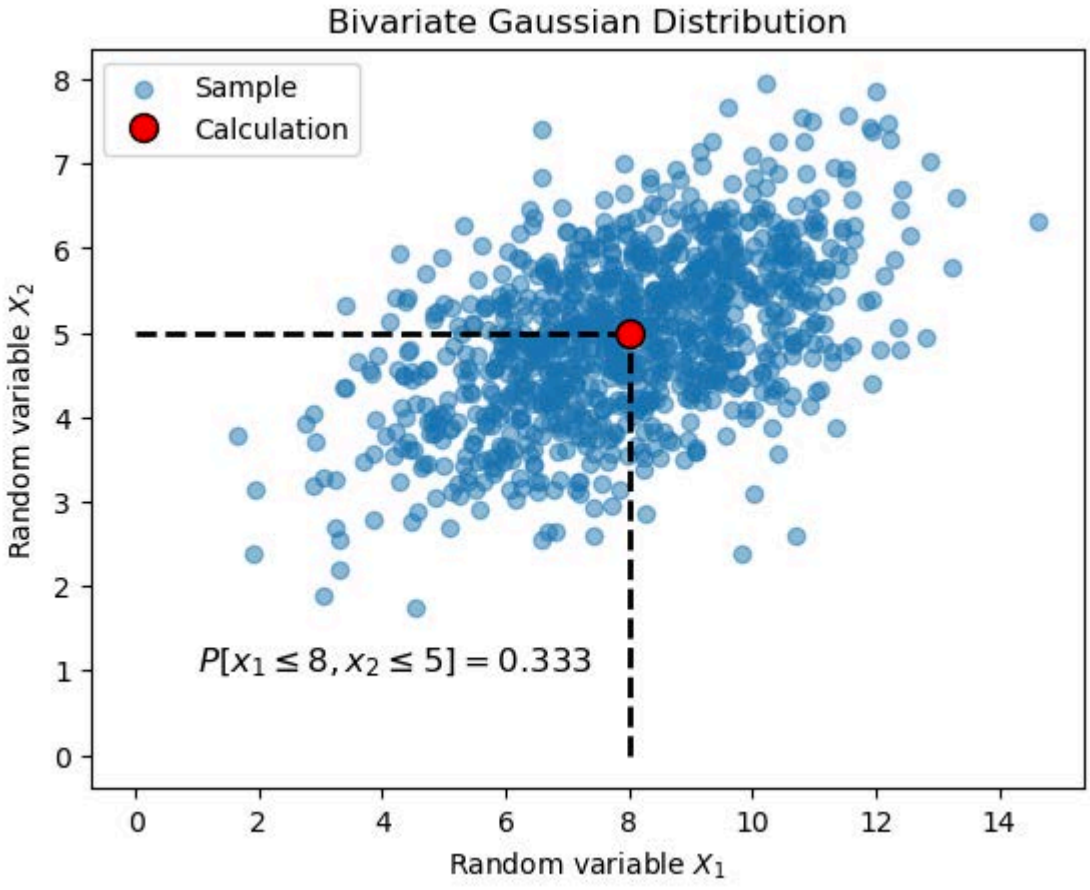
$$p = P[X_1 \leq x_1, X_2 \leq x_2] = F_{X_1,X_2}(x_1, x_2)$$

The following cell computes the non-exceedance probability of the mean values of $X_1$ and $X_2$ and creates a plot to illustrate the computation (with the random samples included to illustrate the density):

$$p = P[X_1 \leq 8, X_2 \leq 5] = F_{X_1,X_2}(8, 5)$$

```
p = bivariate_gaussian.cdf([8, 5])
print(f'The probability of being less than 8 and 5 is {p:6.3f}.')

# create a bivariate plot with the samples and this value
plt.scatter(sample[:,0], sample[:,1], alpha=0.5, label='Sample')
plt.hlines(5, 0, 8, linestyles='--', linewidth=2, color='black')
plt.vlines(8, 0, 5, linestyles='--', linewidth=2, color='black')
plt.plot(8, 5, 'ro', label='Calculation',
         markeredgecolor='black', markersize=10)
plt.xlabel('Random variable $X_1$')
plt.ylabel('Random variable $X_2$')
plt.legend()
plt.title('Bivariate Gaussian Distribution')
plt.text(1, 1,
         f'$P[x_1 \\leq 8, x_2\\leq 5]=${p:5.3f}', fontsize=12,)
plt.show()
```

```
The probability of being less than 8 and 5 is  0.333.
```



The probability computed by the CDF is the integration of the PDF over the lower left region of the random variable space, bounded by the dashed lines.

Recall that for the bivariate case the CDF is the double integral of the bivariate PDF:

$$F_{X_1,X_2}(x_1, x_2) = \int_{-\infty}^{x_2} \int_{-\infty}^{x_1} f_{X_1,X_2}(x_1, x_2)\ \mathrm{d}x_1 \mathrm{d}x_2$$

Integrating over the domain of one variable produces the marginal distribution of the other:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{x_1} f_{X_1,X_2}(x_1, x_2)\ \mathrm{d}x_1 \mathrm{d}x_2 = F_{X_1}(x_1)$$

We can easily do this in Python as follows, confirming our result with the understanding that for the marginal distribution, integrated the PDF below the mean (or evaluating the CDF at the mean) should return a value of 0.5.

```
p = bivariate_gaussian.cdf([8, 1000000])
print(f'The probability of X1 being less than 8 is {p:6.3f}.')
```

```
The probability of X1 being less than 8 is  0.500.
```

# Multivariate Non-Gaussian

Here we use a Copula to describe dependence between $X_1$ and $X_2$. This is an example of a non-Gaussian multivariate distribution (bivariate, in this case). We will use the Clayton Copula to illustrate, as this can be simply defined in a Python function without any extra statistics or probability packages (in fact, it technically doesn't even require Numpy!).

*In this example we only illustrate the computation of the CDF, and not with an object oriented approach, as we don't want to import special packages in the book (we will do that in the in-class workshops).*

As with the Multivariate Gaussian case, we wish to define the distribution $f_{X_1,X_2}(x_1, x_2)$, for which we can specify the same *marginal* distributions:

$$X_1 \sim N(\mu = 8, \sigma = 2)$$
$$X_2 \sim N(\mu = 5, \sigma = 1)$$

The bivariate distribution when using a generic Copula, $C$, is defined by the marginal CDF's $F_{X_1}(x_1)$ and $F_{X_2}(x_2)$:

$$F_{X_1,X_2}(x_1, x_2) = C[F_{X_1}(x_1), F_{X_2}(x_2)]$$

The Clayton Copula is defined as follows, replacing the CDF's with $u$ and $v$, respectively, for simplicity:

$$C[F_{X_1}(x_1), F_{X_2}(x_2)] = C[u, v] = \left[\max\{u^{-\theta} + v^{-\theta} - 1;\ 0\}\right]^{-1/\theta}$$

In this case, instead of the Pearson correlation coefficient (for multivariate Gaussian), dependence is parameterized by $\theta$. Although the values are not equivalent, $\theta$ behaves in a similar fashion to Pearson's $\rho$. Let's try out this bivariate distribution using:

$$\theta = 0.5$$

First we will define the Clayton Copula and evaluate it for the case of $F_{X_1}(x_1) = u = 0.5$ and $F_{X_2}(x_2) = v = 0.5$:

```
def C(F1, F2, theta):
    """
    COmpute bivariate CDF of Clayton copula for X1 and X2.

    Parameters:
    F1: CDF of the first marginal.
    F2: CDF of the second marginal.
    theta (float): Copula parameter.

    Returns: probability.
    """

    return (max(F1**(-theta) + F2**(-theta) - 1, 0))**(-1/theta)

p = C(0.5, 0.5, 0.5)
print(f'The probability calculated using the Clayton copula is {p:.4f}')
```

```
The probability calculated using the Clayton copula is 0.2991
```

Note that this value is not far off from the value of 0.333 computed with the bivariate Gaussian example above with $\rho = 0.5$. As the marginal distributions are Gaussian, evaluating the Copula at values of 0.5 is equivalent to evaluating the random variable values at their median (and mean and mode for the Gaussian distribution).

We can define `scipy.stats` marginal distributions to create a function for the the complete multivariate distribution such that it can be evalutaed as the CDF:

```
def cdf(x):
    X1 = stats.norm(loc=5, scale=1)
    X2 = stats.norm(loc=8, scale=2)
    theta = 0.5
    return C(X1.cdf(x[0]), X2.cdf(x[1]), theta)

p = cdf([5, 8])
print(f'The probability calculated using the Clayton copula is {p:.4f}')
```

```
The probability calculated using the Clayton copula is 0.2991
```

> **Attribution**
>
> This chapter was written by Patricia Mares Nasarre and Robert Lanzafame. Find out more here.

# Additional topics

Next to the topics shown above, a useful resource is the Math Explained website with videos on various math subjects (Calculus, Linear Algebra, Differential Equations, Probability and Statistics) developed by TU Delft.

# References


Agrawal, T. (2021). Introduction to hyperparameters. *Hyperparameter Optimization in Machine Learning: Make Your Machine Learning and Deep Learning Models More Efficient* (pp. 1–30). Berkeley, CA: Apress.

Apostolakis, G.E. (2004). How useful is quantitative risk assessment? *Risk Analysis*, pp. 515–520.

Baecher, G. B. (1982). Statistical methods in site characterization. *Engineering Foundations: Updating Subsurface Samplings of Soils and Rocks and their In-Situ Testing, Santa Barbara*, pp. 463–492.

Baecher, G. B. (2021). IFCEE 2021: Karl Terzaghi Lecture: Greg Baecher: Geotechnical Systems, Uncertainty, and Risk.

Baecher, G. B., & Christian, J. T. (2003). *Reliability and statistics in geotechnical engineering*. John Wiley & Sons.

Bea, R.G. (1998). Human and organization factors: engineering operating safety into offshore structures. *Reliability Engineering & System Safety*, pp. 109–126.

Benjamin, J.R., & Cornell, C.A. (1970). *Probability, Statistics and Decision for Civil Engineers*. McGraw Hill.

BFG (2020). Animation of the unit circle with angles.

Bottelberghs, P.H. (2000). Risk analysis and safety policy developments in the netherlands. *Journal of Hazardous Materials*, *71*, 117-123.

Cardona, O. D., Van Aalst, M. K., Birkmann, J., Fordham, M., Mc Gregor, G., Rosa, P., … others. (2012). Determinants of risk: exposure and vulnerability. *Managing the risks of extreme events and disasters to advance climate change adaptation: special report of the intergovernmental panel on climate change* (pp. 65–108). Cambridge University Press.

Daneshian, M. (2025). Linear programming simplex algorithm. [Retrieved Fall 2023]. URL: thegreatmd4/linear_programming

Deltaprogramma (2014). Deltaprogramma veiligheid. synthesedocument veiligheid.

Eijgenraam, C.J.J. (2006). Optimal safety standards for dike ring areas. *CPB discussion paper*, *62*.

EN 1990:2002 E, E. (2001). Eurocode - basis of structural design.

Faber, M.H., & Stewart, M.G. (2003). Risk assessment for civil engineering facilities: critical overview and discussion. *Reliability Engineering & System Safety*, *80*(2), 173-184. URL: https://www.sciencedirect.com/science/article/pii/S0951832003000279, doi:https://doi.org/10.1016/S0951-8320(03)00027-9

Farmer, F.R. (1967 , April). Siting criteria - a new approach. *Proceedings of IAEA Symposium on the Containment and Siting of Nuclear Power Plants* (pp. 303–324). Vienna: International Atomic Energy Agency (IAEA).

Gelder, P.H.A.J.M. V., Vrijling, J.K., & Slijkhuis, K.A.H. (1997). Coping with uncertainty in the economical optimization of a dike design. *Water for a Changing Global Community*, pp. 554–559.

Jeuken, A., Kind, J., Slootjes, N., Gauderis, J., & Vos, R. (2013 , 01). Cost-benefit analysis of flood risk management strategies for the rhine-meuse delta. *Proceedings of ICFM5At: Rotterdam, Netherlands, Taylor & Francis Group, London, ISBN 978-0-415-62144-1* (p. ).

Jongejan, R.B. (2008). *How safe is safe enough? The government's response to industrial and flood risks* (Doctoral dissertation). Delft University of Technology, the Netherlands.

Jongejan, R.B., Jonkman, S.N., & Vrijling, J.K. (2005). Methods for the valuation of human life. *ESREL*.

Jonkman, B. (2024). Risk evaluation. In Robert Lanzafame (Ed.), Risk and Reliability for Engineers. [Accessed 12-2024]. doi:10.59490/tb.89

Jonkman, B., & Lanzafame, R. (2024). Risk analysis. In Robert Lanzafame (Ed.), Risk and Reliability for Engineers. [Accessed 12-2024]. doi:10.59490/tb.89

Jonkman, S.N., van Gelder, P.H.A.J.M., & Vrijling, J.K. (2003). An overview of quantitative risk measures for loss of life and economic damage. *Journal of Hazardous Materials*, *A99*, 1-30.

Kendall, H.W., Hubbard, R.B., Minor, G.C., & Bryan, W.M. (Union o. C. Scientists). (1977). *The Risks of Nuclear Power Reactors, A Review of the NRC Reactor Safety Study WASH-1400 (NUREG-75/014)*. Cambridge.

Lanzafame, R., Verhagen, S., Mendoza Lugo, M., & Farahat, A. (n.d.). Learn python - sympy. [Accessed 24-03-2025].

Lanzafame, R., Verhagen, S., Mendoza Lugo, M., & Farahat, A. (2024). Python for engineers. [Accessed 12-2024]. URL: https://teachbooks.io/learn-python

Lanzafame, R. (2024). Probabilistic design. In Robert Lanzafame (Ed.), Risk and Reliability for Engineers. [Accessed 12-2024]. doi:10.59490/tb.89

Lanzafame, R. (2024). Risk and reliability for engineers. doi:10.59490/tb.89

Lanzafame, R., & van Woudenberg, T. (2024). Learn programming for engineers. [Accessed 12-2024]. URL: https://teachbooks.io/learn-programming/credits.html

Lanzafame, R., van Woudenberg, T., & Verhagen, S. (2024). Modelling, Uncertainty and Data for Engineers (MUDE) Textbook. Delft University of Technology. https://mude.citg.tudelft.nl/book/2024 CC BY 4.0. doi:10.5281/zenodo.16236358

Microsoft (n.d.). Visual studio live share. [Accessed 11-09-2024].

Microsoft (n.d.). What is live share? [Accessed 11-09-2024].
Moore, J. (2023). Learn multibody dynamics, sympy.
Nogal, M., & Nogal, A. (2021). Sensitivity method for extreme-based engineering problems. *Reliability Engineering & System Safety*, *216*, 107997. URL: https://www.sciencedirect.com/science/article/pii/S095183202100507X, doi:https://doi.org/10.1016/j.ress.2021.107997
Raiffa, H., & Schlaifer, R. (1961). *Applied Statistical Decision Theory*. Cambridge, Massachusetts, United States: Harvard University press.
Slovic, P. (1987). Perception of risk. *Science*, pp. 280–285.
Starr, C. (1967). Social benefit versus technological risk. *Science*, pp. 1232–1238.
Strang, G., & Herman, E. "Jed" (2016). Calculus volume 2. [Publication date: Mar 30, 2016]. URL: https://openstax.org/books/calculus-volume-2/pages/1-introduction
SWOV (2012). Waardering van immateriële kosten van verkeersdoden.
Taerwe, L. (1986). Mastering global quality by an interactive concept. *Proceedings IABSE Symposium Tokyo 1986: Safety and Quality Assurance of Civil Engineering Structures* (pp. 317–325).
TAW, (Technical A. C. o. W. Defences) (1985). Some considerations of an acceptable level of risk in the netherlands.
Tengs, T.O., Adams, M.E., Pliskin, J.S., Gelb Safran, D., Siegel, J.E., Weinstein, M.C., & Graham, J.D. (1995). Five-hundred live saving interventions and their cost effectiveness. *Risk Analysis*, *15*(3), 369-390.
Teunissen, P. J.G. (2024). Adjustment thoery: an introduction. doi:10.59490/tb.95
Teunissen, P. J.G. (2024). Testing thoery: an introduction. doi:10.59490/tb.96
van Dantzig, D. (1956). Economic decision problems for flood prevention. *Econometrica*, *24*.
Vlek, C.A.J. (1996). A multi-level, multi-stage and multi-attribute perspective on risk assessment, decision-making and risk control. *Risk Decision and Policy*, pp. 9–31.
Voortman, H.G., & Vrijling, J.K. (2004). Optimal design of flood defence systems in a changing climate. *Heron*, pp. 75–94.
Vrijling, J.K. (2001). Probabilistic design of water defense systems in the netherlands. *Reliability Engineering & System Safety*, *74*(3), 337-344. URL: https://www.sciencedirect.com/science/article/pii/S0951832001000825, doi:https://doi.org/10.1016/S0951-8320(01)00082-5
Vrijling, J.K., & Gelder, P.H.A.J.M. v. (2000). An analysis of the valuation of a human life. *Foresight and Precaution: Proceedings of ESREL 2000, SARS and SRA-Europe Annual Conference. Edited by M.P. Cottam, D.W. Harvey, R.P. Pape and J. Tait* (pp. 197–200). Edinburgh, Scotland.
Vrijling, J.K., Hengel, W. v., & Houben, R.J. (1995). A framework for risk evaluation. *Journal of Hazardous Materials*, *43*, 245-261.
Vrijling, J.K., Hengel, W. v., & Houben, R.J. (1998). Acceptable risk as a basis for design. *Reliability Engineering and System Safety*, *59*, 141-150.
Vrijling, J.K., & van Beurden, J.A. (1990). Sealevel rise: a probabilistic design problem. *Proceedings of the International Conference on Coastal Engineering*.
Vrouwenvelder, A.C.W.M., & Vrijling, J.K. (1996). *Collegedictaat Probabilistisch ontwerpen b3*. TU Delft.
Workgroup 32, C. (2001). A framework for risk management and risk communications.
CSIRO (n.d.). Page "historical sea level changes; last decades" of website "sea level rise - understanding the past - improving projections for the future". [Accessed 10-2024]. URL: https://www.cmar.csiro.au/sealevel/sl_hist_last_decades.html
CUR (1997). Kansen in de civiele techniek, deel 1: probabilistisch ontwerpen in theorie.
IPCC (2018). Global warming of 1.5°c. an ipcc special report on the impacts of global warming of 1.5°c above pre-industrial levels and related global greenhouse gas emission pathways, in the context of strengthening the global response to the threat of climate change, sustainable development, and efforts to eradicate poverty. [Masson-Delmotte, V., P. Zhai, H.-O. Pörtner, D. Roberts, J. Skea, P.R. Shukla, A. Pirani, W. Moufouma-Okia, C. Péan, R. Pidcock, S. Connors, J.B.R. Matthews, Y. Chen, X. Zhou, M.I. Gomis, E. Lonnoy, T. Maycock, M. Tignor, and T. Waterfield (eds.)]. Accessed in 2024 at https://www.researchgate.net/publication/330090901_Sustainable_development_poverty_eradication_and_reducing_inequalities_In_Glo
SymPy Development Team (2023). Sympy documentation.
Veiligheid Nederland in Kaart. (2014). De veiligheid van nederland in kaart. *Rijkswaterstaat*. URL: https://www.helpdeskwater.nl/publish/pages/131663/eindrapport_vnk.pdf

# Credits and License

## Contents


- How the book is made
- Acknowledgements
- License
- Individual Chapters and Pages
- External Resources
- Contact


You can refer to this book in its entirety as:

> Lanzafame, R., van Woudenberg, T., Verhagen, S. (2024), Modelling, Uncertainty and Data for Engineers (MUDE) Textbook, Delft University of Technology. https://mude.citg.tudelft.nl/book/2024, CC BY 4.0. doi:10.5281/zenodo.16236358.

**BibTeX Citation**

If you would like to refer to the MUDE Textbook with BibTeX (e.g., in a TeachBook or Jupyter Book), the following entry can be used in a `bib` file to get close to the reference shown above. Use the GitHub repository for this book as a reference for setting up APA citations (if desired).

```
@misc{mude2024,
  title={Modelling, {U}ncertainty and {D}ata for {E}ngineers ({MUDE}) {T}extbook
  author={Lanzafame, Robert and van Woudenberg, Tom and Verhagen, Sandra},
  year={2024},
  note={Delft University of Technology. {https://mude.citg.tudelft.nl/book/2024}
  doi={10.5281/zenodo.16236358},
}
```

The text citation is Lanzafame *et al.* (2024)and the parenthetical citation is (Lanzafame *et al.*, 2024).

The introduction, structure of the book and formatting of contents is done under direction of the Editors (Robert Lanzafame, Tom van Woudenberg and Sandra Verhagen), in collaboration with a large team of co-authors and student assistants. Some chapters and pages have additional primary authors who are identified within the book either at the bottom of the first page in a chapter, or at the bottom of an individual page, as necessary. If an author is not listed on a particular chapter or page, the editors may be attributed as the authors.

You can refer to individual chapters or pages within this book as:

> `<Primary Authors>` (2024) `<Title of Chapter or Page>`. In Lanzafame et al. (Eds.), *Modelling, Uncertainty and Data for Engineers (MUDE) Textbook.* Delft University of Technology. https://mude.citg.tudelft.nl/book.

As contents of this book may change each academic year, we cannot guarantee that chapter titles and URL's will remain static indefinitely. Therefore, if it is important for you to reference a specific location within the book, we recommend including the complete URL and date of access in your reference. You can refer to individual chapters or pages within this book as:

> `<Primary Authors>` (2024) `<Title of Chapter or Page>`. In Lanzafame et al. (Eds.), *Modelling, Uncertainty and Data for Engineers (MUDE) Textbook.* Delft University of Technology. https://mude.citg.tudelft.nl/book (chapter `./book/intro/`, retrieved [`<date>`]).

Better yet, include a link to the specific commit! *If you know what this means, we assume you know how to do it.*

# How the book is made

This book is created using open source tools: it is a Jupyter Book that uses a number of features from TeachBooks and is written using Markdown, Jupyter notebooks and Python files to generate some figures. The source files are stored on a public GitHub repository github.com/TUDelft-MUDE/book. Zenodo is used to archive all open versions of the book (beginning with the 2024-25 academic year) and to provide a DOI (10.5281/zenodo.16236358). View the repository README file or contact the editors for additional and up-to-date information.

# Acknowledgements

This book has many contributors, many of whom are also key members of the MUDE Team, as well as critical feedback from MUDE students. The sections below list the primary authors and contributors for each chapter, buit is unfortunately not possible to list all of the small contributions from various people from within and outside Delft University of Technology, not list all contributions in detail.

A better way to see the contributions is to check the Contributors Page of the GitHub repository.

A big “thank you” is also due to the Educational Management Team of the Civil Engineering and Geosciences Faculty at Delft University of Technology for giving the MUDE Team financial and organizational support during the early years of MUDE (especially 2022-2024), in particular Hans Welleman, Director of Education of the faculty. Without the freedom and support to experiment with new tools, this book (and TeachBooks as well!) would not exist!

And in the end, none of this would have happened if it weren’t for the quick meeting between Robert Lanzafame, Caspar Jungbacker and Thirza Feenstra in the Fall of 2022 when we considered making our first Jupyter Book and decided to “go for it!” Caspar Jungbacker set up the first book later that year, which is why the book worm in the TeachBooks logo is named “Caspar.”

# License

This manual is CC BY 4.0 licensed allowing you to share and adapt the material, as long as the source is named. Resources that are *not* included under the CC BY license and external resources that are reused in this book are listed in the sections below.

If an author is not listed on a particular page, it is by the Editors (for example, the introduction page of some chapters).

Auxiliary files such as figures, code, videos, etc, are included under the license of this book and should be attributed to the authors of the chapter or page where they are used, unless otherwise stated below or in the file itself. Text-based files (i.e., non-binary) are typically stored in the repository, within the subdirectory where the source file of the chapter or page is located. Binary files are stored in an FTP server (`https://files.mude.citg.tudelft.nl/<binary_file_name>`). Videos and quiz questions are stored in and served from YouTube and H5p; contact the MUDE Team directly if you are interested in source materials for these resources.

# External Resources

Parts of this book are taken from other external resources and reused in various ways (some of which are *not* shared with a permissive license). Entire chapters or pages are listed individually in the External Resources section below. Resources that are used *within* a page and/or are modified by MUDE authors are listed individually in the Individual Chapters and Pages section below.

# Resources *not* under CC BY

CC BY conditions are *not* applicable to some resources included in this book which resources cannot be reused without explicit permission from the original copyright holder. In some cases, external resources are provided under their own permissive license (e.g., CC BY), in which case permission and instructions for use are already explicitly provided by the copyright holder; however this is not always the case. All resources that are *not* included in the CC BY license of this book are listed individually in the sections below, either: within the summary of each chapter or page, or as entire chapters or pages in External Resources.

# Individual Chapters and Pages

Credits are provided here for chapters and pages that are released under the license of this book (internal resources). Use the guidance provided above to properly share, reuse and cite relevant chapters, pages or any other resources from this book.

## Chapter: Modelling Concepts

> Modelling concepts is written by Alessandro Cabboi, Patricia Mares Nasarre and Robert Lanzafame.
>
> Special thanks goes to João Moura Pereira de Lucas Teixeira, who created first draft of pages from powerpoint slides.

## Chapter: Uncertainty Propagation

> Propagation of Uncertainty is written by Sandra Verhagen.

Special thanks goes to:

- Robert Lanzafame, Patricia Mares Nasarre and Max Ramgraber, who reviewed, commented and/or modified content. Robert and Patricia wrote the page Uncertainty Classification.
- Sophie Keemink, Caspar Jungbacker and Thirza Feenstra, who provided feedback and helped develop exercises.
- Antonio Magherini, who created first draft of pages from powerpoint slides.

Fig. 2.4 and Fig. 2.6 are created by Max Ramgraber (maxramgraber.com/interactive), which are published with a CC BY license and included in this book without modification.

# Chapter: Observation Theory

Observation theory is written by Sandra Verhagen.

Special thanks goes to:

- Peter Teunissen and Christiaan Tiberius who co-shaped the material, indirectly, through collaboration with the author as TU Delft colleagues.
- The books *Adjustment theory: an introduction* (Teunissen, 2024) and *Testing theory: an introduction* (Teunissen, 2024) which provided the framework for this chapter.
- Sophie Keemink, Caspar Jungbacker and Thirza Feenstra, who provided feedback and helped develop exercises.

# Chapter: Numerical Modelling

Numerical Modelling is written by Jaime Arriaga Garcia, Justin Pittman and Robert Lanzafame.

Special thanks goes to:

- Isabel Slingerland and Mona Devos for critical feedback and development of exercises, figures and related content.

- Dhruv Mehta and Ajay Jagadeesh for feedback on structure, content.

Fig. 4.1 is included on page Numerical Modelling but is *not* included under the CC BY license of this book. Original content is used here with explicit permission of Amgad Omer on behalf of Deltares.

Fig. 4.6 is reproduced from Strang and Herman (2016) without modification and is *not* included under the CC BY license of this book. The source content is provided with a CC BY NC SA license and can be accessed for free at https://openstax.org/books/calculus-volume-2/pages/1-introduction.

## Chapters: Univariate and Multivariate Continuous Distributions

Univariate Continuous Distributions and Multivariate Distributions are written by Patricia Mares Nasarre and Robert Lanzafame.

Special thanks goes to Oswaldo Morales Napoles and Elisa Ragno for suggestions on the theoretical and didactic framework, as well as critical feedback and review.

Pages One Random Variable and Two Random Variables are from the Chapter *Probabilistic Design* (Lanzafame, 2024) from the book Risk and Reliability for Engineers (Lanzafame, 2024), published with a CC BY license. Files are included without modification.

Fig. 2.4, Fig. 5.1, Fig. 5.3, Fig. 6.1 and Fig. 6.11 are created by Max Ramgraber (maxramgraber.com/interactive), which are published with a CC BY license and included in this book without modification.

## Chapter: PDEs and the Finite Volume Method

PDEs and the Finite Volume Method is written by Robert Lanzafame and Jaime Arriaga Garcia.

Special thanks goes to:

- Isabel Slingerland and Mona Devos for critical feedback and development of exercises, figures and related content.

- Dhruv Mehta for providing a first draft of the chapter structure and contents.

# Chapter: Finite Element Method

Finite Element Method is written by Frans van der Meer.

*The material in this chapter is also incorporated in an in-depth book "Finite Elements in Civil Engineering and Geosciences" by Oriol Colomés, Iuri Rocha, Frans van der Meer and Martin Lesueur which can be found here.*

Special thanks goes to Lex Niessen who greatly assisted in developing material on the finite element method for the first edition of MUDE, which was the starting point for this chapter.

# Chapter: Signal Processing

Signal Processing is written by Christiaan Tiberius.

*The material in this chapter is related to an in-depth book "Engineering signal analysis - from Fourier to filtering" by Christiaan Tiberius and Max Mulder (TU Delft Open Publishing, 2025).*

Special thanks goes to:

- Max Mulder for being a signal processing soul-mate, sparring-partner and TU Delft colleague of the author who co-shaped the material, indirectly, through collaboration since early 2000's.
- Jelle Knibbe, who reviewed, commented and/or modified content of original powerpoint slides.
- João Moura Pereira de Lucas Teixeira, created first draft of pages from powerpoint slides.
- Antonio Magherini, who reviewed, commented and/or modified content.

Fig. 3.18 is included on page Fourier Series but is *not* included under the CC BY license of this book. Original content licensed under CC BY-SA 4.0 by BFG (2020) and can be found here; used here without modification.

# Chapter: Time Series Analysis

Time Series Analysis is written by Alireza Amiri-Simkooei, Christiaan Tiberius and Sandra Verhagen.

*The initial framework and contents of this chapter were created by Alireza Amiri-Simkooei, which was then revised and updated by Sandra and Christian.*

Special thanks goes to:

- Berend Bouvy, who created a number ofinteractive figures and exercises, as well as provided critical feedback.
- Serge Kaplev, who provided critical feedback and suggestions for the theoretical content.
- Antonio Magherini, who created the first draft material from powerpoint slides and prepared notebooks as exercises.

The following resources are used in this chapter but are *not* included under the CC BY license of this book:

- Fig. 4.17 is used on page Time Series Analysis (not modified) and is from IPCC (2018).
- Fig. 4.19 is used on page Components of time series (not modified) and is from CSIRO (n.d.).

# Chapter: Optimization

Optimization is written by Gonçalo Homem de Almeida Correia, Maria Nogal Macho, Jie Gao and Bahman Ahmadi.

*Gonçalo Homem de Almeida Correia created most of the material. Maria Nogal Macho and Bahman Ahmadi made contributions to various parts. Bahman Ahmadi developed the exercises in Python and Jupyter notebooks. Jie Gao created the genetic algorithm material.*

Special thanks goes to:

- Jialei Ding and Nadia Pourmohammadzia, who reviewed material and made improvements to the traffic exercise.
- Tom van Woudenberg, who edited text and improved content and structure for online interactive textbook format.
- João Moura Pereira de Lucas Teixeira, who created first draft of pages from powerpoint slides.

The Road Network Design Problem pages are included in this chapter but are *not* included under the CC BY license of this book; they are in-class exercises that will be shared by the authors of this book as part of a future publication (also under a CC BY license; citation will be provided here after publication).

Several figures are included in this chapter but are *not* included under the CC BY license of this book:

- Fig. 5.31 is used on page Taxonomy of optimization models; the convex 3D figure is from Agrawal (2021) and the non-convex 3D figures is from Nogal and Nogal (2021). Both are included as part of the figure shown in this book modification.
- Fig. 5.36 is used on page Genetic Algorithm and is from Daneshian (2025) (left) and the figure on the right is from an unknown source. Both are included as part of the figure shown in this book without modification.
- Figures Fig. 5.37, Fig. 5.38, Fig. 5.39, Fig. 5.40, Fig. 5.41, Fig. 5.42, Fig. 5.43, Fig. 5.44 and Fig. 5.45 on page Genetic Algorithm are included under the CC BY license of this book, as they are from unknown sources and will be replaced.

# Chapter: Introduction to Machine Learning

Introduction to Machine Learning is written by Iuri Rocha, Anne Poot, Joep Storm and Leon Riccius.

# Chapter: Extreme Value Analysis

Extreme Value Analysis is written by Patricia Mares Nasarre.

Special thanks goes to Oswaldo Morales Napoles, Elisa Ragno and Robert Lanzafame for suggestions on the theoretical and didactic framework, as well as critical feedback

> and review.

# External Resources

The following chapters and pages are included directly from an external resource, are not included under the CC BY license of this book. Unless otherwise noted below, the contents have not been edited by the authors of this book.

## Chapter: Risk Analysis

> Risk Analysis reuses the Chapters *Risk Analysis* (Jonkman and Lanzafame, 2024) and *Risk Evaluation* (Jonkman, 2024) and associated exercises from the book *Risk and Reliability for Engineers* (Lanzafame, 2024), published with a CC BY license. Files were modified to provide short introduction and explanation for readers (i.e., MUDE students).

## Programming Chapters

The chapters in the Programming part of this book are reused from two sources: *Learn Programming for Engineers* (Lanzafame and van Woudenberg, 2024) and *Python for Engineers* (Lanzafame *et al.*, 2024). Both books are published with a CC BY license and are available online at teachbooks.io/learn-programming and teachbooks.io/learn-python.

All chapters are from *Learn Programming for Engineers*, except Chapters 1.6 and 1.8 (Errors and Sympy, respectively), which are from *Python for Engineers.* Content has been modified slightly to fit the MUDE context.

# Contact

If you have questions on the content, contact the MUDE team at MUDE-CEG@tudelft.nl. If you have technical questions regarding this book, contact the IT-coordinator of MUDE (Tom): T.R.vanWoudenberg@tudelft.nl