TU Delft

Delft University of Technology

Applying optimization algorithms in different types of optimization problems

van Woudenberg, T.R.; Nogal Macho, M.

DOI
10.5281/zenodo.15099966

Publication date
2024

Document Version
Final published version

Citation (APA)
van Woudenberg, T. R., & Nogal Macho, M. (2024). *Applying optimization algorithms in different types of optimization problems*. Delft University of Technology. https://doi.org/10.5281/zenodo.15099966

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.

Introduction

This is the `2024-2025`-version of the book. Go to /CME4501 to view the most recent version of this book, or adapt the year in /CME4501/2024 to the year when you took the course.

This book focuses on how to apply optimization algorithms to specific optimization problems. It does so using python, scipy and pymoo. It contains the preparations for the practical sessions in CME4501 Engineering Systems Optimization. All other information about this course is available on Brightspace.

How to use this interactive book

Contents

This book focuses on how to apply optimization algorithms to specific optimization problems. It contains the preparations for the practical sessions in CME4501 Engineering Systems Optimization. All other information about this course is available on Brightspace.

Interactive features

This book includes interactive features!

- Interactive quizzes: Some page include interactive 'H5p' quizzes. This allows you to check your understanding. Any interactions you have with this system are not stored, unless you login into H5p.
- Python interactivity: Click 🚀 –> **Live Code** on the top right corner of interactive page to start up a python-kernel in your browser! Any interactions you do here are not stored. You can also download the page as a notebook to apply the content on your own computer.

Spot a mistake?

If you spot any mistakes, you can click on  –>  , login with a GitHub account and report your issue. It'll be solved soon!

Personalised book

If you'd like to make this TeachBook more personal by adding (private or public) annotations I can recommend the Hypothesis extension. This is only for your own use, I won't monitor public post on this platform.

Version

This is the `2024-2025`-version of the book. Go to /CME4501 to view the most recent version of this book, or adapt the year in /CME4501/2024 to the year when you took the course.

Contact information

Contents

This module is taught by Maria Nogal and Tom van Woudenberg. The practical sessions are taught by Tom. Please contact Tom if you've any questions, feedback or when you've personal circumstances which we should know.

Tom van Woudenberg

- Room 6.45
- 015-2789739
- T.R.vanWoudenberg@tudelft.nl

Python preparations

In this book you can use the interactive python functionality to run the code in your browser. However, you'll need to install Python to be able to do the analysis on your own computer. The following chapter will guide you through the installation, show you the basics of Jupyter Notebooks and Python, and show you a template of how we'll code optimization problems.

Python installation instruction

Contents

Python is a popular language for research computing, and great for general-purpose programming as well. Installing all of its research packages individually can be a bit difficult, so you're recommended to use Anaconda, an all-in-one installer. To make sure you're on the same versions you can set up an 'environment'.

Installation Anaconda

We will teach Python using the Jupyter Notebook, a programming environment that runs in a web browser (Jupyter Notebook will be installed by Anaconda). For this to work you will need a reasonably up-to-date browser. The current versions of the Chrome, Safari and Firefox browsers are all supported (some older browsers, including Internet Explorer version 9 and below, are not). Notebooks can be run after you install Python and the appropriate packages.

Installation on Windows | **Installation on MAC** | **Installation on Linux**

1. https://www.anaconda.com/products/individual#download-section with your web browser.
2. Download the Anaconda for Windows installer with Python 3. (If you are not sure which version to choose, you probably want the 64-bit Graphical Installer Anaconda3-…-Windows-x86_64.exe)
3. Install Python 3 by running the Anaconda Installer, using all of the defaults for installation except make sure to check Add Anaconda to my PATH environment variable.

Source: Carpentries [2023]

Exercise 1

Set up the environment

Now we will make sure you have an environment to get started in this course. This year we will use Python 3.11, so we are going to make sure that's installed and ready to go. Other Python packages that are not included in the base installation will be installed in your environment when we need to use them, to make sure we know exactly what is going on when working with code in this course.

The following steps will create an Anaconda environment called `optimization` and install Python version 3.11. Even if you already have Python 3.11, it is still good practice to create a dedicated Anaconda environment for each of your major projects. Please make sure you have upgraded Anaconda before proceeding with these steps:

1. Open Anaconda prompt from the start menu

2. Execute: `conda create -n optimization python=3.11 numpy matplotlib scipy pymoo` (this may take several minutes)
3. Activate: `conda activate optimization`
4. Check: you should now see `optimization` displayed somewhere in the prompt between parenthesis, like this: `(optimization)`

One important reminder: throughout the course you should be using the optimization environment every time you use Python, except when instructed otherwise. All you have to do is remember to use the command `conda activate optimization` prior to opening your files, or select the new environment in the Anaconda Navigator prior to opening any python-program.

Add other packages

If you want to install additional packages, you can do so by using `pip install <package name>`. Make sure you do this from within your activated environment.

Source: MUDE [2023]

Jupyter Notebook and Python basics

Contents

After installing Anaconda and setting up the environment, start the Navigator and select the `optimization environment`. This can be done on the top of the screen at `All application` on `base (root)`: change `base (root)` to `optimization`. Or in the left tab under `Environments`, click `optimization.` Then, launch the Jupyter Notebook from the list of applications under `home`, which will open the Jupyter Notebook dashboard in your webbrowser (starting in your Documents folder on Windows and your home directory on a Mac):

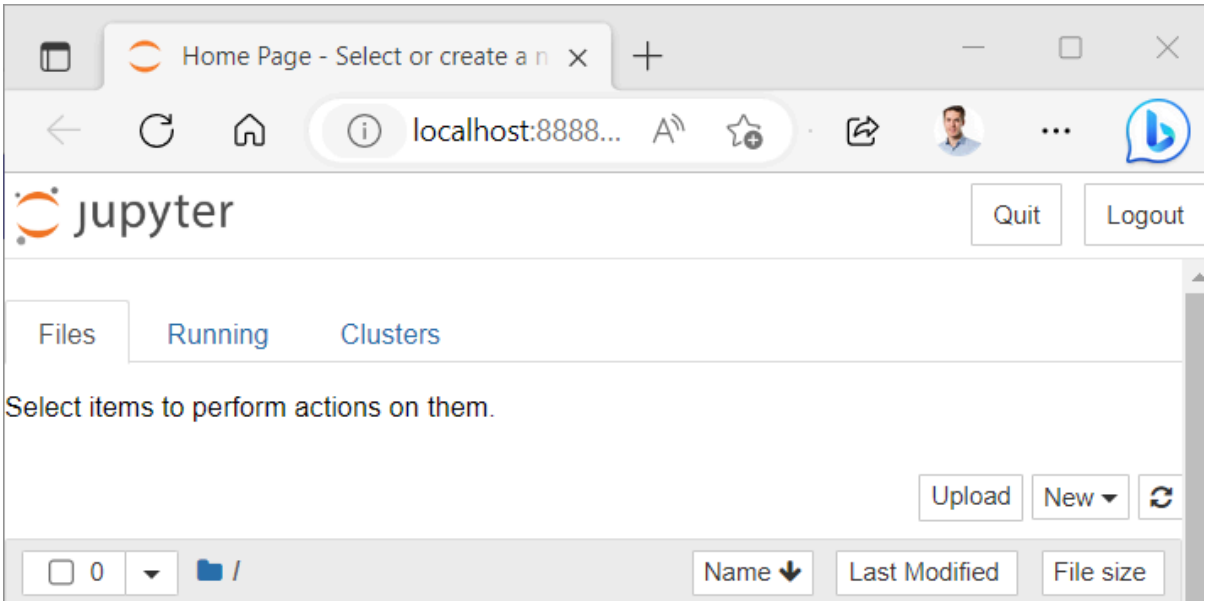


Fig. 1 Jupyter notebook start page

Use the dashboard to surf to the directory where you stored your Notebooks (you can only surf to lower directories not higher directories) and click on the one you want to open. Launch a new notebook by clicking on the “New” button and then selecting “Python 3”:

Exercise 2

Working in a Jupyter Notebook

Notebooks consist of cells which can contain code or text/figures (in markdown language). You can choose this cell type for each cell:

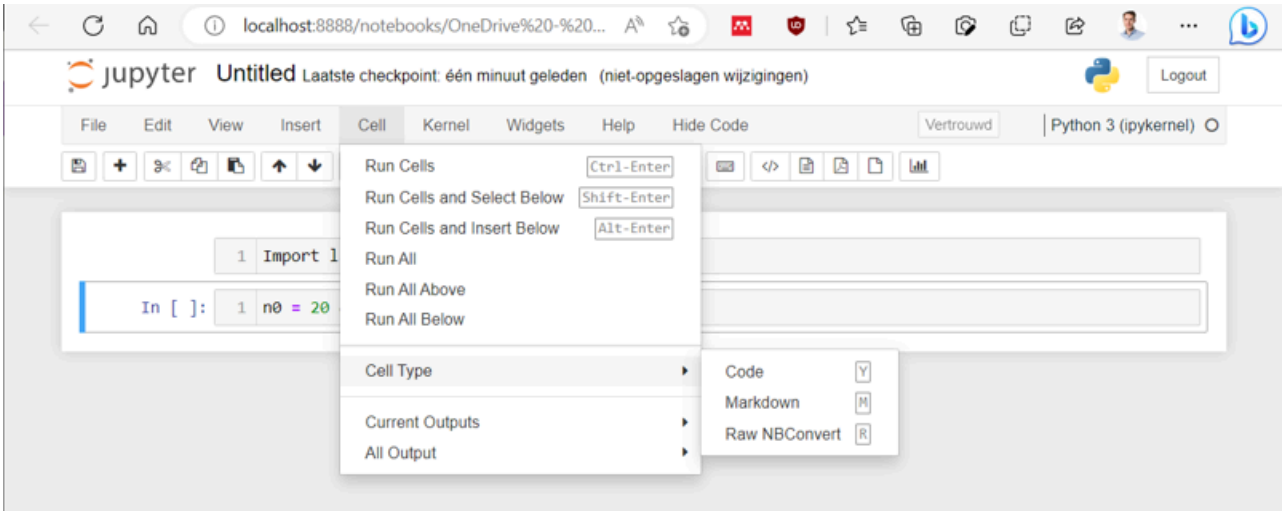


Fig. 2 Jupyter Notebook cell types

Both code and text cells can be run by clicking `execute` or pressing `shift + enter`. Code cells run, resulting in a counter between the square brackets, while text cells are just rendered:

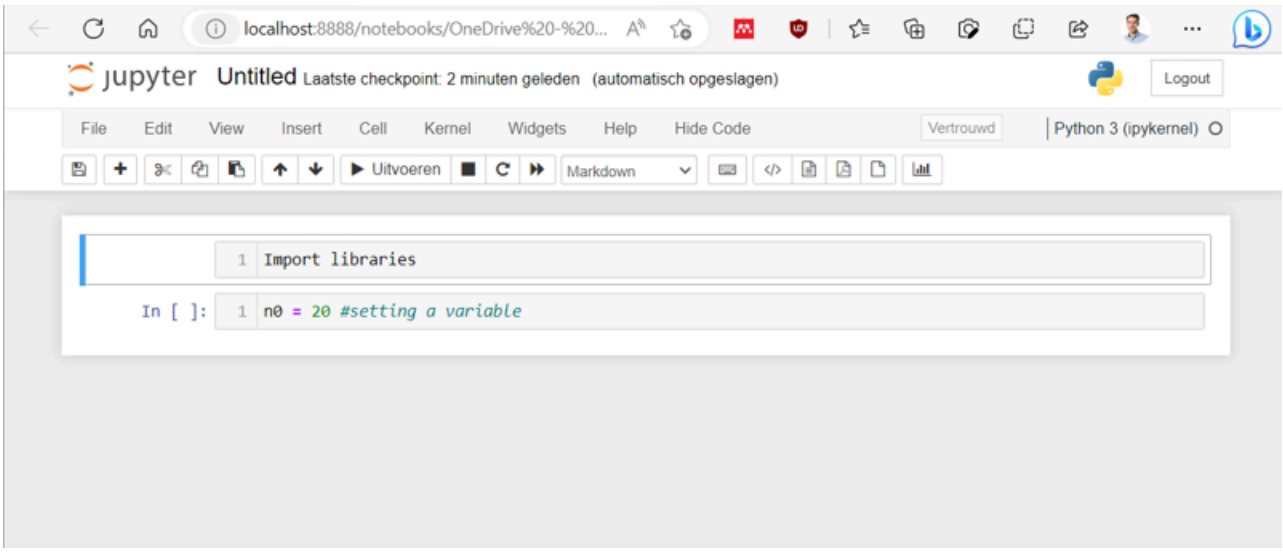


Fig. 3 Before running cells

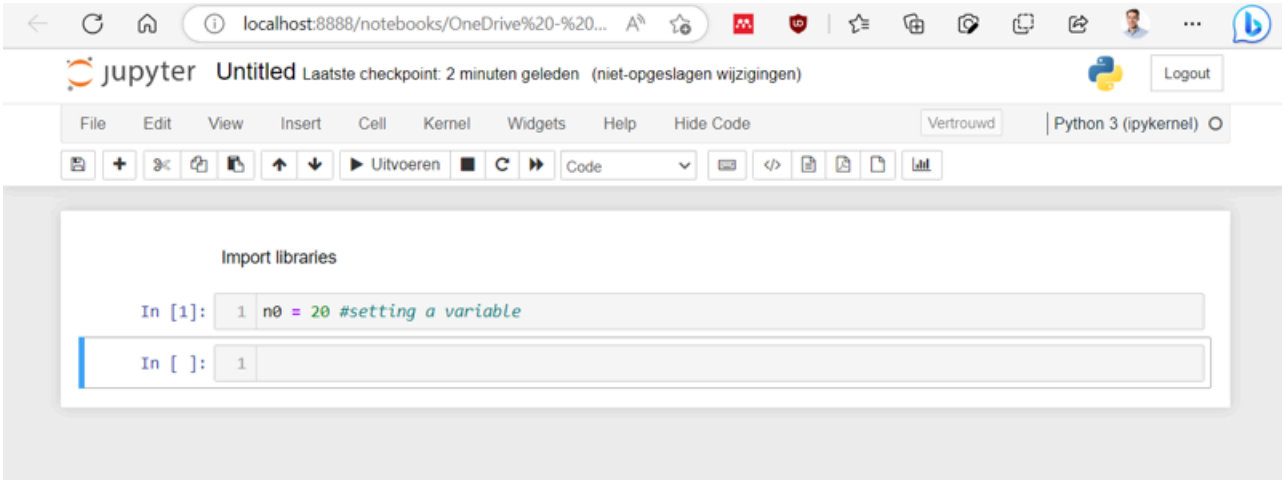


Fig. 4 After running cells

Exercise 3

Tutorial Python

If you've no experience in using Python, please work through this material on your own. Specifically, for this course you'll need:

1. Variables, operators and functions.
2. Modules, conditions, data structures and loops
3. Numpy
4. Matplotlib
5. Errors

Template for optimization problems in Jupyter Notebooks

Contents

For this course, our Jupyter Notebooks follow the same structure for all problems by defining the following sections:

- Import libraries
- Define variables
- Define objective function
- Define constrain function(s) if needed
- Solve the problem
- Postprocess results if needed

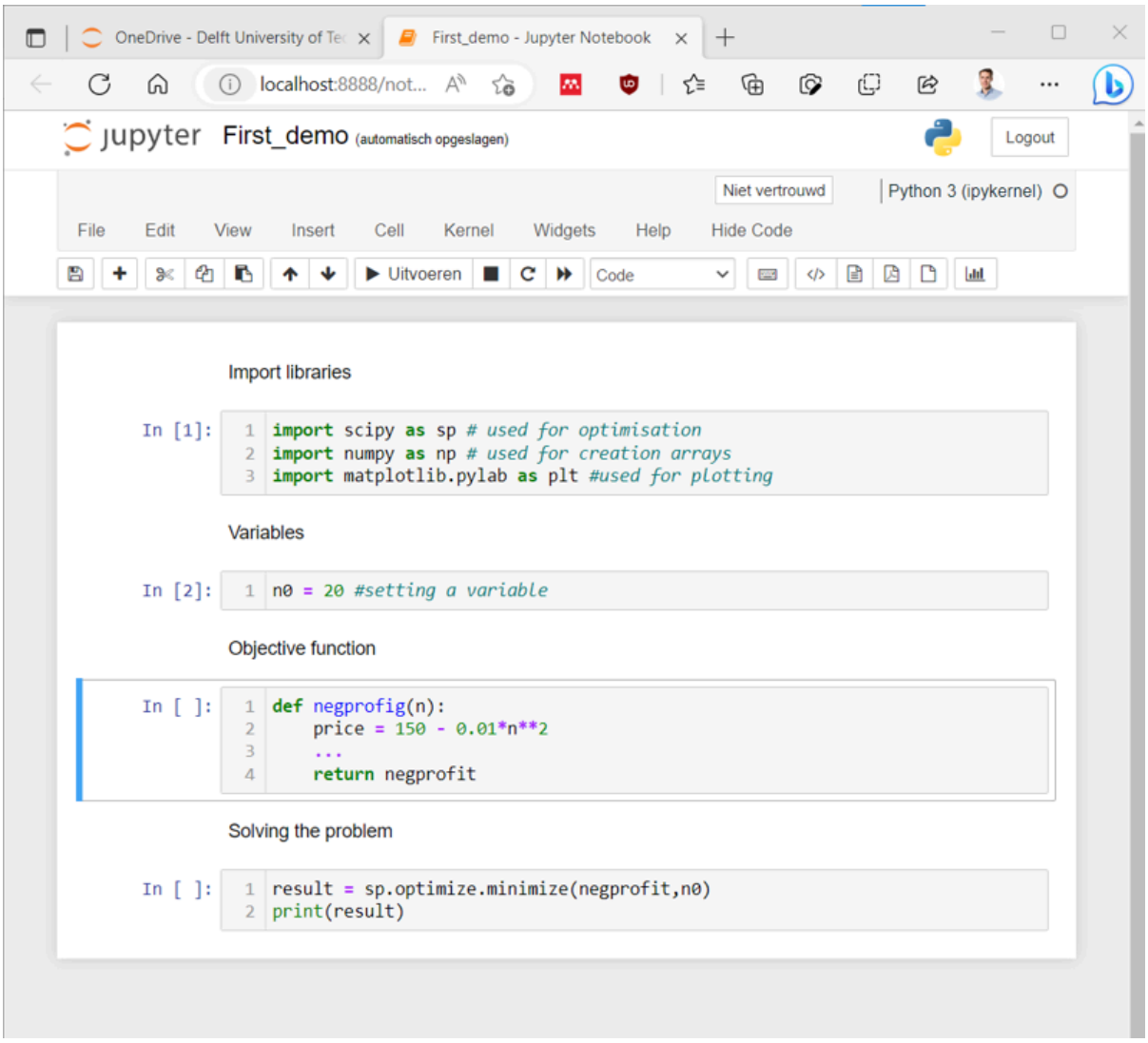


Fig. 5 General layout Jupyter Notebooks in this course

The structure is shown for an example which will be treated in-depth later.

Import libraries

Python cannot do optimization on its own, therefore we make use of separate packages which are installed with Anaconda:

- `scipy` used for optimization
- `numpy` used for matrix algebra
- `matplotlib` used for plotting

Libraries are imported with `import ... as ...` which directly abbreviates the packages for later use.

```
import scipy as sp
import numpy as np
import matplotlib.pylab as plt
```

Exercise 4

Define variables

In this part of the notebook we can store all our variables in the form of integers, floats and arrays.

```
n0 = 20
```

Exercise 5

Define objective function

In this part we define our constraint function as a `callable` with `def ...: ... return ...` These functions can contain multiple lines of calculation but should return one values which is minimized.

```
def negprofit(n):
    price = 150 - 0.01 * n**2
    revenues = price * n
    totalcost = 75 * n
    profit = revenues - totalcost
    return -profit
```

Exercise 6

Define constraint function

Similar to the objective function, the constrain function(s) can be defined, which may return multiple values

Solve the problem

Making using of the `scipy`-library, the problem is solved. Using `print`, the result is shown

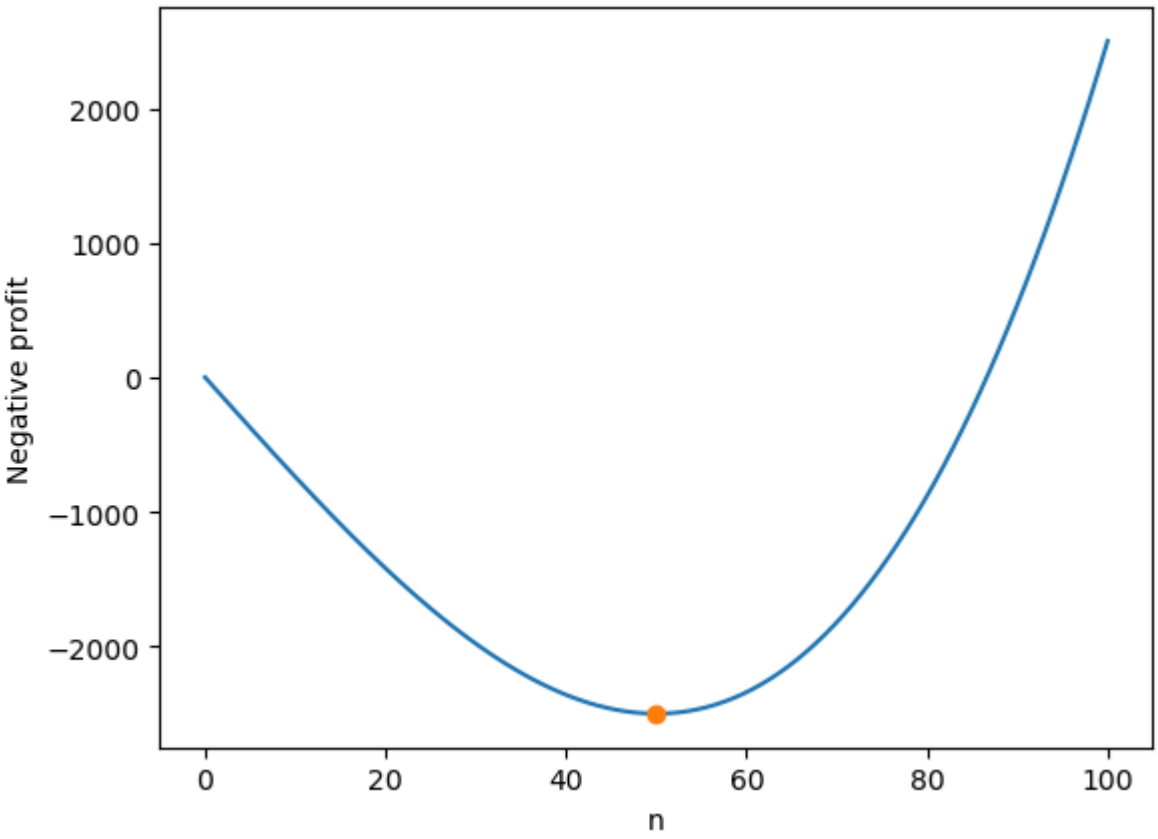
```
result = sp.optimize.minimize(negprofit,n0)
print(result)
```

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: -2499.9999999998727
        x: [ 5.000e+01]
      nit: 8
      jac: [ 0.000e+00]
 hess_inv: [[ 3.503e-01]]
     nfev: 22
     njev: 11
```

Postprocess results

If needed, we can postprocess the result or analyse the problem in another way

```
# checking the results with exhaustive search

n_range = np.linspace(0,100,100)
negprofit_result = negprofit(n_range)
plt.plot(n_range,negprofit_result)
plt.plot(result.x,result.fun,'o')
plt.xlabel('n')
plt.ylabel('Negative profit');
```



Exercise 7

Unconstrained optimization using scipy

Contents

In this chapter, we'll cover how to apply `scipy.optimize.minimize` to unconstrained optimization problems. As a reminder, unconstrained optimization considers:

$$\min_x f(x) \tag{1}$$

with:

- x: the design variable of length n
- f: the objective function.

Method

In this course, we're making use of the function `scipy.optimize.minimize`. The documentation of this function is available here: https://docs.scipy.org/doc/scipy-1.15.0/reference/generated/scipy.optimize.minimize.html [The SciPy community, 2024]. In this course we'll cover only the relevant parts.

For unconstrained optimization we need to run at least `scipy.optimize.minimize(fun, x0, ...)` with:

- `fun`, the objective function $f(x)$ to be minimized. `fun` is a callable. The `scipy.optimize.minimize` function takes care of defining and inputting our design variable x.
- `x0`, the initial guess for our design variable x. It needs to be a `ndarray` with length n

The function `scipy.optimize.minimize` outputs an object `scipy.optimize.OptimizeResult`. with:

- `scipy.optimize.OptimizeResult.x` the optimized solution of the design variable x. It is a `ndarray` with length n
- `scipy.optimize.OptimizeResult.success`, a indication whether or not the optimizer was executed successfully. It is a `bool`, indicating `True` or `False`
- `scipy.optimize.OptimizeResult.message`, a message describing the cause of termination of the optimization algorithm. It is a `str`.
- `scipy.optimize.OptimizeResult.fun`, the values of the optimized objective function f. It is a `int` or `float`
- `scipy.optimize.OptimizeResult.nit`, the number of iteration performed by the optimizer. It is a `int`

Exercise 8

Homework exercise: Bathymetry maps

Contents

Click 🚀 –> Live Code to activate live coding on this page!

Problem

It is desired to determine the number of bathymetry maps n of a local area that should be produced to maximize the profit of a company. The total cost of production and distribution is €75 per unit n. The revenues are proportional to the number of units multiplied by its price: $Revenues = n \cdot Price$

The demand depends on the price ($n = \sqrt{100 \cdot (Price - 150)}$), as shown in the graph:

▶ Show code cell source

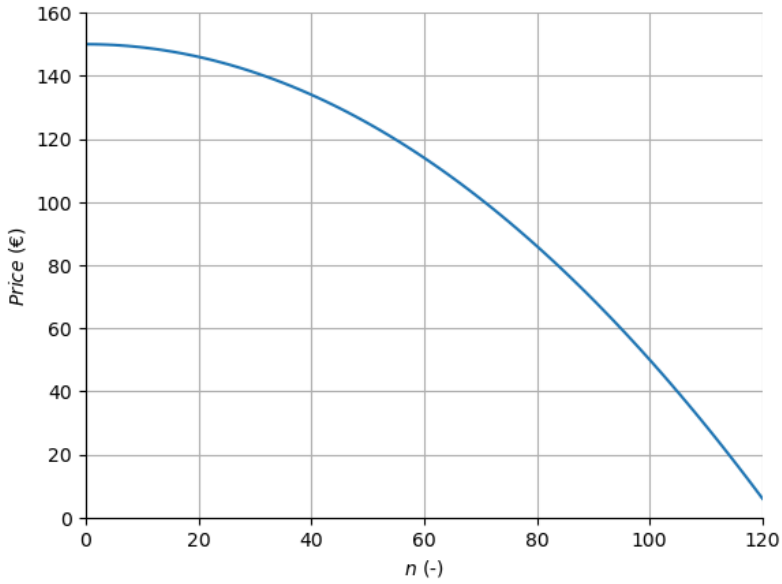


Fig. 6 $Price = 150 - 0.01n^2$

The profit can be estimated as the revenues minus the total costs.

This problem could be solved by taking the derivative and solving the root of the derivative. However, those steps might not be possible for more complex problems, in that case optimization is required.

Model

We need to define our problem in the form of unconstrained optimization (1).

The function for the profit can be found by combining the relations in the problem statement. However, this is the profit which should be maximized. To turn this into a minimization problem, the profit can be multiplied with -1. The final model of this problem results in:

$$\min_n \left(75n - \left(150 - 0.01n^2\right)n\right) \tag{2}$$

Exercise 9

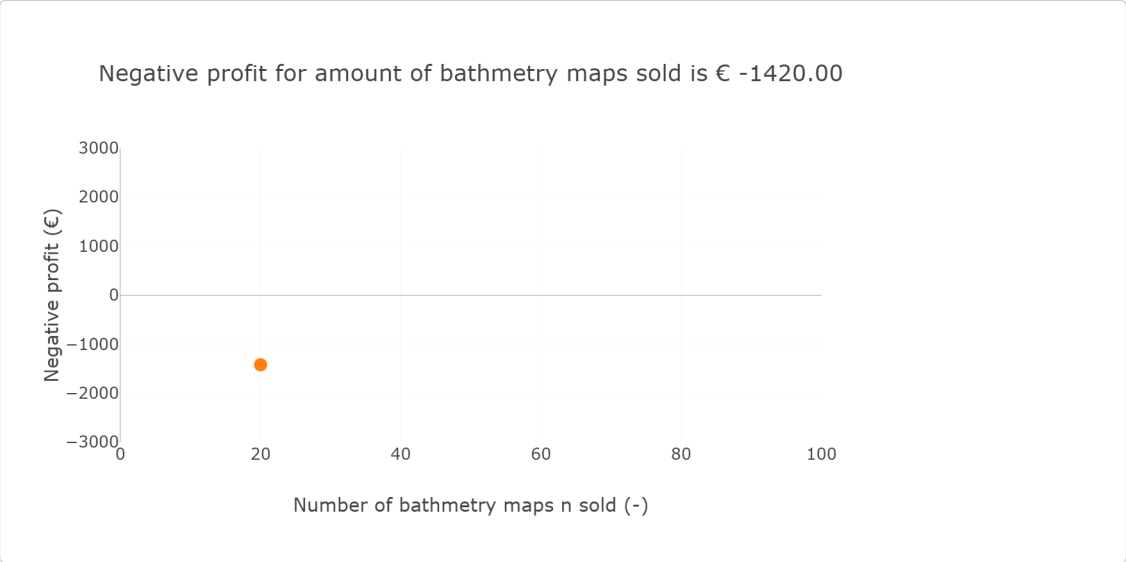
Find best solution manually

An approach to solve this problem might be to try out some values. You can do so in the applet below. The plot below shows the negative profit for some number of bathmetry maps sold.

Exercise 10

Try and adjust the values for n, the number of bathmetry maps sold. How small can you get the negative profit?

#maps n sold 20 Show objective function



As this case is only one-dimensional and the potential range of values is limited, this approach (exhaustive search) is valid: evaluating all possible values for n doesn't take a lot of computing power. The resulting values from the objective function show a clear minimum. For problem in which the objective function required more computational power or the amount of dimensions of the design variables increases, this approach quickly becomes infeasible.

Method

Now let's solve this problem using an optimization method. This model is described using `scipy.optimize.minimize` according to the standard structure in this course

Import libraries

For this problem, we'll use all three packages `scipy`, `numpy`, `matplotlib`.

```
import scipy as sp
import numpy as np
import matplotlib.pylab as plt
```

Define the variables

There are very few variables in this problem. In fact, the only variable we have to specify is the initial guess for the optimization algorithm. The objective function will be treated later. The length of n doesn't have to be specified.

```
1 n0 = 20
```

run | run all | add cell | clear

Exercise 11

Define the objective function

In the objective function, the formula derived above in (2) ean be inserted. Or, each individual step can be calculated on a seperate line. Again, note that the profit is multiplied with -1 to maximize the profit in the minimization formulation. This results in:

```
1 def negprofit(n):
2     price = 150 - 0.01 * n**2
3     revenues = price * n
4     totalcost = 75 * n
5     profit = revenues - totalcost
6     return -profit
```

run | run all | add cell | clear

Solve the problem

Now, the problem can be solved. The result is stored in the variables `result` which is printed.

```
result = sp.optimize.minimize(negprofit,n0)
print(result)
```

run | run all | add cell | clear

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: -2499.9999999998727
        x: [ 5.000e+01]
      nit: 8
      jac: [ 0.000e+00]
 hess_inv: [[ 3.503e-01]]
     nfev: 22
     njev: 11
```

Exercise 12

Postprocess results

As seen before, this problem is very small and can be solved by evaluating all possible values (or applying algebra). These values can be plotted and the optimum solution is clearly in the minimum.

```
n_range = np.linspace(0,100,100)
negprofit_result = negprofit(n_range)
plt.figure()
plt.plot(n_range,negprofit_result)
plt.plot(result.x,result.fun,'o');
plt.xlabel('$n$')
plt.ylabel('Negative profit');
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
```

run | run all | add cell | clear

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[8], line 1
----> 1 n_range = np.linspace(0,100,100)
      2 negprofit_result = negprofit(n_range)
      3 plt.figure()

NameError: name 'np' is not defined
```

Exercise 13

Exercise 14

Click 🚀 –> Live Code and adapt the code to answer the following question.

Homework exercise: Curve-fitting

Contents

Click 🚀 –> **Live Code** to activate live coding on this page!

Problem

8 data points are collected where X_i is the explanatory variable and Y_i the response variable. It is believed that the data follows the trend $Y = aX^b + c$ where a, b and c are the parameters to be determined to fit the model.

X_i	Y_i
0.97	0.97
0	0.06
0.5	0.7
0.85	0.74
0.7	0.2
0.19	0.34
0.41	0.29
0.78	0.94

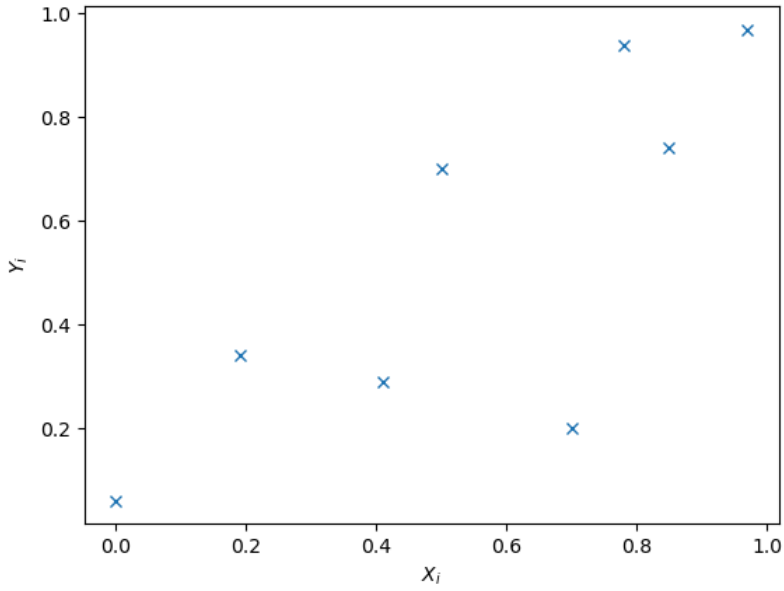


Fig. 7 8 data points

Examples of the source of this data can be:

- Stress of a pre-stressed concrete specimen based on the deformation when applied an axial force

- Permeability-coefficient for compacted sand based on the grain size
- Job performance based on the number of working hours
- etc.

Model

Although it's possible to solve this problem too with a direct mathematical approach which doens't require optimization, we'll define our problem in the form of unconstrained optimization (1).

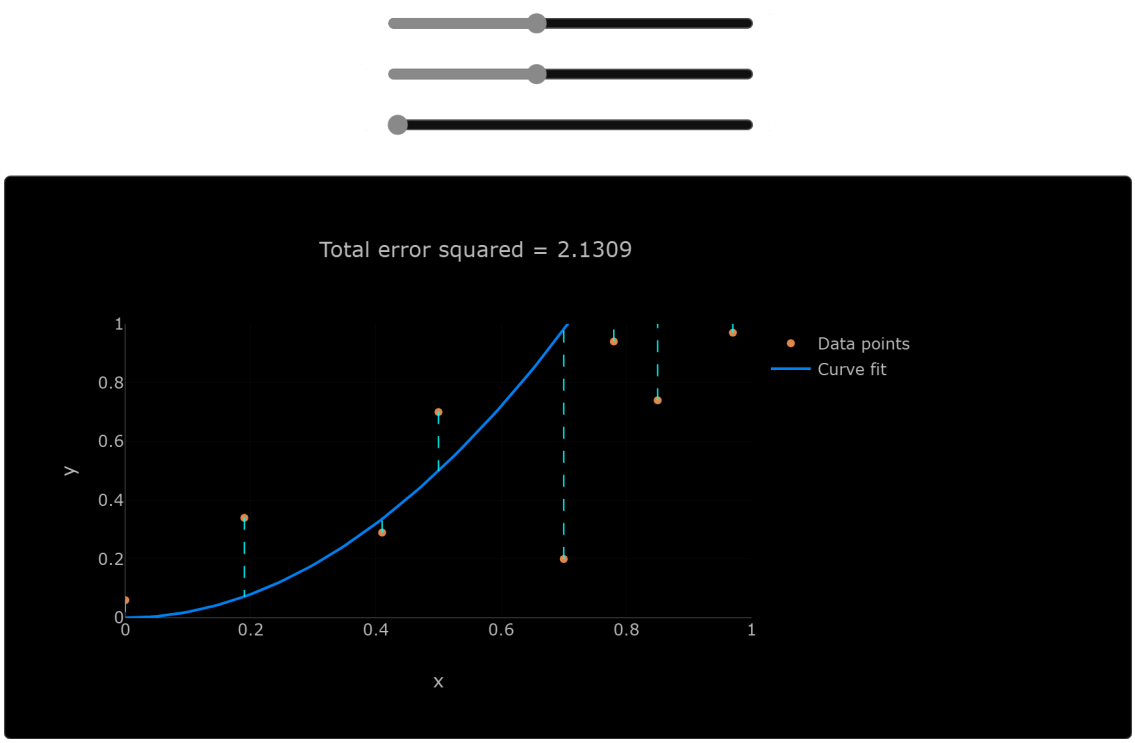
The problem is modelled as a minimization of the squared error, where error is the difference between $aX_i^b + C$ and Y_i

$$\min_{a,b,c} \sum_{i=1}^{8} \left(aX_i{}^b + c - Y_i \right)^2 \tag{3}$$

Find best solution manually

Exercise 15

Try and adjust the values for a, b and c. How small can you get the total error squared?



Method

Again, this model is described using `scipy.optimize.minimize` according to the standard structure in this course

Import libraries

```
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
```

Define the variables

In this problem, there are 3 design variables: a, b and c. In `scipy.optimize`, these variables are part of one array. You don't have to specify this variable beforehand, but the optimization algorithm will define the number of variables on your other input. Furthermore, input data is given which is required to specify explicitly. Additionally, an initial guess is required for the otimization algorithm.

```
xi = np.array([0.97, 0   , 0.5, 0.85, 0.7, 0.19, 0.41, 0.78])
yi = np.array([0.97, 0.06, 0.7, 0.74, 0.2, 0.34, 0.29, 0.94])
abc0 = np.array([4., 2., 0.5]) #initial guess
```

run | run all | add cell | clear

Define objective function

The objective function was defined in (3). Because `a`, `b`, `c` and `xi` are all numpy arrays, we can simply use `*`, `**` and `+` to deal with the multiplication, exponent and summation of those arrays. `np.sum` is used to sum up all components of the array which is the result of `(yi-y_est)**2`

```
def squarederror(abc):
    a = abc[0]
    b = abc[1]
    c = abc[2]
    y_est = a * xi ** b + c
    error = np.sum((yi-y_est)**2)
    return error
```

run | run all | add cell | clear

Solve the problem

Now we can solve the actual problem

```
result2 = sp.optimize.minimize(squarederror,abc0)
print(result2)
```

run | run all | add cell | clear

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 0.33422039804883197
        x: [ 8.028e-01  1.231e+00  1.233e-01]
      nit: 22
      jac: [-8.568e-07  1.006e-07 -1.036e-06]
hess_inv: [[ 6.552e-01 -4.308e-01 -3.826e-01]
           [-4.308e-01  1.803e+01  2.737e+00]
           [-3.826e-01  2.737e+00  6.369e-01]]
     nfev: 96
     njev: 24
```

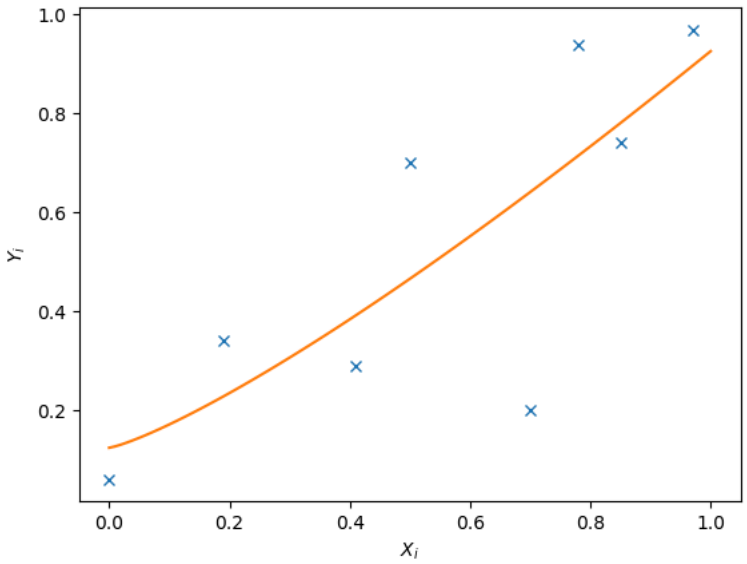
Exercise 16

Postprocess results

As this problem involves three design variables, a plot of all possible design variables with the objective function is not possible because it would require four dimensions. However, we can plot the data which the obtained solution to see how it looks like

```
x_range = np.linspace(0,1,100)
plt.plot(xi,yi,'x');
plt.plot(x_range,result2.x[0] * x_range ** result2.x[1] + result2.x[2])
plt.xlabel('$X_i$')
plt.ylabel('$Y_i$');
```

run | run all | add cell | clear



Exercise

Exercise 17

Click 🚀 –> **Live Code** and adapt the code to answer the following question.

Linear constrained optimization using scipy

Contents

In this chapter, we'll cover how to apply `scipy.optimize.linprog` to linear constrained optimization problems. As a reminder, linear constrained optimization considers:

$$\begin{aligned} &\min_x f(x) \\ &\text{such that} \quad g_j(x) \leq 0 \qquad j = 1, m \\ &\qquad\qquad\quad h_k(x) = 0 \qquad k = 1, p \\ &\qquad\qquad\quad x_i^l \leq x_i \leq x_i^u \; i = 1, n \end{aligned} \tag{4}$$

with:

- $f(x)$, the linear objective function
- x, the n design variables
- $g_j(x)$, the m linear inequality constraints
- $h_k(x)$, the p linear inequality constraints
- x_k^l and x_k^u, the n low er and upper bounds of the design variable

Because all functions are linear, this problem can be rewritten as:

$$\begin{aligned} &\min_x \; c^T x \\ &\text{such that} \quad A_{ub} x \leq b_{ub} \\ &\qquad\qquad\quad A_{eq} x = b_{eq} \\ &\qquad\qquad\quad x_i^l \leq x_i \leq x_i^u \; i = 1, n \end{aligned} \tag{5}$$

with:

- x, the n design variables

- c, the n coefficients of the linear objective function
- A_{ub}, the inequality constraint matrix of m inequality constraints
- b_{ub}, the inequality constraint vector of m inequality constraints
- A_{eq}, the equality constraint matrix of p equality constraints
- b_{eq}, the equality constraint vector of p equality constraints
- x_i^l and x_i^u, the n lower and upper bounds of the design variable

Method

For linear programs, we can use the function `scipy.optimize.linprog`. In contrast to `scipy.optimize.minimize`, this function is limited to linear functions. The documentation of this function is available here: https://docs.scipy.org/doc/scipy-1.15.0/reference/generated/scipy.optimize.linprog.html [The SciPy community, 2024]. In this course we'll cover only the relevant parts.

Again, we won't use all options, but a minimum requirement for our problem is the command `scipy.optimize.linprog(c, A_ub, b_ub, A_eq, b_eq, bounds, ...)` with:

- `c`, a onedimensional numpy array with the n coefficients of the linear objective function c.
- `A_ub`, a twodimensional numpy array with the n coefficient of the m linear inequality constraints matrix A_{ub}.
- `b_ub`, a onedimensional numpy array with the upper bound of the m linear inequality constraint vector b_{ub}.
- `A_eq`, a twodimensional numpy array with the n coefficient of the p linear equality constraints matrix A_{eq}.
- `b_eq`, a onedimensional numpy array with value of the p linear equality constraint vector b_{eq}.
- `Bounds`: A sequence of i `(min, max)` pairs for each element in x, defining the minimum x_i^l and maximum values x_i^u of that decision variable. Use `None` to indicate that there is no bound.

The function `scipy.optimize.linprog` outputs an object `scipy.optimize.OptimizeResult` similar as `scipy.optimize.minimize` explained for unconstrained optimization.

Exercise 18

Homework exercise: Book distribution

Contents

Click 🚀 –> **Live Code** to activate live coding on this page!

Problem

We'll consider planning the shipment of books from distribution centers to stores where they are needed. There are three distribution centers at different cities: Amsterdam, Den Haag and Rotterdam. They have 250, 130 and 235 books. There are four stores in Haarlem, Utrecht, Delft and Breda. They ordered 70, 230, 240 and 70 books. All stores can receive books from all distribution centers, as shown below:

There are the following costs in € of transportation per book:

From \ To	Haarlem	Utrecht	Delft	Breda
Amsterdam	7	11	16	26
Den Haag	7	13	5	20
Rotterdam	16	28	7	12

The goal is to minimize the shipping costs while meeting demand.

Model

We need to define our model in the form of a linear constrained optimization model (4) and to apply `scipy.optimize.linprog` in matrix notation (5).

We'll define the model as follows:

- Design variables: how many books go from which distribution center to which store
- Objective function: minimum shippings costs
- Inequality constraint functions: don't run out of stock in warehouses
- Equality constraint functions: meet demand of stores
- Bounds: book are only transported from distribution centers to stores.

Find best solution manually

Exercise 19

Try and adjust the book transports yourself. Can you find the optimal solution?

Amsterdam → Haarlem 0 books

Amsterdam → Utrecht 0 books

Amsterdam → Delft 0 books

Amsterdam → Breda 0 books

Den Haag → Haarlem 0 books

Den Haag → Utrecht 0 books

Den Haag → Delft 0 books

Den Haag → Breda 0 books

Rotterdam → Haarlem 0 books

Rotterdam → Utrecht 0 books

Rotterdam → Delft 0 books

Rotterdam → Breda 0 books

Total costs: €0

bankrupt @ Amsterdam

bankrupt @ Den Haag

bankrupt @ Rotterdam

didn't receive the correct amount of books @ Haarlem: 70 too few books received

didn't receive the correct amount of books @ Utrecht: 230 too few books received

didn't receive the correct amount of books @ Delft: 240 too few books received

didn't receive the correct amount of books @ Breda: 70 too few books received

Design variables

Let's start with our design variables. In this case that could be the amount of books x transported from each distribution center $i = [1\,(\text{Amsterdam}), 2\,(\text{Den Haag}), 3\,(\text{Rotterdam})]$ to each store $j = [1\,(\text{Haarlem}), 2\,(\text{Utrecht}), 3\,(\text{Delft}), 4\,(\text{Breda})]$:

$$x_{ij} = \begin{bmatrix} x_{\text{Amsterdam}\ \to\text{Haarlem}} & x_{\text{Amsterdam}\ \to\text{Utrecht}} & x_{\text{Amsterdam}\ \to\text{Delft}} & x_{\text{Amsterdam}\ \to\text{Breda}} \\ x_{\text{Den Haag}\ \to\text{Haarlem}} & x_{\text{Den Haag}\ \to\text{Utrecht}} & x_{\text{Den Haag}\ \to\text{Delft}} & x_{\text{Den Haag}\ \to\text{Breda}} \\ x_{\text{Rotterdam}\ \to\text{Haarlem}} & x_{\text{Rotterdam}\ \to\text{Utrecht}} & x_{\text{Rotterdam}\ \to\text{Delft}} & x_{\text{Rotterdam}\ \to\text{Breda}} \end{bmatrix} \tag{6}$$

$$x_{ij} = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & x_{22} & x_{23} & x_{24} \\ x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}$$

Exercise 20

Alternatively, the design variables can be expressed as follows:

$$x = \begin{bmatrix} x_{\text{Amsterdam}} & \rightarrow \text{Haarlem} \\ x_{\text{Amsterdam}} & \rightarrow \text{Utrecht} \\ x_{\text{Amsterdam}} & \rightarrow \text{Delft} \\ x_{\text{Amsterdam}} & \rightarrow \text{Breda} \\ x_{\text{Den Haag}} & \rightarrow \text{Haarlem} \\ x_{\text{Den Haag}} & \rightarrow \text{Utrecht} \\ x_{\text{Den Haag}} & \rightarrow \text{Delft} \\ x_{\text{Den Haag}} & \rightarrow \text{Breda} \\ x_{\text{Rotterdam}} & \rightarrow \text{Haarlem} \\ x_{\text{Rotterdam}} & \rightarrow \text{Utrecht} \\ x_{\text{Rotterdam}} & \rightarrow \text{Delft} \\ x_{\text{Rotterdam}} & \rightarrow \text{Breda} \end{bmatrix} \tag{7}$$

$$x = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} & x_{21} & x_{22} & x_{23} & x_{24} & x_{31} & x_{32} & x_{33} & x_{34} \end{bmatrix}^T$$

Objective function

Now we can define the costs as the sum of the amount of books transported with the cost per transport as in (4)in the form of $\min_x f(x_{ij})$:

$$\min_x f(x_{ij}) = 7x_{11} + 11x_{12} + 16x_{13} + 26x_{14} + 7x_{21} + 13x_{22} + 5x_{23} + 20x_{24} + 16x_{31} + 28x_{32} + 7x_{33} + 12x_{34} \tag{8}$$

This can be converted to matrix notation. In case of the design variables defined as in (5) in the form $\min_x c^T x$ with c:

$$c = \begin{bmatrix} 7 & 11 & 16 & 26 & 7 & 13 & 5 & 20 & 16 & 28 & 7 & 12 \end{bmatrix}^T \tag{9}$$

Exercise 21

Inequality constraints

Let's continue with the inequality constraints, which should deal with that the stock of the distribution centers should always be bigger than 0. Or, stated differently, the sum of the amount of books transported out of each distribution center should be small or equal than the stock of each distribution center. These can be defined in the form $g(x_{ij}) \leq 0$ as:

$$
\begin{aligned}
g_1\left(x_{ij}\right) &= x_{11} + x_{12} + x_{13} + x_{14} - 250 \\
g_2\left(x_{ij}\right) &= x_{21} + x_{22} + x_{23} + x_{24} - 130 \\
g_3\left(x_{ij}\right) &= x_{31} + x_{32} + x_{33} + x_{34} - 235
\end{aligned}
\tag{10}
$$

Or in matrix notation in the form $A_{ub}x \leq b_{ub}$ as:

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{bmatrix} x \leq \begin{bmatrix} 250 \\ 130 \\ 235 \end{bmatrix}
\tag{11}
$$

Exercise 22

Equality constraints

Finally, let's define the inequality constraints, which should deal with that the each store receives the correct amount of books. Or, stated differently, the sum amount of paper books transported to each store should be equal to the demand of that store. These can be defined in the from $h\left(x_{ij}\right) = 0$ as:

$$
\begin{aligned}
h_1\left(x_{ij}\right) &= x_{11} + x_{21} + x_{31} - 70 \\
h_2\left(x_{ij}\right) &= x_{12} + x_{22} + x_{32} - 230 \\
h_3\left(x_{ij}\right) &= x_{13} + x_{23} + x_{33} - 240 \\
h_3\left(x_{ij}\right) &= x_{14} + x_{24} + x_{34} - 70
\end{aligned}
\tag{12}
$$

Or in matrix notation in the form $A_{eq}x = b_{eq}$ as:

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1
\end{bmatrix} x = \begin{bmatrix} 70 \\ 230 \\ 240 \\ 70 \end{bmatrix}
\tag{13}
$$

Bounds

The number of books cannot be negative (assuming that book are only transported from distribution centers to stores). The maximum number of books transported could be the number of book available in a distribution center, but this is already defined by the constraint functions. Therefore, the bounds can be defined as:

$$
0 < x_i \; i = 1, 2, \ldots, 12
\tag{14}
$$

Method

Now let's solve this problem using an optimization method.

Import libraries

As this problem is higher-dimensional, we cannot easily plot the solution space. Therefore, we won't import `matplotlib`.

```
import scipy as sp
import numpy as np
```

Define variables

As before, the (length of) design variable itself doesn't have to be specified. So there's actually nothing be be done here.

Define objective function

The objective function was defined in (9), which gives:

```
c = np.array([7, 11, 16, 26, 7, 13, 5, 20, 16, 28, 7, 12])
```

run | run all | add cell | clear

Define constraint functions

The constraint functions were defined in (11) and (13), which can be coded as follows:

```
A_ub = np.array([[1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0],
                 [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0],
                 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1]])
b_ub = np.array([250, 130, 235])

A_eq = np.array([[1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0],
                 [0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0],
                 [0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0],
                 [0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1]])
b_eq = np.array([70, 230, 240, 70])
```

run | run all | add cell | clear

Bounds

The bounds were defined in (14) and result in:

```
bounds = np.array([[0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None],
          [0, None]])
```

run | run all | add cell | clear

Solve the problem

```
result = sp.optimize.linprog(c, A_ub, b_ub, A_eq, b_eq, bounds)
print(result)
```

run | run all | add cell | clear

```
       message: Optimization terminated successfully. (HiGHS Status 7: Optimal)
       success: True
        status: 0
           fun: 5380.0
             x: [ 2.000e+01  2.300e+02  0.000e+00  0.000e+00  5.000e+01
                  0.000e+00  8.000e+01  0.000e+00  0.000e+00  0.000e+00
                  1.600e+02  7.000e+01]
           nit: 7
         lower:  residual: [ 2.000e+01  2.300e+02  0.000e+00  0.000e+00
                             5.000e+01  0.000e+00  8.000e+01  0.000e+00
                             0.000e+00  0.000e+00  1.600e+02  7.000e+01]
                marginals: [ 0.000e+00  0.000e+00  1.100e+01  1.600e+01
                             0.000e+00  2.000e+00  0.000e+00  1.000e+01
                             7.000e+00  1.500e+01  0.000e+00  0.000e+00]
         upper:  residual: [       inf        inf        inf        inf
                                   inf        inf        inf        inf
                                   inf        inf        inf        inf]
                marginals: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             0.000e+00  0.000e+00  0.000e+00  0.000e+00
                             0.000e+00  0.000e+00  0.000e+00  0.000e+00]
         eqlin:  residual: [ 0.000e+00  0.000e+00  0.000e+00  0.000e+00]
                marginals: [ 9.000e+00  1.300e+01  7.000e+00  1.200e+01]
       ineqlin:  residual: [ 0.000e+00  0.000e+00  5.000e+00]
                marginals: [-2.000e+00 -2.000e+00 -0.000e+00]
mip_node_count: 0
mip_dual_bound: 0.0
       mip_gap: 0.0
```

Exercise 23

Click 🚀 –> Live Code to activate live coding on this page.

Class exercise: Furniture manufacturing

> ***Added in version v2024.1.0:*** After class February 19th
>
> Solutions in text and downloads

A furniture manufacturing company wishes to determine how many tables, chairs, desks and bookcases it should make to optimize its profit.

These products utilize two different types of lumber, the company has on hand:

- 1500 m of type 1
- 1000 m of type 2.

There are 800 labour-hours available for the job.

Sales forecasts plus back orders require the company to make at least:

- 40 tables
- 130 chair
- 30 desk
- no more than 10 bookcases.

Each table, chair, desk, and bookcase requires:

- of lumber type 1: 5, 1, 9, and 12 m, respectively
- of lumber type 2: 2, 3, 4, and 1 m, respectively.

A table requires 3 labour-hours to make, a chair 2 hours, a desk 5 hours, and a bookcase 3 hours.

The company makes a total of profit:

- €12 on a table
- €5 on a chair
- €15 on a desk

- €10 on a bookcase.

Exercise 24 (Model)

Write out the linear programming model of this problem.

Solution to Exercise 24 (Model)

$$\min_x \quad [-12 \quad -5 \quad -15 \quad -10]x$$

with $$x = \begin{bmatrix} x_{table} \\ x_{chair} \\ x_{desk} \\ x_{bookcases} \end{bmatrix}$$

such that $$\begin{bmatrix} 5 & 1 & 9 & 12 \\ 2 & 3 & 4 & 1 \\ 3 & 2 & 5 & 3 \end{bmatrix} x \leq \begin{bmatrix} 1500 \\ 1000 \\ 800 \end{bmatrix}$$

$$[1 \quad -4 \quad 0 \quad 0]x = 0$$

$$\begin{bmatrix} 40 \\ 160 \\ 30 \\ 0 \end{bmatrix} \leq x \leq \begin{bmatrix} \infty \\ \infty \\ \infty \\ 10 \end{bmatrix}$$

Exercise 25 (Method)

Now let's solve this problem using an optimization method.

Click 🚀 –> Live Code to activate live coding on this page!

```
import scipy as sp
import numpy as np
```

```
#YOUR CODE HERE
```

Solution to Exercise 25 (Method)

```
import scipy as sp
import numpy as np
```

```
c = np.array([-12, -5, -15, -10])
```

```
A_ub = np.array([[5, 1, 9, 12],
                 [2, 3, 4, 1 ],
                 [3, 2, 5, 3]])
b_ub = np.array([1500, 1000, 800])
```

```
bounds = np.array([[40, None],
          [130, None],
          [30, None],
          [0, 10]])
```

```
result = sp.optimize.linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=None, b_eq=None, bour
print(result.x)
print(result.fun)
```

```
[130. 130.  30.   0.]
-2660.0
```

The optimal solution is:

- 130 tables
- 130 chairs
- 30 desk
- 0 bookcases

With a profit of €2660

Exercise 26 (Additional exercise)

What happens if you impose that four chairs should be made for every table?

```
#YOUR CODE HERE
```

Solution to Exercise 26 (Additional exercise)

```
A_eq = np.array([[4,-1,0,0]])
b_eq = np.array([0])
```

```
result = sp.optimize.linprog(c, A_ub=A_ub, b_ub=b_ub, A_eq=A_eq, b_eq=b_eq, bour
print(result.x)
print(result.fun)
```

```
[ 40. 160.  66.  10.]
-2370.0
```

A different solution is found with a lower profit:

- 40 tables
- 160 chairs
- 66 desk
- 10 bookcases

With a profit of €2370

Nonlinear constrained optimization using scipy

Contents

In this chapter, we'll cover how to apply `scipy.optimize.minimize` to nonlinear constrained optimization problems. As a reminder, nonlinear constrained optimization considers:

$$\begin{aligned} &\min_x f(x) \\ &\text{such that} \quad g_j(x) \leq 0 \qquad j = 1, m \\ &\qquad\qquad\quad h_k(x) = 0 \qquad k = 1, p \\ &\qquad\qquad\quad x_i^l \leq x_i \leq x_i^u \; i = 1, n \end{aligned} \tag{15}$$

with:

- $f(x)$, the linear or nonlinear objective function.
- x, the n design variables
- $g_j(x)$, the m linear or nonlinear inequality constraints
- $h_k(x)$, the p linear or nonlinear inequality constraints
- x_k^l and x_k^u, the n low er and upper bounds of the design variable

Method

For linear programs, we can use the function `scipy.optimize.minimize` again. The documentation of this function is available here: https://docs.scipy.org/doc/scipy-1.15.0/reference/generated/scipy.optimize.minimize.html [The SciPy community, 2024]. In this course we'll cover only the relevant parts.

For unconstrained optimization we need to run at least `scipy.optimize.minimize(fun, x0, bounds, constraints, ...)` with:

- `fun`, the function representing the objective function $f(x)$ to be minimized. `fun` is a callable. The `scipy.optimize.minimize` function takes care of defining and inputting our design variable x.
- `x0`, the initial guess for our design variable x. It needs to be a `ndarray` with length n
- `Bounds`: A sequence of i `(min, max)` pairs for each element in x, defining the minimum x_i^l and maximum values x_i^u of that decision variable. `None` is used to specify no bound.
- `constraints`, a single or a list of constraint objective either being:
  - `scipy.optimize.LinearConstraint`
  - `scipy.optimize.NonlinearConstraint`

Exercise 27

As you can see, the constraints are stored in an object `scipy.optimize.LinearConstraint` and/or `scipy.optimize.NonlinearConstraint`. These function have the following input, for `scipy.optimize.NonlinearConstraint(fun, lb, ub, ...)`:

- `fun`, the function representing the constraint function $g(x)$ or $h(x)$ to be minimized. Again, `fun` is a callable. The `scipy.optimize.minimize` function takes care of defining and inputting our design variable x.
- `lb` and `ub`, two arrays containing the lower- and upper bounds for each of the constraint functions $g(x)$. This lower bound can be `-np.inf` or `np.inf` to represent one-sides constraints. If the lower- and upper bound are set to the same value, an equality function is modelled.

For linear constraints, the constraint function is stored in a matrix again: `scipy.optimize.NonlinearConstraint(A, lb, ub, ...)`:

- `A`, a twodimensional numpy array with the n coefficient of the m linear inequality constraints matrix A_{ub}.
- `lb` and `ub` as for `scipy.optimize.NonlinearConstraint`

Please note that unlike with linear constraints optimization, the right-hand-side of the constraints are not stored in an upper bound vector, but defined with `lb` and `ub`.

Exercise 28

The function `scipy.optimize.linprog` outputs an object `scipy.optimize.OptimizeResult` similar as `scipy.optimize.minimize` explained for unconstrained optimization.

Homework exercise: Breakwater blocks

Contents

Click 🚀 –> Live Code to activate live coding on this page!

Problem

It is desired to design the cheapest breakwater block (box-shaped). The price of the brick will depend only on the amount of used material. It is required that each face has a minimum surface of 0.8 m^2. Also, for stability reasons, the block weight has to be larger than 3000 kg. Let's assume concrete density of 2500 kg$/m^2$.

Fig. 8 Breakwater blocks

Model

We need to define our model in the form of a nonlinear constrained optimization model (15) to apply `scipy.optimize.minimize`.

We'll define the model as follows:

- Design variables: width, height and depth of a block
- Objective function: minimum volume of the block
- Inequality constraint functions: minimum surface area of each face of 0.8 m^2 and maximum weight of 3000 kg
- Equality constraint functions: none
- Bounds: positive dimensions

Design variables

Let's start with our design variables. In this case a logical choice could be the width, height and depth of our block

$$x = \begin{bmatrix} x_{width} \\ x_{depth} \\ x_{height} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{16}$$

Objective function

Now we can define the objective function as the product of the dimension to represent $\min_x f(x)$ in (15):

$$\min_x f(x) = x_1 \cdot x_2 \cdot x_3 \tag{17}$$

Inequality constraints

Let's continue with the inequality constraints, which should deal with the required positive dimensions, minimum surface area of each face of 0.8 m^2 and maximum weight of 3000 kg. These can be defined in the form $g(x_{ij}) \leq 0$ as:

$$\begin{aligned} g_1(x) &= -x_1 \cdot x_2 + 0.8 \\ g_2(x) &= -x_2 \cdot x_3 + 0.8 \\ g_3(x) &= -x_1 \cdot x_3 + 0.8 \\ g_4(x) &= -x_1 \cdot x_2 \cdot x_3 \cdot 2500 + 3000 \end{aligned} \tag{18}$$

Exercise 29

Bounds

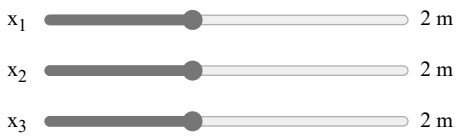
The dimensions of the block cannot be negative. Therefore, the bounds can be defined as:

$$0 < x_i \, i = 1, 2, 3 \tag{19}$$

Find best solution manually

Exercise 30

Try and adjust the values for x_1, x_2 and x_3. Can you find the optimal solution?



Objective function: 8.000 m^3

- Constraint function 1: -3.20 m^2
- Constraint function 2: -3.20 m^2
- Constraint function 3: -3.20 m^2
- Constraint function 4: -17000 kg

Method

Now let's solve this problem using an optimization method.

Import libraries

```
import scipy as sp
import numpy as np
```

Define variables

As before, we don't need to specify our variable x itself as defined in (16). However, this optimization method requires an initial guess. An arbitrary value is chosen here:

```
x0 = np.array([5,0,1])
```

run | run all | add cell | clear

Define objective function

The objective function was defined in (17), which gives:

```
def func(x):
    vol = x[0]*x[1]*x[2]
    return vol
```

run | run all | add cell | clear

Define constrain functions

The constraint functions were defined in (18). We had no equality constraints. Unlike before with the method to solve linear constrained problem, we need an object which defines the upper and lower bounds. As this problem has only an upper bound of 0, the lower bound is set to ∞ which is `np.inf` in python. Note that a single constraint object can include multiple constraints.

```
def nonlinconfun(x):
    c1 = 0.8 - x[0]*x[1]
    c2 = 0.8 - x[0]*x[2]
    c3 = 0.8 - x[1]*x[2]
    c4 = 3000 - 2500 * x[0] * x[1] * x[2]
    return np.array([c1,c2,c3,c4])

cons = sp.optimize.NonlinearConstraint(nonlinconfun, np.array([-np.inf,-np.inf,-np.inf,-np.inf]), np.
```

run run all add cell clear

Define bounds

The bounds were defined in (19) and result in:

```
bounds = [[0, None],
          [0, None],
          [0, None]]
```

run run all add cell clear

Solve the problem

Now let's solve the problem. The `cons` object can be added directly, in the case of equality constraints as well you can define a list of constrainer objects as an input.

```
result = sp.optimize.minimize(fun = func,x0 = x0,bounds = bounds,constraints=cons)
print(result)
```

run run all add cell clear

Exercise 31

Click 🚀 –> Live Code to activate live coding on this page.

Class exercise: Suspended beams

> **Added in version v2024.2.0:** After class February 26th
>
> Solutions in text and downloads

3 beams are suspended on ropes A, B, C, D, E and G:

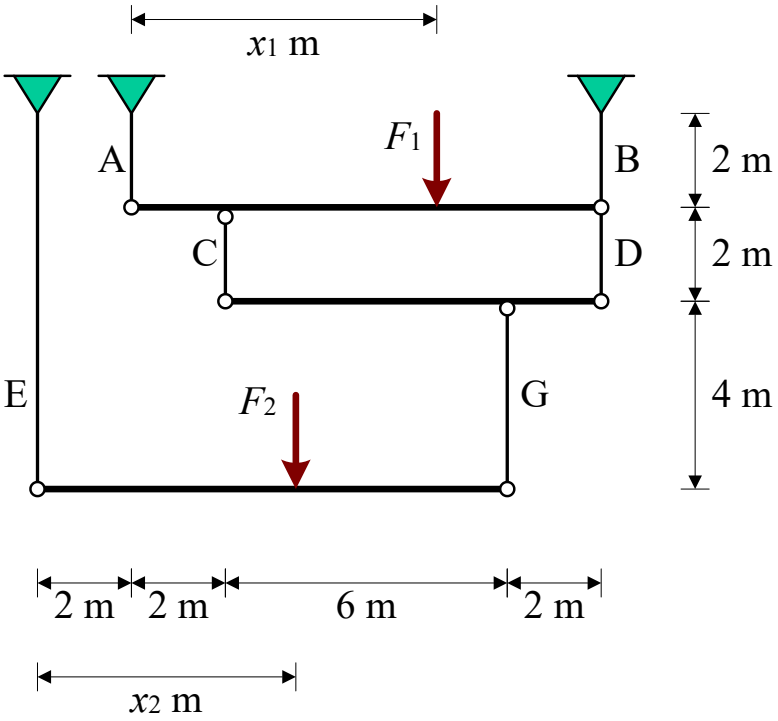


Fig. 9 Suspended beams

Find the permissible total load which doesn't exceed the maximum tension N in rope A and B, C and D, and E and G of respectively 300 N, 200 N and 100 N. What is the tension in the ropes for the permissible total load?

> **Exercise 32 (Model)**
>
> Write out the nonlinear programming model of this problem.

Solution to Exercise 32 (Model)

From mechanics it follows that:

$$
\begin{aligned}
\sum F_{\mathrm{v}}^{\mathrm{EG}} &= 0 \rightarrow N_E + N_G - F_2 = 0 \\
\sum T_{\mathrm{E}}^{\mathrm{EG}} &= 0 \rightarrow 10N_G - x_2 F_2 = 0 \\
\sum F_{\mathrm{v}}^{\mathrm{CD}} &= 0 \rightarrow N_C + N_D - N_G = 0 \\
\sum T_{\mathrm{C}}^{\mathrm{CD}} &= 0 \rightarrow 8N_D - 6N_G = 0 \\
\sum F_{\mathrm{v}}^{\mathrm{AB}} &= 0 \rightarrow N_A + N_B - N_C - N_D - F_1 = 0 \\
\sum T_{\mathrm{A}}^{\mathrm{AB}} &= 0 \rightarrow 10N_B - 10N_D - F_1 x_1 - 2N_C = 0
\end{aligned}
$$

The problem can be stated as follows:

$$
\min_x \left(-F_1 - F_2\right)
$$

with

$$
x = \begin{bmatrix} F_1 \\ F_2 \\ x_1 \\ x_2 \end{bmatrix}
$$

such that

$$
\begin{aligned}
N_A\left(F_1, F_2, x_1, x_2\right) &\leq 300 \\
N_B\left(F_1, F_2, x_1, x_2\right) &\leq 300 \\
N_C\left(F_1, F_2, x_1, x_2\right) &\leq 200 \\
N_D\left(F_1, F_2, x_1, x_2\right) &\leq 200 \\
N_E\left(F_1, F_2, x_1, x_2\right) &\leq 100 \\
N_G\left(F_1, F_2, x_1, x_2\right) &\leq 100 \\
0 &\leq F_1 \\
0 &\leq F_2 \\
0 \leq x_1 &\leq 10 \\
0 \leq x_2 &\leq 10
\end{aligned}
$$

This could be solved by solving the systems of equations when evaluation the constraints, or solving the systems of equation symbolically on beforehand. In the second case, we can rephrase the model as:

$$\min_x \left(-F_1 - F_2\right)$$

with $$x = \begin{bmatrix} F_1 \\ F_2 \\ x_1 \\ x_2 \end{bmatrix}$$

such that

$$-\frac{F_1 x_1}{10} + F_1 + \frac{F_2 x_2}{50} \leq 300$$

$$\frac{F_1 x_1}{10} + \frac{2F_2 x_2}{25} \leq 300$$

$$\frac{F_2 x_2}{40} \leq 200$$

$$\frac{3F_2 x_2}{40} \leq 200$$

$$-\frac{F_2 x_2}{10} + F_2 \leq 100$$

$$\frac{F_2 x_2}{10} \leq 100$$

$$0 \leq F_1$$

$$0 \leq F_2$$

$$0 \leq x_1 \leq 10$$

$$0 \leq x_2 \leq 10$$

Exercise 33 (Method)

Now let's solve this problem using an optimization method.

Click 🚀 –> **Live Code** to activate live coding on this page!

```
import scipy as sp
import numpy as np
```

```
#YOUR CODE HERE
```

Solution to Exercise 33 (Method)

```
import scipy as sp
import numpy as np
```

```
x0 = np.array([500, 100, 2, 9])
```

```
def func(x):
    return -x[0] - x[1]
```

```
# solving the systems of equations during every iteration
def nonlinconfun(x):
    F_1 = x[0]
    F_2 = x[1]
    x_1 = x[2]
    x_2 = x[3]
    A = np.array([[0,0,0,0,1,1],
                  [0,0,0,0,0,10],
                  [0,0,1,1,0,-1],
                  [0,0,0,8,0,-6],
                  [1,1,-1,-1,0,0],
                  [0,10,-2,-10,0,0]])
    b = np.array([F_2, x_2*F_2, 0, 0, F_1, F_1*x_1])
    sol = np.linalg.solve(A,b)
    return sol

cons = sp.optimize.NonlinearConstraint(nonlinconfun, np.array([-np.inf,-np.inf,-
```

```
bounds = [[0, None],
          [0, None],
          [0, 10],
          [0, 10]]
```

```
result = sp.optimize.minimize(fun = func, x0 = x0, bounds = bounds,constraints=c
print(result.fun)
print(result.x)
print(nonlinconfun(result.x))
```

```
-700.0000000000525
[549.95615294 150.04384706   4.7270118    3.33528152]
[300.         300.          12.51096176  37.53288529 100.
  50.04384706]
```

```
x0 = np.array([1000,1000,10,10])
```

```
result = sp.optimize.minimize(fun = func, x0 = x0, bounds = bounds,constraints=c
print(result.fun)
print(result.x)
print(nonlinconfun(result.x))
```

```
-700.0
[500.  200.    4.4   5. ]
[300. 300.  25.  75. 100. 100.]
```

We get different results for different initial guesses

- With initial guess: $F_1 = 500$, $F_2 = 100$, $x_1 = 2$, $x_2 = 9$:
  - Total permissible load ≈ 700
  - $F_1 \approx 550$, $F_2 = 150$, $x_1 = 4.73$, $x_2 = 3.34$
  - $N_A = 300$, $N_B = 300$, $N_C \approx 12.5$, $N_D \approx 37.5$, $N_E \approx 100$, $N_G \approx 50.0$
- With initial guess: $F_1 = 1000$, $F_2 = 1000$, $x_1 = 10$, $x_2 = 10$:
  - Total permissible load $= 700$
  - $F_1 = 500$, $F_2 = 200$, $x_1 = 4.4$, $x_2 = 5$
  - $N_A = 300$, $N_B = 300$, $N_C \approx 25$, $N_D \approx 75$, $N_E \approx 100$, $N_G \approx 100.0$

Multi-objective optimization using scipy

Contents

In this chapter, we'll cover how to solve multi-objective optimization problem using `scipy`. As a reminder, nonlinear constrained optimization considers:

$$\min_x f_1(x), f_2(x) \tag{20}$$

with:

- $f_1(x)$ and $f_2(x)$, the linear or nonlinear objective functions.
- x, the n design variables
- Constraints and bounds as for single-objective optimization problems.

Model

Three different ways of solving multi-objective optimization problems were introduced, which all effectively convert the problem to a single-objective optimization problem. All of this is assuming minimization problems:

1. Weighted objective function: setting pre-determined weight on the two objectives. In general this requires the two objectives to have a comparable unit:

$$\min_x \left(\delta_{1,\text{predefined}} \cdot f_1(x) + \delta_{2\text{predefined}} \cdot f_2(x)\right) \tag{21}$$

2. Goal attainment, minimizing the maximum difference with respect to two goal values for the objectives. Again, this requires the two objectives to have a comparable unit:

$$\min_x \left(\max \left(f_1\left(x\right) - f_{1,\text{goal}} \ , f_2\left(x\right) - f_{2,\text{goal}} \ \right)\right) \tag{22}$$

3. Pareto front: finding many possible optimal solution for a large set weights. It's good practise to normalize the objective functions.

$$\begin{gathered}\min_x \ \left(\delta_i \cdot f_{1,\text{normalized}} \quad (x) + \delta_j \cdot f_{2,\text{normalized}} \quad (x)\right) \\ \text{with } 0 < \delta_i < 1 \text{ and } \delta_j = 1 - \delta_i\end{gathered} \tag{23}$$

All of these methods could also be applied to problems which include more than two goals.

Exercise 34

Normalize objective functions

Normalizing the objectives functions can be done by setting the domain of every goal f between 0 and 1 by finding (or estimating) the lower and upper bounds for these objective functions within the domain:

$$\begin{gathered}f_{\text{normalized}} \quad (x) = \frac{f\left(x\right) - \min_x \ \left(f\left(x\right)\right)}{\max_x \ \left(f\left(x\right)\right) - \min_x \ \left(f\left(x\right)\right)} \\ \text{with } \min_x \ \left(f\left(x\right)\right), \max_x \ \left(f\left(x\right)\right) \text{ for } x_i^l \leq x_i \leq x_i^u \text{ with } i = 1, n\end{gathered} \tag{24}$$

Note that here are various ways to normalize functions / data. This is just an example.

Exercise 35

Homework exercise: engine power vs emissions

Contents

Click 🚀 –> Live Code to activate live coding on this page!

Problem

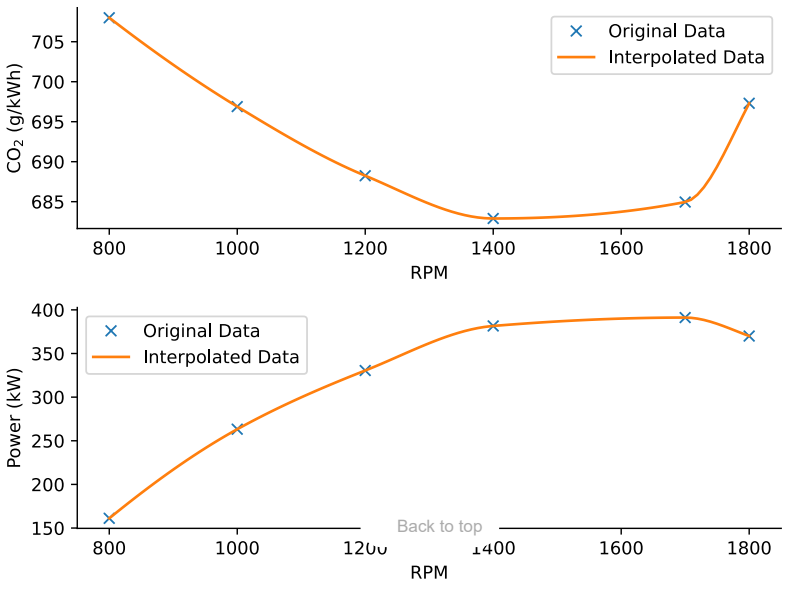
In this problem we're trying to optimize a diesel engine for maximum power and minimum CO_2 emissions. Depending on the optimal engine speed, the power and CO_2 emissions change.

The following data is available:

Rotations per minute	CO_2 Emissions	Power
800	708	161.141
1000	696.889	263.243
1200	688.247	330.51
1400	682.897	381.561
1700	684.955	391.17
1800	697.3	370

This data is interpolated to obtain a continuous relation:

▶ Show code cell source



Back to top

Fig. 10 Interpolation of CO_2 emissions and Power

Model

We'll define the model as follows:

- Design variables: scalar value representing the rotations per minute (RPM)
- Objective function: weighted sum of the power and CO_2 emissions, max of difference with goals or pareto front
- Bounds: only interpolation so between 800 and 1800 RPM

Design variables

Let's start with our design variables. In this case a logical choice could be the width, height and depth of our block

$$x = \text{rotations per minute} \tag{25}$$

Objective function

Let's explore all three possible objective functions as defined in (21), (22) and (23):

Weighted objective function

We can define the weighted objective function as defined by (21) with some predefined weights. For example a weight of $\frac{1}{3}$ for the power value and $\frac{2}{3}$ for the CO_2 emissions.

$$\min_x \left(-\frac{1}{3} \cdot f_{\text{power}} \quad (x) + \frac{2}{3} \cdot f_{\text{CO }_2 \text{ emissions}} \quad (x) \right) \tag{26}$$

Please note that we need a minus for the power objective because it's an maximization objective and we now apply the weights to the non-normalized functions while they have different units. Therefore, it might be wise to apply normalization as defined by (24).

Goal attainment

If we define two fails for the power and CO_2 emissions, we can apply goal attainment. Those goals could be 460 kW for the generated power and 640 g/kWh for the CO_2 emissions. Now the objective function can as in (22) as:

$$\min_x \left(\max \left(460 - f_{\text{power}}\left(x\right), f_{\text{CO}_2\text{emissions}}\left(x\right) - 640 \right) \right) \tag{27}$$

Pareto front

Finally, we can find the pareto front by defining the problem as in (23):

$$\min_x \left(-\delta_i \cdot f_{\text{power} \quad \text{,normalized}} \quad (x) + \delta_j \cdot f_{\text{CO }_2 \text{ emissions} \quad \text{,normalized}} \quad (x) \right) \tag{28}$$

For normalization, we could take the value from our plot, but for more real-life complex problems the dimensions are higher and you cannot find the maximum and minimum of each single objective function. Therefore, let's estimate the maxima and minima as:

	CO_2 Emissions	**Power**
Minimum value	680	150
Maximum value	710	400

Another approach would be to optimize for both CO_2 Emissions and Power separately, and use the respective objective function values as minimum and maximum values.

With a linear normalization, this gives:

$$
\begin{aligned}
f_{\text{power, normalized}}(x) &= \frac{f_{\text{power}}(x) - 150}{400 - 150} \\
f_{\text{CO}_2\text{ emissions, normalized}}(x) &= \frac{f_{\text{CO}_2\text{ emissions}}(x) - 680}{710 - 680}
\end{aligned}
\tag{29}
$$

To find the full pareto front this optimization model has to be solved for a large set of δ_i and δ_j

Bounds

Let's limit our solution to within our interpolation domain. Therefore, the bound can be defined as:

$$
800 < x < 1800 \tag{30}
$$

Find best solution manually

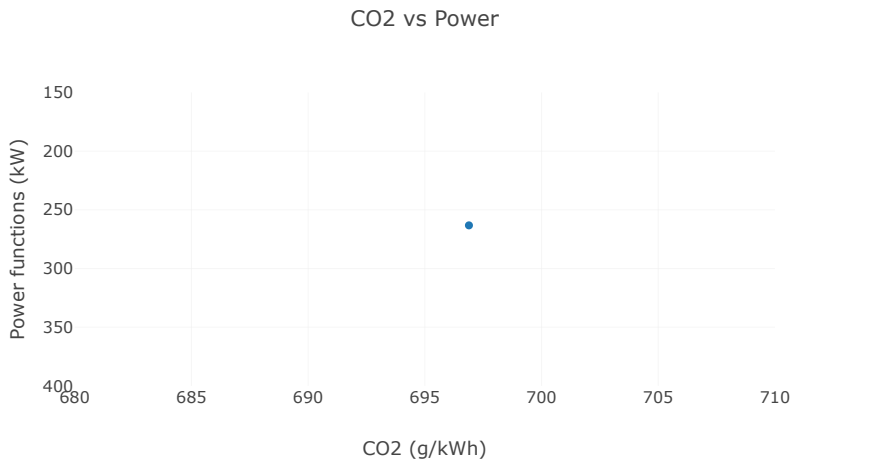
Exercise 36

Try and adjust the values for x. Can you find the optimal solution? How does it change for different models?

RPM 1000

Weighted Objective: = - 1/3 * POWfunc(1000) + 2/3 * CO2func(1000) = - 1/3 * 263.24 + 2/3 * 696.89 = 376.85

Goal Attainment: max(460 - POWfunc(1000), CO2func(1000) - 640) = max(460 - 263.24, 696.89 - 640) = 196.76



Method

Now let's solve this problem using an optimization method. The interpolated data is stored in `CO2func` and `POWfunc`.

Import libraries

```
import scipy as sp
import numpy as np
import matplotlib.pyplot as plt
```

Define variables

As before, we don't need to specify our variable x itself as defined in (16). However, this optimization method requires an initial guess. An arbitrary value is chosen here:

```
1 x0 = np.array(1200)
```

run run all add cell clear

Define objective function

Let's define the objective function for each of the three models

Weighted objective function

The objective function was defined in (26), which gives:

```
1 def weighted_obj(x):
2     delta_p = 1/3
3     delta_c = 1 - delta_p
4     return -delta_p * POWfunc(x) + delta_c * CO2func(x)
```

run run all add cell clear

Goal attainment

The objective function was defined in (27), which gives:

```
def goal_attainment(x):
    Pt = 460
    Ct = 640
    return max(Pt - POWfunc(x),CO2func(x)-Ct)
```

run run all add cell clear

Pareto front

For the pareto front, the objective functions needed to be normalized as defiend by (29):

```
def POWfunc_normalized(x):
    return (POWfunc(x) - 150)/(400 - 150)

def CO2func_normalized(x):
    return (CO2func(x) - 680)/(710 - 680)
```

run run all add cell clear

The objective function was defined in (28), which gives:

```
def weighted_obj_pareto(x):
    return -delta_p * POWfunc_normalized(x) + delta_c * CO2func_normalized(x)
```

run run all add cell clear

in which `delta_p` and `delta_c` are defined when solving this problem

Define bounds

The one single bound was defined in (30) and results in:

```
bounds = [[800,1800]]
```

run | run all | add cell | clear

Solve the problem

Now let's solve the problem for each of the three models. We're gonna time them to see how long the analysis takes. Please note that when running this in the browser, the `CPU times` gives invalid values.

Weighted objective function

```
%%time
result = sp.optimize.minimize(fun = weighted_obj, x0 = x0, bounds = bounds)
print(result)
```

run | run all | add cell | clear

Goal attainment

```
%%time
result2 = sp.optimize.minimize(fun = goal_attainment, x0 = x0, bounds=bounds)
print(result2)
```

run run all add cell clear

Pareto front

For the pareto front we need to solve this problem for a collection of weights. The results are stored in a list

```
%%time
x_pareto_opt =[]
delta_p_list = np.linspace(0,1,101)
delta_c_list = 1 - delta_p_list
for i in range(101):
    delta_p = delta_p_list[i]
    delta_c = delta_c_list[i]
    result_i = sp.optimize.minimize(fun = weighted_obj_pareto,x0=x0,bounds=bounds)
    x_pareto_opt.append(result_i.x[0])
```

run run all add cell clear

Now the pareto front can be plotted by evaluating the objectives functions for our collection of optimum values for x:

```
P_pareto_opt = POWfunc(x_pareto_opt)
C_pareto_opt = CO2func(x_pareto_opt)

plt.figure()
plt.plot(C_pareto_opt,P_pareto_opt,'x')
for i in range(101):
    if i%10 == 0:
        plt.annotate(f"($\delta_i,\delta_j$)=({round(delta_p_list[i],1)},{round(delta_c_list[i],1)})
plt.ylabel('Power (kW)')
plt.xlabel('CO$_2$ (g/kWh)')
plt.title('Pareto Front')
ax = plt.gca()
ax.invert_yaxis()
ax.spines['right'].set_color('none')
```

run run all add cell clear

Exercise 37

Exercise 38

Click 🚀 –> Live Code and answer the following question:

Class exercise: Ice cream cone

> ***Added in version v2024.3.0:*** After class March 10th
>
> Solutions in text and downloads

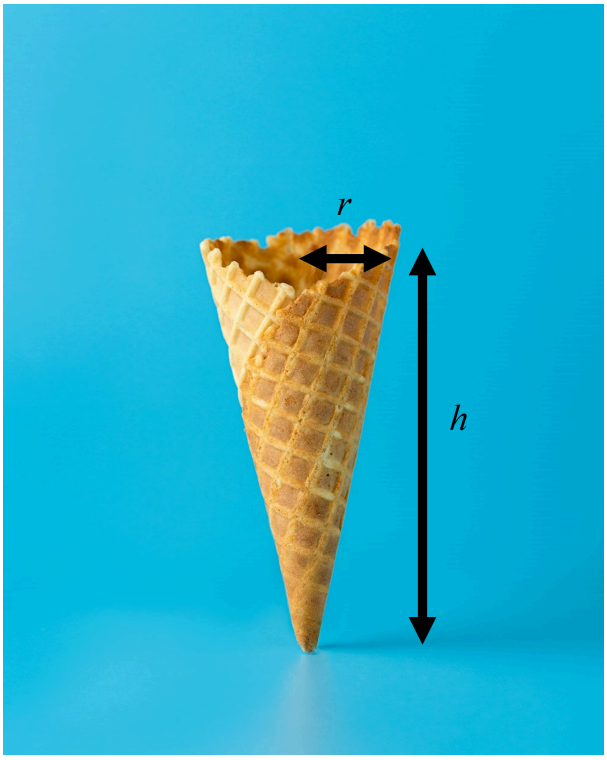


Fig. 11 ice cone

We want to minimize the surface area of a cone, while not making the gap on the top too big (in terms of area). Let's define this as a multi-objective optimization problem with

1. the first objective to minimize the ice cream cone area $\pi r\sqrt{r^2+h^2}$
2. the second objective to minimize the ice cream cone area + the area of the gap $\pi r\sqrt{r^2+h^2}+\pi r^2$.

The total volume inside the cone must be at least 200.

Find the pareto front using normalized objective functions

Exercise 39 (Model)

Write out the model of this problem.

Solution to Exercise 39 (Model)

$$\min_{r,h}\left(\delta_{Area}\cdot\frac{Area\,(r,h)-\min_{r,h}Area\,(r,h)}{\max_{r,h}Area\,(r,h)-\min_{r,h}Area\,(r,h)}+\delta_{Total}\cdot\frac{Total\,(r,}{\max_{r,h}Tota}\right.$$

$$\text{with } Area\,(r,h)=\pi r\sqrt{r^2+h^2}$$

$$\text{and } Total\,(r,h)=Area\,(r,h)+\frac{4}{3}\pi r^2$$

$$\text{such that}\frac{1}{3}\pi r^2 h$$

The minimum and maximum $Area$ and Gap could be find by solving two single-objective problems:

$$\min_{r,h}\pi r\sqrt{r^2+h^2}$$

$$\text{such that } \frac{1}{3}\pi r^2 h\geq 200$$

```
import scipy as sp
import numpy as np
```

```
def Area(x):
    return np.pi* x[0] * np.sqrt(x[0]**2 + x[1]**2)
def Total(x):
    return np.pi* x[0] * np.sqrt(x[0]**2 + x[1]**2) + 4/3 * np.pi*x[0]**2

x0 = np.array([5,10])
def nonlinconfun(x):
    c = np.pi / 3 * x[0]**2 * x[1]
    return c

cons = sp.optimize.NonlinearConstraint(nonlinconfun, 200, np.inf)

bounds = [[0, np.inf],
          [0, np.inf]]
```

```
result = sp.optimize.minimize(fun = Area, x0 = x0, bounds = bounds, constraints
print(result)
print(Area(result.x))
print(Total(result.x))
```

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 143.23025086897755
       x: [ 5.131e+00  7.256e+00]
     nit: 6
     jac: [ 3.722e+01  1.316e+01]
    nfev: 18
    njev: 6
143.23025086897755
253.48896806009228
```

$$\min_{r,h} \frac{4}{3}\pi r^3$$

$$\text{such that } \frac{1}{3}\pi r^2 h \geq 200$$

```
result = sp.optimize.minimize(fun = Total, x0 = x0, bounds = bounds, constraints
print(result)
print(Area(result.x))
print(Total(result.x))
```

```
 message: Optimization terminated successfully
 success: True
  status: 0
     fun: 224.73872628150949
       x: [ 3.840e+00  1.296e+01]
     nit: 10
     jac: [ 7.804e+01  1.157e+01]
    nfev: 27
    njev: 9
162.98691601464716
224.73872628150949
```

Leading to:

- $\min_{r,h} Area\,(r,h) \approx 143$
- $\max_{r,h} Area\,(r,h) \approx 162$
- $\min_{r,h} Total\,(r,h) \approx 224$
- $\max_{r,h} Total\,(r,h) \approx 253$

Exercise 40 (Method)

Now let's solve this problem using an optimization method.

Click 🚀 –> **Live Code** to activate live coding on this page!

```
import scipy as sp
import numpy as np
```

```
#YOUR CODE HERE
```

Solution to Exercise 40 (Method)

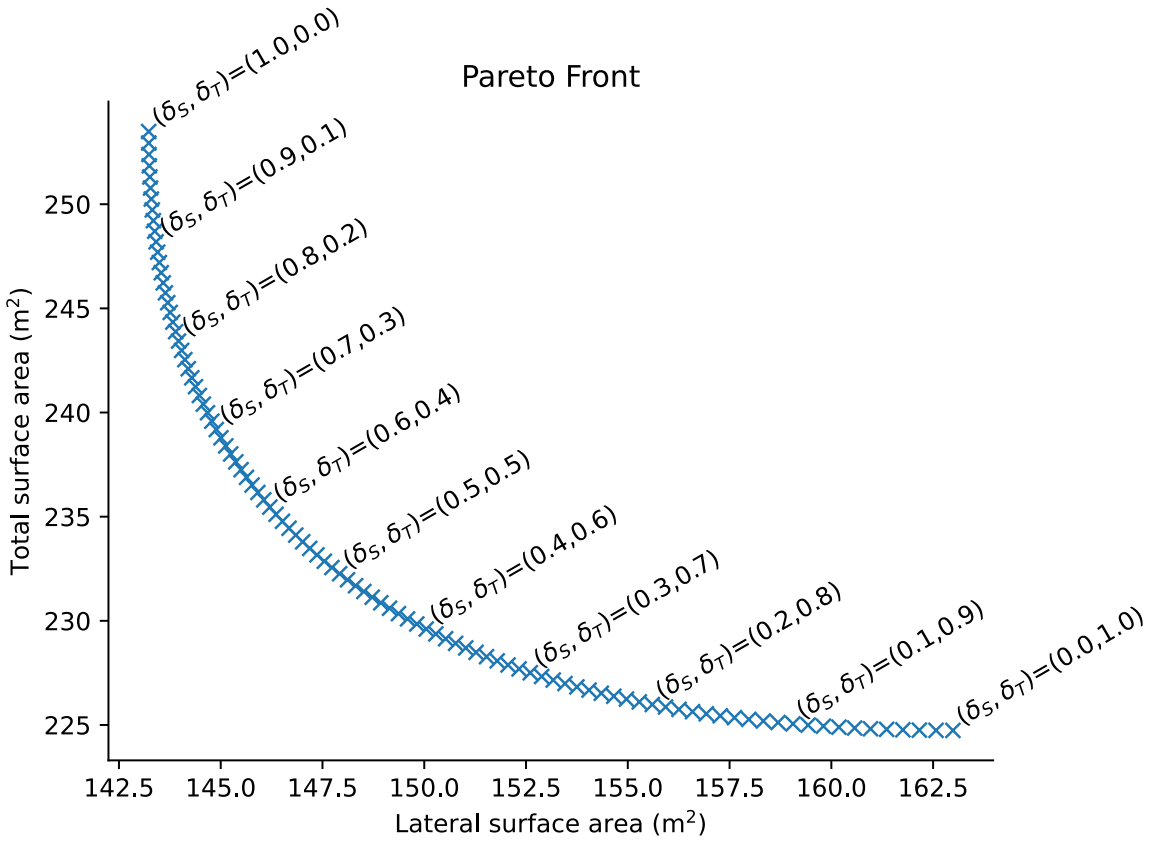
```
def weighted_obj_pareto(x):
    return delta_Area * ( Area(x) -143) / ( 162 - 143) + delta_Total * ( Total(x
```

```
%%time
x_pareto_opt =[]
delta_Area_list = np.linspace(0,1,101)
delta_Area_opt = []
delta_Total_list = 1 - delta_Area_list
delta_Total_opt = []
for i in range(101):
    delta_Area = delta_Area_list[i]
    delta_Total = delta_Total_list[i]
    result_i = sp.optimize.minimize(fun = weighted_obj_pareto,x0=x0,bounds=bound
    if result_i.success:
        x_pareto_opt.append(result_i.x)
        delta_Area_opt.append(delta_Area)
        delta_Total_opt.append(delta_Total)
    else:
        print(result_i.message)
```

```
CPU times: total: 234 ms
Wall time: 227 ms
```

```
import matplotlib.pyplot as plt
%config InlineBackend.figure_formats = ['svg']
```

```
Area_pareto_opt = []
Total_pareto_opt = []
for i in range(len(x_pareto_opt)):
    Area_pareto_opt.append(Area(x_pareto_opt[i]))
    Total_pareto_opt.append(Total(x_pareto_opt[i]))

plt.figure()
plt.plot(Area_pareto_opt,Total_pareto_opt,'x')
for i in range(len(x_pareto_opt)):
    if i%10 == 0:
        plt.annotate(f"($\delta_S,\delta_T$)=({round(delta_Area_opt[i],1)},{rou
plt.xlabel('Lateral surface area (m$^2$)')
plt.ylabel('Total surface area (m$^2$)')
plt.title('Pareto Front')
ax = plt.gca()
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
```


Pareto Front
Total surface area (m^2)
Lateral surface area (m^2)
250
245
240
235
230
225
142.5
145.0
147.5
150.0
152.5
155.0
157.5
160.0
162.5
(δ_S, δ_T)=(1.0,0.0)
(δ_S, δ_T)=(0.9,0.1)
(δ_S, δ_T)=(0.8,0.2)
(δ_S, δ_T)=(0.7,0.3)
(δ_S, δ_T)=(0.6,0.4)
(δ_S, δ_T)=(0.5,0.5)
(δ_S, δ_T)=(0.4,0.6)
(δ_S, δ_T)=(0.3,0.7)
(δ_S, δ_T)=(0.2,0.8)
(δ_S, δ_T)=(0.1,0.9)
(δ_S, δ_T)=(0.0,1.0)

Metaheuristics using scipy and pymoo

Contents

In this chapter, we'll cover how to solve all optimization problem using metaheuristics, which are methods. Depending on the actual metaheuristic different models can be solved, but in general metaheuristics are more versatile than the gradient-based methods.

Model

The two method covered in this chapter will be applied on our most general nonlinear constrained optimization problem as defined before in (15).

Method

Two different methods will be treated: `scipy.optimize.differential_evolution` and the genetic algorithm from `pymoo.optimize`. There are many different methods available in different programming languagues, these two methods are only an example of what you could use.

Scipy

`scipy` has implemented, among others, the differential evolution method [Storn and Price, 1997]. The documentation of this function is available here: https://docs.scipy.org/doc/scipy-1.15.0/reference/generated/scipy.optimize.differential_evolution.html [The SciPy community, 2024]. In this course we'll cover only the relevant parts.

We need to run at least `scipy.optimize.differential_evolution(fun, x0, bounds, constraints, integrality, strategy, popsize, mutation, recombination, ...)` with:

- `fun`, the function representing the objective function $f(x)$ to be minimized. `fun` is a callable. The `scipy.optimize.minimize` function takes care of defining and inputting our design variable x.
- `x0`, the initial guess for our design variable x. It needs to be a `ndarray` with length n
- `Bounds`: A sequence of i `(min, max)` pairs for each element in x, defining the minimum x_i^l and maximum values x_i^u of that decision variable. For this function `None` cannot be used to specify no bound. The values `min` and `max` need to be finite.
- `constraints`, a single or a list of constraint objective defined in the same way as in nonlinear constrained optimization with:
  - `scipy.optimize.LinearConstraint`
  - `scipy.optimize.NonlinearConstraint`
- `integrality`, an `ndarray` specifying whether part of the n design variables is constrained to integer values.
- `strategy`, optional `string` argument which allows you to select another differential evolution strategy. The default is `best1bin`
- `popsize`, an optional `integer` argument which allows you to set the total population size.
- `mutation`, an optional `float` or `tuple(float,float)` argument which allows you to set the mutation constant.
- `recombination`, an optional `float` argument which allows you to set the recombinatino constant.

Please note that are even more options to adapt the optimization algorithm.

The function `scipy.optimize.differential_evolution` outputs an object `scipy.optimize.OptimizeResult` similar as `scipy.optimize.minimize` explained for unconstrained optimization.

Exercise 41

pymoo

`pymoo` has implemented, among others, the genetic algorithm. The documentation of this function, although very limited, is available here: https://pymoo.org/ [Blank and Deb, 2020]. Again, we'll cover only the relevant parts. Make sure you install pymoo as explained here in the section Add other packages and in the documentation of pymoo.

The main function we need is `pymoo.minimize(problem, algorithm, ...)` with:

- `problem`, pymoo object containing the problem
- `algorithm`, pymoo object containing the method

This results in an object `pymoo.core.result.Result` with:

- `Result.x`, the solution found
- `Result.F`, value of the objective function for the solution
- `Result.G`, value of the constraint functions for the solution
- `Result.time`, the time required to run the algorithm

The problem needs to be defined in an object, therefore we'll use `pymoo.problems.functional(n_var, objs, constr_ieq=[], constr_eq=[], xl, xu, ...)` with:

- `n_var`, the number of design variables n, this needs to be an `int`.

- `objs`, the objective function f to be minimized, this needs to be a `callable`.
- `constr_ieq`, list of m inquality constraint functions g, this needs to be a list of `callable`
- `constr_eq`, list of n equality constraint functions h, this needs to be a list of `callable`.
- `xl`, `Float` or `ndarray` of length n representing the lower bounds of the design variables.
- `xu`, `Float` or `ndarray` of length n representing the upper bounds of the design variables.

As a method, we'll use the genetic algorithm. This is stored in the object `pymoo.algorithms.soo.nonconvex.ga(pop_size=100, sampling=<pymoo.operators.sampling.rnd.FloatRandomSampling object>, selection=<pymoo.operators.selection.tournament.TournamentSelection object>, crossover=<pymoo.operators.crossover.sbx.SBX object>, mutation=<pymoo.operators.mutation.pm.PM object>, survival=<pymoo.algorithms.soo.nonconvex.ga.FitnessSurvival object>, ...)` with:

- `pop_size`, `int` defining size of the population
- `sampling`, pymoo object defining how sampling should happen. If you want to solve integer problems, input must be `pymoo.operators.sampling.rnd.IntegerRandomSampling()`
- `selection`, pymoo object defining how selection should happen
- `crossover`, pymoo object defining how crossover should happen. If you want to solve integer problems, input must be `pymoo.operators.crossover.sbx.SBX(repair=pymoo.operators.repair.rounding.RoundingRepair())`
- `mutation`, pymoo object defining how mutation should happen. If you want to solve integer problems, input must be `pymoo.operators.mutation.pm.PM(repair=pymoo.operators.repair.rounding.RoundingRepair())`
- `survival`, pymoo object defining how survival should happen

Error

Note that pymoo doesn't work in the interactive coding functionality of this website. Please install pymoo on your computer to use it locally.

Exercise 42

Homework exercise: Construction site

Contents

Click 🚀 –> **Live Code** to activate live coding on this page! Unfortunately, the `pymoo` package doesn't work yet in this book. Download this page to run the cells on your own computer.

Problem

As managers of a construction site, we have to decide the required equipment to move a given volume of excavated soil. In order to achieve the deadline for this working unit, we need to guarantee an averaged efficiency of $2700 \ \frac{\mathrm{m}^3}{h}$ during one month.

We have our own equipment, and also we can sub-contract another (just one) company. The efficiency of each equipment and the cost are given in the table:

Own equip.		Company 1		Company 2	
Eff. $\left(\frac{\mathrm{m}^3}{h}\right)$	Cost $\left(\frac{€}{h}\right)$	Eff. $\left(\frac{\mathrm{m}^3}{h}\right)$	Cost $\left(\frac{€}{h}\right)$	Eff. $\left(\frac{\mathrm{m}^3}{h}\right)$	Cost $\left(\frac{€}{h}\right)$
200	500	470	4.000	640	5.400
240	800	700	5.700	730	5.500
265	1.000	800	6.500	775	6.800
330	1.500				

Then, we can use part or all of our own equipment with some of the equipment options provided by another company.

What is the optimal equipment combination that minimizes the cost?

Model

We'll define the model as follows:

- Design variables: Whether or not to use each type of equipment
- Objective function: sum of the costs of the selected equipment
- Constraint functions: make use of one sub-contractor and create an efficiency of at least $2700\ \frac{\mathrm{m}^3}{h}$

Design variables

Let's start with our design variables. let's define them as a list of integer values either being 0 or 1, which selects the equipment type.

$$x = \begin{bmatrix} x_{\text{First item of own equipment}} \\ x_{\text{Second item of own equipment}} \\ x_{\text{Third item of own equipment}} \\ x_{\text{Fourth item of own equipment}} \\ x_{\text{First item of company 1}} \\ x_{\text{Second item of company 1}} \\ x_{\text{Third item of company 1}} \\ x_{\text{First item of company 2}} \\ x_{\text{Second item of company 2}} \\ x_{\text{Third item of company 2}} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \end{bmatrix}$$

The design variables either being 0 or 1 can be defined mathemetically:

$$x_i \in \{0, 1\}\, i = 1, 2, \ldots, 10 \tag{32}$$

Objective function

Now we can define the objective function as the product of the dimension to represent $\min_x f(x)$ in (4):

$$f(x) = 500 \cdot x_1 + 800 \cdot x_2 + 1000 \cdot x_3 + 1500 \cdot x_4 + 4000 \cdot x_5 + 5700 \cdot x_6 + 650 \tag{33}$$

As this is a linear relation, this can be converted to matrix notation. In case of the design variables defined as in (5) in the form $\min_x c^T x$ with c:

$$c = \begin{bmatrix} 500 & 800 & 1000 & 1500 & 4000 & 5700 & 6500 & 5400 & 5500 & 6800 \end{bmatrix}^T \tag{34}$$

Constraint function

We need to define two constraint functions, let's start with the inequality constraint functions as $g(x) \leq 0$ as defined in (4):

$$g(x) = -200 \cdot x_1 - 240 \cdot x_2 - 265 \cdot x_3 - 330 \cdot x_4 - 470 \cdot x_5 - 700 \cdot x_6 - 800 \cdot x \tag{35}$$

As this is a linear relation, this can be converted to matrix notation as defined in (5) in the form $A_{ub}x \leq b_{ub}$:

$$\begin{bmatrix}-200 & -240 & -265 & -330 & -470 & -700 & -800 & -640 & -730 & -775\end{bmatrix} x \tag{36}$$

Let's add the equality constraint function. As soon as we hire one piece of equipment this should be violated. It is defined as $h\left(x\right) \leq 0$ as in (15):

$$h\left(x\right) = \max\left(x_5, x_6, x_7\right) \cdot \max\left(x_8, x_9, x_{10}\right) \tag{37}$$

Clearly, this function is not linear, so we cannot define it in matrix formulation.

Method

Now let's solve this problem using both `scipy` and `pymoo`.

Import libraries

```
import numpy as np
```

SciPy

For SciPy this is easy:

```
import scipy as sp
```

pymoo

For pymoo we need to import a lot more and be more specific:

```
from pymoo.problems.functional import FunctionalProblem as FunctionalProblem
from pymoo.algorithms.soo.nonconvex.ga import GA
from pymoo.optimize import minimize
from pymoo.operators.sampling.rnd import IntegerRandomSampling
from pymoo.operators.crossover.sbx import SBX
from pymoo.operators.mutation.pm import PM
from pymoo.operators.repair.rounding import RoundingRepair
```

Error

Note that pymoo doesn't work in the interactive coding functionality of this website. Please download this notebook to use it locally.

Define variables

This is done differently in SciPy and pymoo.

SciPy

As before, we don't need to specify our variable x itself as defined in (31). However, we do need to specify that we have integers (specifically only 0 and 1 which will be covered by the bounds) In SciPy we use an array of booleans to specify which design variables are integers:

```
integers=np.array([True,True,True,True,True,True,True,True,True,True])
```

pymoo

In pymoo we don't have to specify that have integers, but we have to adapt all of our functions later on so that the outputs are integers. Furthermore, we must specify explicitly how many design variables we have:

```
n_var = 10
```

Define bounds

Now let's continue with the bounds, specified by (31) too:

SciPy

The bounds are defined as before:

```

```

```
bounds = [[0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1],
          [0,1]]
```

pymoo

In pymoo, the bounds are defined as separate arrays:

```
xl = np.array([0,0,0,0,0,0,0,0,0,0])
xu = np.array([1,1,1,1,1,1,1,1,1,1])
```

Define objective function

Let's define the objective function as defined in (34).

```
def obj(x):
    return np.array([500, 800, 1000, 1500, 4000, 5700, 6500, 5400, 5500, 6800])@x
```

Define constraint function

Let's define the constraint function as defined in (36) and (37)

```
g = np.array([-200, -240, -265, -330, -470, -700, -800, -640, -730, -775])

def nonlinconfun(x):
    return max(x[4:7]) * max(x[7:])
```

SciPy

In SciPy we need to store the functions in a scipy-object including the bounds

```
lincon = sp.optimize.LinearConstraint(g, lb=-np.inf, ub=-2700)
nonlincon = sp.optimize.NonlinearConstraint(nonlinconfun, 0, 0)
```

pymoo

In pymoo the functions can be inserted directly in the problem object later on. However, this requires a function for the linear constraints instead of just the array

```
def linconfun(x):
    return g@x + 2700
```

Solve the problem

Now let's solve the problem using both SciPy and pymoo.

SciPy

```
result_scipy = sp.optimize.differential_evolution(func = obj,bounds = bounds,constraints=
print(result_scipy)
```

```
             message: Optimization terminated successfully.
             success: True
                 fun: 19000.0
                   x: [ 1.000e+00  1.000e+00  0.000e+00  1.000e+00
                        1.000e+00  1.000e+00  1.000e+00  0.000e+00
                        0.000e+00  0.000e+00]
                 nit: 34
                nfev: 1226
          population: [[ 1.000e+00  1.000e+00 ...  0.000e+00  0.000e+00]
                       [ 1.000e+00  1.000e+00 ...  0.000e+00  0.000e+00]
                       ...
                       [ 1.000e+00  1.000e+00 ...  0.000e+00  0.000e+00]
                       [ 1.000e+00  1.000e+00 ...  0.000e+00  0.000e+00]]
 population_energies: [ 1.900e+04  1.900e+04 ...  1.900e+04  1.900e+04]
              constr: [array([ 0.000e+00]), array([ 0.000e+00])]
    constr_violation: 0.0
               maxcv: 0.0
```

Pymoo

In pymoo we have to define the problem and algorithm as objects, and call them from the `pymoo.minimize` function. For the `GA` functions, we repair some stuff so that we only deal with integer values

```
```

```
problem = FunctionalProblem(n_var=n_var, objs=obj, constr_ieq=[linconfun],constr_eq = [no
algorithm = GA(sampling=IntegerRandomSampling(),crossover=SBX(repair=RoundingRepair()),mu
```

Now we can solve the problem

```
result_pymoo = minimize(problem, algorithm)
print(result_pymoo.X)
print(result_pymoo.F)
```

```
[0 1 1 1 1 1 1 0 0 0]
[19500.]
```

Exercise 43

Class exercise: choose yourself!

Solve one of the previous class exercises using metaheuristics!

- Class exercise: Furniture manufacturing
- Class exercise: Suspended beams
- Class exercise: Ice cream cone

Exercise 44 (Method)

Let's solve this problem using a metaheuristic!

Click 🚀 –> **Live Code** to activate live coding on this page!

```
import scipy as sp
import numpy as np
```

```
#YOUR CODE HERE
```

Stochastic workshop

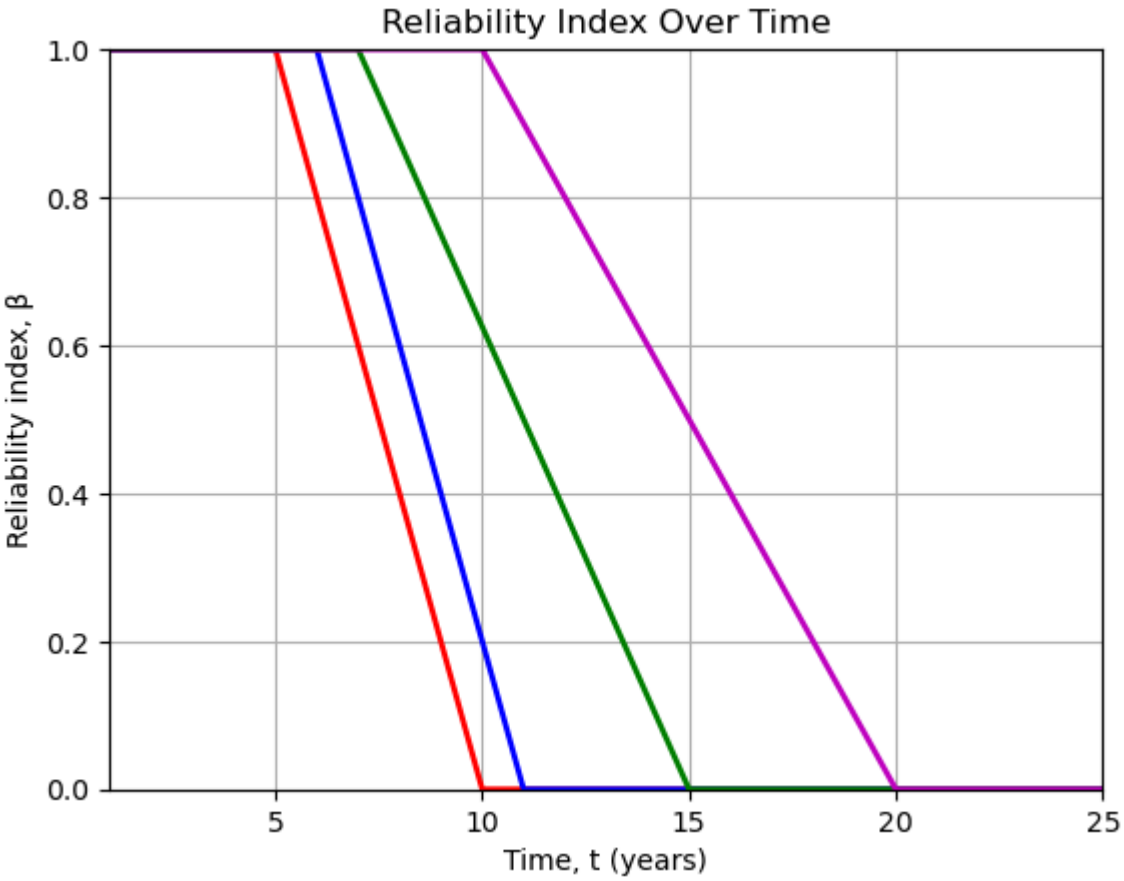
Deterministic

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution
import scipy as sp

def reliability(beta0_b, d_b, L_b, Thoriz):
    # --- Reliability estimation
    # beta_bt: reliability curve over time for bridge b
    # beta0_b: Initial reliability of bridge b at t=0
    # d_b, L_b: when the degradation starts and reaches beta=0 for each bridge
    # Thoriz: maximum time studied
    nb = len(beta0_b)
    beta_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        #horziontal branch
        timesteps = int(np.floor(d_b[b]))
        beta_bt[b, :timesteps] = beta0_b[b]

        #degradation till L_b
        servicelife = int(np.floor(L_b[b]))
        t = np.arange(timesteps + 1, servicelife + 1)
        beta_bt[b, timesteps:servicelife] = beta0_b[b] / (L_b[b] - d_b[b]) * (L_b[b] - t)

    return beta_bt

def MaintenanceApplication(beta_bt, X_bt, Thoriz):
    # --- Given a maintenance strategy given by X_bt, where 1 means that bridge
    # b undergoes maintenance at time t, this function gives the updated
    # reliability profile
    updatedbeta_bt = beta_bt.copy()
    InterventionTimes = np.where(np.sum(X_bt, axis=0) > 0)[0]

    for t in InterventionTimes:
        intervbrid = np.where(X_bt[:, t] == 1)[0]
        timecells = Thoriz - t - 1
        updatedbeta_bt[intervbrid, t+1:Thoriz] = beta_bt[intervbrid, :timecells]

    return updatedbeta_bt

def Frequency2Schedule(y_b, nb, nt, Thoriz):
    # Transforms a vector indicating the frequency to a schedule matrix [0/1]
    freq_b = y_b.reshape((nt, nb)).T
    cumFreq_b = cumulateFreq(freq_b)
    X_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        idx = cumFreq_b[b, cumFreq_b[b, :] != 0].astype(int)
        idx = idx[idx <= Thoriz]
        if len(idx) > 0:
            X_bt[b, idx - 1] = 1

    return X_bt

def cumulateFreq(freq_b):
    # creates the cumulative frequency
    nb, nt = freq_b.shape
    cumFreq_b = np.zeros((nb, nt))

    for b in range(nb):
        f_b = freq_b[b, freq_b[b, :] != 0]
        cumFreq_b[b, :len(f_b)] = np.cumsum(f_b)
```

```
    return cumFreq_b

def CountNinterv(y_b, nt, nb):
    # from vector y_b, it computes the number of interventions per bridge
    y_b01 = y_b != 0
    freq_b = y_b01.reshape((nt, nb)).T
    ninterv_b = np.sum(freq_b, axis=1)
    return ninterv_b
```

```
Thoriz = 25
# for each bridge b:  beta0 d(years) L(years) C(Meuros)
data = np.array([
    [1, 5, 10, 0.6],
    [1, 6, 11, 0.8],
    [1, 7, 15, 1.0],
    [1, 10, 20, 0.7]
])
beta0_b, d_b, L_b, C_b = data[:, 0], data[:, 1], data[:, 2], data[:, 3]

# Annual resources (M euro)
R = 1.4

# Initial reliability estimation
nb = len(beta0_b)
beta_bt0 = reliability(beta0_b, d_b, L_b, Thoriz)

plt.figure()
col = ['r', 'b', 'g', 'm']
for b in range(nb):
    plt.plot(range(1, Thoriz + 1), beta_bt0[b, :], f'-{col[b]}', linewidth=2)
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()

# Deterministic Optimization
# Variables ->  e.g., freq=[10 2 0 0 ; 5, 5 0 0; 0 0 0 0];

maxInter_b = np.ceil(Thoriz / L_b).astype(int) + 1 #max number of interventions expected
nt = max(maxInter_b) #to dimension the matrix, max interventions experienced by the worst
nvars = nb * nt #number of design variables
integers = np.array([True] * nvars) #all variables are integers

#  Range of definition: [0,max number of years with no interventions==L_b]

lb = np.zeros(nvars)
ub = np.zeros(nvars)
pos = 0
for b in range(nb):
    # This guarantees interventions freq. within the service life (if conducted)
    ub[pos:pos + maxInter_b[b]] = L_b[b] - 1
    # this forces zeros after the max number of interventions expected
    if maxInter_b[b] < nb:
        ub[pos + maxInter_b[b]:pos + nb - 1] = 0
    pos += nt

def nonlcon(y_b):
    #  From frequency to a schedule matrix [0/1]
    X_bt = Frequency2Schedule(y_b, nb, nt, Thoriz)

    # Constraint 1 >>> the last intervention is within Thoriz --> LastInterv<Thoriz --> L
    lastInterv_b = np.max(cumulateFreq(y_b.reshape((nt, nb)).T), axis=1)
    c1 = lastInterv_b - Thoriz - 1

    # Constraint 2 >>> C_b*X_bt <= R   for all t
    c2 = np.sum(C_b[:, None] * X_bt, axis=0) - R

    # Constraint 3 >>> beta_bt>0  --> sum(sum(beta_bt))<=0
    beta_bt = MaintenanceApplication(beta_bt0, X_bt, Thoriz)
```

```
    c3 = np.array([np.sum(beta_bt == 0)])

    c = np.concatenate([c1, c2, c3])
    return c

cons = sp.optimize.NonlinearConstraint(nonlcon, np.array([-np.inf] * (nb + Thoriz + 1)),

def fun(y_b):
    #  minimize cost trying to go for the largest time between interventions (== max y_b)
    ninterv_b = CountNinterv(y_b, nt, nb)
    return np.dot(C_b, ninterv_b)

bounds = [[lb[i], ub[i]] for i in range(nvars)]
```




```
result = differential_evolution(fun, bounds, constraints=cons, integrality=integers, disp
print(result)
```

```
differential_evolution step 1: f(x)= 11.0
differential_evolution step 2: f(x)= 11.0
differential_evolution step 3: f(x)= 8.200000000000001
differential_evolution step 4: f(x)= 8.200000000000001
differential_evolution step 5: f(x)= 8.200000000000001
differential_evolution step 6: f(x)= 8.200000000000001
differential_evolution step 7: f(x)= 8.200000000000001
differential_evolution step 8: f(x)= 8.200000000000001
differential_evolution step 9: f(x)= 8.200000000000001
differential_evolution step 10: f(x)= 8.200000000000001
differential_evolution step 11: f(x)= 8.200000000000001
differential_evolution step 12: f(x)= 8.200000000000001
differential_evolution step 13: f(x)= 8.200000000000001
differential_evolution step 14: f(x)= 8.200000000000001
differential_evolution step 15: f(x)= 8.200000000000001
differential_evolution step 16: f(x)= 8.200000000000001
differential_evolution step 17: f(x)= 8.200000000000001
differential_evolution step 18: f(x)= 8.200000000000001
differential_evolution step 19: f(x)= 8.200000000000001
differential_evolution step 20: f(x)= 8.200000000000001
differential_evolution step 21: f(x)= 8.200000000000001
differential_evolution step 22: f(x)= 8.200000000000001
differential_evolution step 23: f(x)= 8.200000000000001
differential_evolution step 24: f(x)= 8.200000000000001
differential_evolution step 25: f(x)= 8.200000000000001
differential_evolution step 26: f(x)= 8.200000000000001
differential_evolution step 27: f(x)= 8.200000000000001
differential_evolution step 28: f(x)= 8.200000000000001
differential_evolution step 29: f(x)= 8.200000000000001
differential_evolution step 30: f(x)= 8.200000000000001
differential_evolution step 31: f(x)= 8.200000000000001
differential_evolution step 32: f(x)= 8.200000000000001
differential_evolution step 33: f(x)= 8.200000000000001
differential_evolution step 34: f(x)= 8.200000000000001
differential_evolution step 35: f(x)= 8.200000000000001
differential_evolution step 36: f(x)= 8.200000000000001
differential_evolution step 37: f(x)= 8.200000000000001
differential_evolution step 38: f(x)= 8.200000000000001
differential_evolution step 39: f(x)= 8.200000000000001
differential_evolution step 40: f(x)= 8.200000000000001
differential_evolution step 41: f(x)= 7.8
differential_evolution step 42: f(x)= 7.8
differential_evolution step 43: f(x)= 7.8
differential_evolution step 44: f(x)= 7.8
differential_evolution step 45: f(x)= 7.8
differential_evolution step 46: f(x)= 7.8
differential_evolution step 47: f(x)= 7.8
differential_evolution step 48: f(x)= 7.8
differential_evolution step 49: f(x)= 7.8
differential_evolution step 50: f(x)= 7.8
differential_evolution step 51: f(x)= 7.8
differential_evolution step 52: f(x)= 7.8
differential_evolution step 53: f(x)= 7.8
differential_evolution step 54: f(x)= 7.8
differential_evolution step 55: f(x)= 7.8
differential_evolution step 56: f(x)= 7.8
differential_evolution step 57: f(x)= 7.8
differential_evolution step 58: f(x)= 7.8
differential_evolution step 59: f(x)= 7.8
differential_evolution step 60: f(x)= 7.8
differential_evolution step 61: f(x)= 7.8
differential_evolution step 62: f(x)= 7.8
differential_evolution step 63: f(x)= 7.8
```

```
differential_evolution step 64: f(x)= 7.8
differential_evolution step 65: f(x)= 7.8
differential_evolution step 66: f(x)= 7.8
differential_evolution step 67: f(x)= 7.8
differential_evolution step 68: f(x)= 7.8
differential_evolution step 69: f(x)= 7.8
differential_evolution step 70: f(x)= 7.8
differential_evolution step 71: f(x)= 7.8
differential_evolution step 72: f(x)= 7.8
differential_evolution step 73: f(x)= 7.8
differential_evolution step 74: f(x)= 7.8
differential_evolution step 75: f(x)= 7.8
differential_evolution step 76: f(x)= 7.8
differential_evolution step 77: f(x)= 7.8
differential_evolution step 78: f(x)= 7.8
differential_evolution step 79: f(x)= 7.8
differential_evolution step 80: f(x)= 7.8
differential_evolution step 81: f(x)= 7.8
differential_evolution step 82: f(x)= 7.8
differential_evolution step 83: f(x)= 7.8
differential_evolution step 84: f(x)= 7.8
differential_evolution step 85: f(x)= 7.8
differential_evolution step 86: f(x)= 7.8
differential_evolution step 87: f(x)= 7.8
differential_evolution step 88: f(x)= 7.8
differential_evolution step 89: f(x)= 7.8
differential_evolution step 90: f(x)= 7.8
differential_evolution step 91: f(x)= 7.8
differential_evolution step 92: f(x)= 7.8
differential_evolution step 93: f(x)= 7.8
differential_evolution step 94: f(x)= 7.8
differential_evolution step 95: f(x)= 7.8
differential_evolution step 96: f(x)= 7.8
differential_evolution step 97: f(x)= 7.8
differential_evolution step 98: f(x)= 7.8
differential_evolution step 99: f(x)= 7.8
differential_evolution step 100: f(x)= 7.8
differential_evolution step 101: f(x)= 7.8
differential_evolution step 102: f(x)= 7.8
differential_evolution step 103: f(x)= 7.8
differential_evolution step 104: f(x)= 7.8
differential_evolution step 105: f(x)= 7.8
differential_evolution step 106: f(x)= 7.8
differential_evolution step 107: f(x)= 7.8
differential_evolution step 108: f(x)= 7.8
differential_evolution step 109: f(x)= 7.8
differential_evolution step 110: f(x)= 7.8
differential_evolution step 111: f(x)= 7.8
differential_evolution step 112: f(x)= 7.8
differential_evolution step 113: f(x)= 7.8
differential_evolution step 114: f(x)= 7.8
differential_evolution step 115: f(x)= 7.8
differential_evolution step 116: f(x)= 7.8
differential_evolution step 117: f(x)= 7.8
differential_evolution step 118: f(x)= 7.8
differential_evolution step 119: f(x)= 7.8
differential_evolution step 120: f(x)= 7.8
differential_evolution step 121: f(x)= 7.8
differential_evolution step 122: f(x)= 7.8
differential_evolution step 123: f(x)= 7.8
differential_evolution step 124: f(x)= 7.8
differential_evolution step 125: f(x)= 7.8
differential_evolution step 126: f(x)= 7.8
differential_evolution step 127: f(x)= 7.8
```

```
differential_evolution step 128: f(x)= 7.8
differential_evolution step 129: f(x)= 7.8
differential_evolution step 130: f(x)= 7.8
differential_evolution step 131: f(x)= 7.8
differential_evolution step 132: f(x)= 7.8
differential_evolution step 133: f(x)= 7.8
differential_evolution step 134: f(x)= 7.8
differential_evolution step 135: f(x)= 7.8
differential_evolution step 136: f(x)= 7.8
differential_evolution step 137: f(x)= 7.8
differential_evolution step 138: f(x)= 7.6
differential_evolution step 139: f(x)= 7.6
differential_evolution step 140: f(x)= 7.6
differential_evolution step 141: f(x)= 7.6
differential_evolution step 142: f(x)= 7.6
differential_evolution step 143: f(x)= 7.6
differential_evolution step 144: f(x)= 7.6
differential_evolution step 145: f(x)= 7.6
differential_evolution step 146: f(x)= 7.6
differential_evolution step 147: f(x)= 7.6
differential_evolution step 148: f(x)= 7.6
differential_evolution step 149: f(x)= 7.6
differential_evolution step 150: f(x)= 7.6
differential_evolution step 151: f(x)= 7.6
differential_evolution step 152: f(x)= 7.6
differential_evolution step 153: f(x)= 7.6
differential_evolution step 154: f(x)= 7.6
differential_evolution step 155: f(x)= 7.6
differential_evolution step 156: f(x)= 7.6
differential_evolution step 157: f(x)= 7.6
differential_evolution step 158: f(x)= 7.6
differential_evolution step 159: f(x)= 7.6
differential_evolution step 160: f(x)= 7.6
differential_evolution step 161: f(x)= 7.6
differential_evolution step 162: f(x)= 7.6
differential_evolution step 163: f(x)= 7.6
differential_evolution step 164: f(x)= 7.6
differential_evolution step 165: f(x)= 7.6
differential_evolution step 166: f(x)= 7.6
differential_evolution step 167: f(x)= 7.6
differential_evolution step 168: f(x)= 7.6
differential_evolution step 169: f(x)= 7.6
differential_evolution step 170: f(x)= 7.6
differential_evolution step 171: f(x)= 7.6
differential_evolution step 172: f(x)= 7.6
differential_evolution step 173: f(x)= 7.6
differential_evolution step 174: f(x)= 7.6
differential_evolution step 175: f(x)= 7.6
differential_evolution step 176: f(x)= 7.6
differential_evolution step 177: f(x)= 7.6
differential_evolution step 178: f(x)= 7.6
differential_evolution step 179: f(x)= 7.6
differential_evolution step 180: f(x)= 7.6
differential_evolution step 181: f(x)= 7.6
differential_evolution step 182: f(x)= 7.6
differential_evolution step 183: f(x)= 7.6
differential_evolution step 184: f(x)= 7.6
differential_evolution step 185: f(x)= 7.6
differential_evolution step 186: f(x)= 7.6
differential_evolution step 187: f(x)= 7.6
differential_evolution step 188: f(x)= 7.6
differential_evolution step 189: f(x)= 7.6
differential_evolution step 190: f(x)= 7.2
differential_evolution step 191: f(x)= 7.2
```

```
differential_evolution step 192: f(x)= 7.2
differential_evolution step 193: f(x)= 7.2
differential_evolution step 194: f(x)= 7.2
differential_evolution step 195: f(x)= 7.2
differential_evolution step 196: f(x)= 7.2
differential_evolution step 197: f(x)= 7.2
differential_evolution step 198: f(x)= 7.2
differential_evolution step 199: f(x)= 7.2
differential_evolution step 200: f(x)= 7.2
differential_evolution step 201: f(x)= 7.2
differential_evolution step 202: f(x)= 7.2
differential_evolution step 203: f(x)= 7.2
differential_evolution step 204: f(x)= 7.2
differential_evolution step 205: f(x)= 7.2
differential_evolution step 206: f(x)= 7.2
differential_evolution step 207: f(x)= 7.2
differential_evolution step 208: f(x)= 7.2
differential_evolution step 209: f(x)= 7.2
differential_evolution step 210: f(x)= 7.2
differential_evolution step 211: f(x)= 7.2
differential_evolution step 212: f(x)= 7.2
differential_evolution step 213: f(x)= 7.2
differential_evolution step 214: f(x)= 7.2
differential_evolution step 215: f(x)= 7.2
differential_evolution step 216: f(x)= 7.2
differential_evolution step 217: f(x)= 7.2
differential_evolution step 218: f(x)= 7.2
differential_evolution step 219: f(x)= 7.2
differential_evolution step 220: f(x)= 7.2
differential_evolution step 221: f(x)= 7.2
differential_evolution step 222: f(x)= 7.2
differential_evolution step 223: f(x)= 7.2
differential_evolution step 224: f(x)= 7.2
differential_evolution step 225: f(x)= 7.2
differential_evolution step 226: f(x)= 7.2
differential_evolution step 227: f(x)= 7.2
differential_evolution step 228: f(x)= 7.2
differential_evolution step 229: f(x)= 7.2
differential_evolution step 230: f(x)= 7.2
differential_evolution step 231: f(x)= 7.2
differential_evolution step 232: f(x)= 7.2
differential_evolution step 233: f(x)= 7.2
differential_evolution step 234: f(x)= 7.2
differential_evolution step 235: f(x)= 7.2
differential_evolution step 236: f(x)= 7.2
differential_evolution step 237: f(x)= 7.2
differential_evolution step 238: f(x)= 7.2
differential_evolution step 239: f(x)= 7.2
differential_evolution step 240: f(x)= 7.2
differential_evolution step 241: f(x)= 7.2
differential_evolution step 242: f(x)= 7.2
differential_evolution step 243: f(x)= 7.2
differential_evolution step 244: f(x)= 7.2
differential_evolution step 245: f(x)= 7.2
differential_evolution step 246: f(x)= 7.2
differential_evolution step 247: f(x)= 7.2
differential_evolution step 248: f(x)= 7.2
differential_evolution step 249: f(x)= 7.2
differential_evolution step 250: f(x)= 7.2
differential_evolution step 251: f(x)= 7.2
differential_evolution step 252: f(x)= 7.2
differential_evolution step 253: f(x)= 7.2
differential_evolution step 254: f(x)= 7.2
differential_evolution step 255: f(x)= 7.2
```

```
differential_evolution step 256: f(x)= 6.6
differential_evolution step 257: f(x)= 6.6
differential_evolution step 258: f(x)= 6.6
differential_evolution step 259: f(x)= 6.6
differential_evolution step 260: f(x)= 6.6
differential_evolution step 261: f(x)= 6.4
differential_evolution step 262: f(x)= 6.4
differential_evolution step 263: f(x)= 6.4
differential_evolution step 264: f(x)= 6.4
differential_evolution step 265: f(x)= 6.4
differential_evolution step 266: f(x)= 6.4
differential_evolution step 267: f(x)= 6.4
differential_evolution step 268: f(x)= 6.4
differential_evolution step 269: f(x)= 6.4
differential_evolution step 270: f(x)= 6.4
differential_evolution step 271: f(x)= 6.4
differential_evolution step 272: f(x)= 5.9
differential_evolution step 273: f(x)= 5.9
differential_evolution step 274: f(x)= 5.9
differential_evolution step 275: f(x)= 5.9
differential_evolution step 276: f(x)= 5.9
differential_evolution step 277: f(x)= 5.9
differential_evolution step 278: f(x)= 5.9
differential_evolution step 279: f(x)= 5.9
differential_evolution step 280: f(x)= 5.9
differential_evolution step 281: f(x)= 5.9
differential_evolution step 282: f(x)= 5.9
differential_evolution step 283: f(x)= 5.9
differential_evolution step 284: f(x)= 5.9
differential_evolution step 285: f(x)= 5.9
differential_evolution step 286: f(x)= 5.9
differential_evolution step 287: f(x)= 5.9
differential_evolution step 288: f(x)= 5.9
differential_evolution step 289: f(x)= 5.9
differential_evolution step 290: f(x)= 5.9
differential_evolution step 291: f(x)= 5.9
differential_evolution step 292: f(x)= 5.9
differential_evolution step 293: f(x)= 5.9
differential_evolution step 294: f(x)= 5.9
differential_evolution step 295: f(x)= 5.9
differential_evolution step 296: f(x)= 5.9
differential_evolution step 297: f(x)= 5.9
differential_evolution step 298: f(x)= 5.9
differential_evolution step 299: f(x)= 5.9
differential_evolution step 300: f(x)= 5.9
differential_evolution step 301: f(x)= 5.9
differential_evolution step 302: f(x)= 5.9
differential_evolution step 303: f(x)= 5.9
differential_evolution step 304: f(x)= 5.9
differential_evolution step 305: f(x)= 5.9
differential_evolution step 306: f(x)= 5.9
differential_evolution step 307: f(x)= 5.9
differential_evolution step 308: f(x)= 5.9
differential_evolution step 309: f(x)= 5.9
differential_evolution step 310: f(x)= 5.9
differential_evolution step 311: f(x)= 5.9
differential_evolution step 312: f(x)= 5.9
differential_evolution step 313: f(x)= 5.9
differential_evolution step 314: f(x)= 5.9
differential_evolution step 315: f(x)= 5.9
differential_evolution step 316: f(x)= 5.9
differential_evolution step 317: f(x)= 5.9
differential_evolution step 318: f(x)= 5.9
differential_evolution step 319: f(x)= 5.9
```

```
differential_evolution step 320: f(x)= 5.9
differential_evolution step 321: f(x)= 5.9
differential_evolution step 322: f(x)= 5.9
differential_evolution step 323: f(x)= 5.9
differential_evolution step 324: f(x)= 5.9
differential_evolution step 325: f(x)= 5.9
differential_evolution step 326: f(x)= 5.9
differential_evolution step 327: f(x)= 5.9
differential_evolution step 328: f(x)= 5.9
differential_evolution step 329: f(x)= 5.9
differential_evolution step 330: f(x)= 5.9
differential_evolution step 331: f(x)= 5.9
differential_evolution step 332: f(x)= 5.9
differential_evolution step 333: f(x)= 5.9
differential_evolution step 334: f(x)= 5.9
differential_evolution step 335: f(x)= 5.9
differential_evolution step 336: f(x)= 5.9
differential_evolution step 337: f(x)= 5.9
differential_evolution step 338: f(x)= 5.9
differential_evolution step 339: f(x)= 5.9
differential_evolution step 340: f(x)= 5.9
differential_evolution step 341: f(x)= 5.8
differential_evolution step 342: f(x)= 5.8
differential_evolution step 343: f(x)= 5.8
differential_evolution step 344: f(x)= 5.699999999999999
differential_evolution step 345: f(x)= 5.699999999999999
differential_evolution step 346: f(x)= 5.699999999999999
differential_evolution step 347: f(x)= 5.699999999999999
differential_evolution step 348: f(x)= 5.699999999999999
differential_evolution step 349: f(x)= 5.699999999999999
differential_evolution step 350: f(x)= 5.699999999999999
differential_evolution step 351: f(x)= 5.699999999999999
differential_evolution step 352: f(x)= 5.699999999999999
differential_evolution step 353: f(x)= 5.699999999999999
differential_evolution step 354: f(x)= 5.699999999999999
differential_evolution step 355: f(x)= 5.699999999999999
differential_evolution step 356: f(x)= 5.1
differential_evolution step 357: f(x)= 5.1
differential_evolution step 358: f(x)= 5.1
differential_evolution step 359: f(x)= 5.1
differential_evolution step 360: f(x)= 5.1
differential_evolution step 361: f(x)= 5.1
differential_evolution step 362: f(x)= 5.1
differential_evolution step 363: f(x)= 5.1
differential_evolution step 364: f(x)= 5.1
differential_evolution step 365: f(x)= 5.1
differential_evolution step 366: f(x)= 5.1
differential_evolution step 367: f(x)= 5.1
differential_evolution step 368: f(x)= 5.1
differential_evolution step 369: f(x)= 5.1
differential_evolution step 370: f(x)= 5.1
differential_evolution step 371: f(x)= 5.1
differential_evolution step 372: f(x)= 5.1
differential_evolution step 373: f(x)= 5.1
differential_evolution step 374: f(x)= 5.1
differential_evolution step 375: f(x)= 5.1
differential_evolution step 376: f(x)= 5.1
differential_evolution step 377: f(x)= 5.1
differential_evolution step 378: f(x)= 5.1
differential_evolution step 379: f(x)= 5.1
differential_evolution step 380: f(x)= 5.1
differential_evolution step 381: f(x)= 5.1
differential_evolution step 382: f(x)= 5.1
differential_evolution step 383: f(x)= 5.1
```

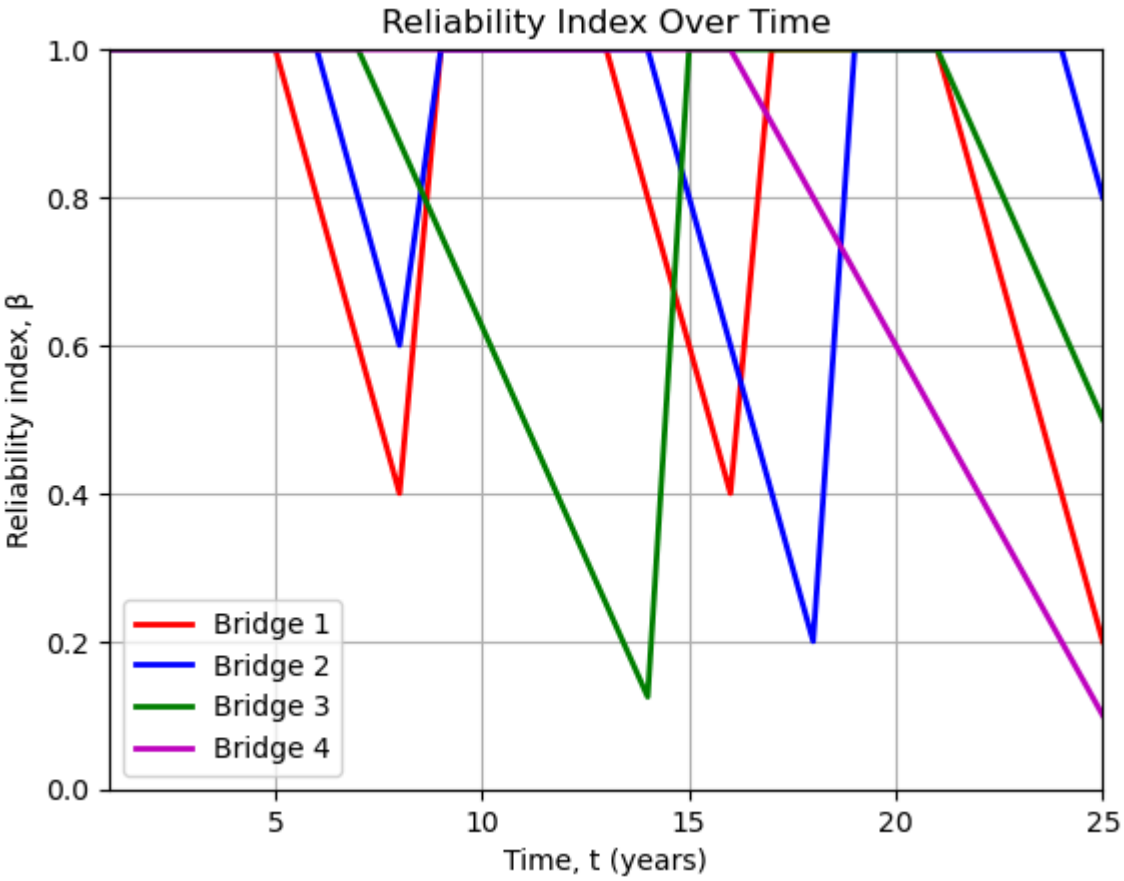
```
differential_evolution step 384: f(x)= 5.1
differential_evolution step 385: f(x)= 5.1
differential_evolution step 386: f(x)= 5.1
differential_evolution step 387: f(x)= 5.1
differential_evolution step 388: f(x)= 5.1
differential_evolution step 389: f(x)= 5.1
differential_evolution step 390: f(x)= 5.1
differential_evolution step 391: f(x)= 5.1
differential_evolution step 392: f(x)= 5.1
differential_evolution step 393: f(x)= 5.1
differential_evolution step 394: f(x)= 5.1
differential_evolution step 395: f(x)= 5.1
differential_evolution step 396: f(x)= 5.1
differential_evolution step 397: f(x)= 5.1
differential_evolution step 398: f(x)= 5.1
differential_evolution step 399: f(x)= 5.1
differential_evolution step 400: f(x)= 5.1
differential_evolution step 401: f(x)= 5.1
differential_evolution step 402: f(x)= 5.1
differential_evolution step 403: f(x)= 5.1
differential_evolution step 404: f(x)= 5.1
differential_evolution step 405: f(x)= 5.1
differential_evolution step 406: f(x)= 5.1
differential_evolution step 407: f(x)= 5.1
differential_evolution step 408: f(x)= 5.1
differential_evolution step 409: f(x)= 5.1
differential_evolution step 410: f(x)= 5.1
differential_evolution step 411: f(x)= 5.1
differential_evolution step 412: f(x)= 5.1
differential_evolution step 413: f(x)= 5.1
differential_evolution step 414: f(x)= 5.1
differential_evolution step 415: f(x)= 5.1
differential_evolution step 416: f(x)= 5.1
differential_evolution step 417: f(x)= 5.1
differential_evolution step 418: f(x)= 5.1
differential_evolution step 419: f(x)= 4.5
differential_evolution step 420: f(x)= 4.5
differential_evolution step 421: f(x)= 4.5
differential_evolution step 422: f(x)= 4.5
differential_evolution step 423: f(x)= 4.5
differential_evolution step 424: f(x)= 4.5
differential_evolution step 425: f(x)= 4.5
differential_evolution step 426: f(x)= 4.5
differential_evolution step 427: f(x)= 4.5
differential_evolution step 428: f(x)= 4.5
differential_evolution step 429: f(x)= 4.5
differential_evolution step 430: f(x)= 4.5
differential_evolution step 431: f(x)= 4.5
differential_evolution step 432: f(x)= 4.5
differential_evolution step 433: f(x)= 4.5
differential_evolution step 434: f(x)= 4.5
differential_evolution step 435: f(x)= 4.5
differential_evolution step 436: f(x)= 4.5
differential_evolution step 437: f(x)= 4.5
differential_evolution step 438: f(x)= 4.5
differential_evolution step 439: f(x)= 4.5
differential_evolution step 440: f(x)= 4.5
differential_evolution step 441: f(x)= 4.5
differential_evolution step 442: f(x)= 4.5
differential_evolution step 443: f(x)= 4.5
differential_evolution step 444: f(x)= 4.5
differential_evolution step 445: f(x)= 4.5
differential_evolution step 446: f(x)= 4.5
differential_evolution step 447: f(x)= 4.5
```

```
differential_evolution step 448: f(x)= 4.5
differential_evolution step 449: f(x)= 4.5
differential_evolution step 450: f(x)= 4.5
differential_evolution step 451: f(x)= 4.5
differential_evolution step 452: f(x)= 4.5
differential_evolution step 453: f(x)= 4.5
differential_evolution step 454: f(x)= 4.5
differential_evolution step 455: f(x)= 4.5
differential_evolution step 456: f(x)= 4.5
differential_evolution step 457: f(x)= 4.5
differential_evolution step 458: f(x)= 4.5
differential_evolution step 459: f(x)= 4.5
differential_evolution step 460: f(x)= 4.5
differential_evolution step 461: f(x)= 4.5
differential_evolution step 462: f(x)= 4.5
differential_evolution step 463: f(x)= 4.5
differential_evolution step 464: f(x)= 4.5
differential_evolution step 465: f(x)= 4.5
differential_evolution step 466: f(x)= 4.5
differential_evolution step 467: f(x)= 4.5
differential_evolution step 468: f(x)= 4.5
differential_evolution step 469: f(x)= 4.5
differential_evolution step 470: f(x)= 4.5
differential_evolution step 471: f(x)= 4.5
differential_evolution step 472: f(x)= 4.5
differential_evolution step 473: f(x)= 4.5
differential_evolution step 474: f(x)= 4.5
differential_evolution step 475: f(x)= 4.5
differential_evolution step 476: f(x)= 4.5
differential_evolution step 477: f(x)= 4.5
differential_evolution step 478: f(x)= 4.5
differential_evolution step 479: f(x)= 4.5
differential_evolution step 480: f(x)= 4.5
differential_evolution step 481: f(x)= 4.5
differential_evolution step 482: f(x)= 4.5
differential_evolution step 483: f(x)= 4.5
differential_evolution step 484: f(x)= 4.5
differential_evolution step 485: f(x)= 4.5
differential_evolution step 486: f(x)= 4.5
differential_evolution step 487: f(x)= 4.5
differential_evolution step 488: f(x)= 4.5
differential_evolution step 489: f(x)= 4.5
differential_evolution step 490: f(x)= 4.5
differential_evolution step 491: f(x)= 4.5
differential_evolution step 492: f(x)= 4.5
differential_evolution step 493: f(x)= 4.5
differential_evolution step 494: f(x)= 4.5
differential_evolution step 495: f(x)= 4.5
differential_evolution step 496: f(x)= 4.5
differential_evolution step 497: f(x)= 4.5
differential_evolution step 498: f(x)= 4.5
differential_evolution step 499: f(x)= 4.5
differential_evolution step 500: f(x)= 4.5
differential_evolution step 501: f(x)= 4.5
differential_evolution step 502: f(x)= 4.5
differential_evolution step 503: f(x)= 4.5
differential_evolution step 504: f(x)= 4.5
differential_evolution step 505: f(x)= 4.5
differential_evolution step 506: f(x)= 4.5
differential_evolution step 507: f(x)= 4.5
differential_evolution step 508: f(x)= 4.5
differential_evolution step 509: f(x)= 4.5
differential_evolution step 510: f(x)= 4.5
differential_evolution step 511: f(x)= 4.5
```

```
differential_evolution step 512: f(x)= 4.5
differential_evolution step 513: f(x)= 4.5
differential_evolution step 514: f(x)= 4.5
differential_evolution step 515: f(x)= 4.5
differential_evolution step 516: f(x)= 4.5
differential_evolution step 517: f(x)= 4.5
             message: Optimization terminated successfully.
             success: True
                 fun: 4.5
                   x: [ 0.000e+00  8.000e+00 ...  1.400e+01  0.000e+00]
                 nit: 517
                nfev: 17373
          population: [[ 0.000e+00  8.000e+00 ...  1.400e+01  0.000e+00]
                       [ 0.000e+00  7.000e+00 ...  1.400e+01  0.000e+00]
                       ...
                       [ 0.000e+00  8.000e+00 ...  1.300e+01  0.000e+00]
                       [ 0.000e+00  9.000e+00 ...  1.400e+01  0.000e+00]]
 population_energies: [ 4.500e+00  4.500e+00 ...  4.500e+00  4.500e+00]
              constr: [array([ 0.000e+00,  0.000e+00, ...,  0.000e+00,
                               0.000e+00])]
    constr_violation: 0.0
                maxcv: 0.0
```

```python
Yopt = result.x
OptInterv = cumulateFreq(Yopt.reshape((nt, nb)).T)
X_bt = Frequency2Schedule(Yopt, nb, nt, Thoriz)
beta_bt = MaintenanceApplication(beta_bt0, X_bt, Thoriz)

print(f'Min Cost = {result.fun:.2f}')
ninterv = np.sum(CountNinterv(Yopt, nt, nb))
print(f'Number of interventions = {ninterv}')
c = nonlcon(Yopt)
BudgetIssues = np.sum(c[-Thoriz - 1:-1] > 0)
print(f'Number of time intervals with exceeding budget = {BudgetIssues}')
reliabIssues = c[-1]
print(f'Number of time intervals with reliability issues = {reliabIssues}')

plt.figure()
h = []
leg = []
for b in range(nb):
    h.append(plt.plot(range(1, Thoriz + 1), beta_bt[b, :], f'-{col[b]}', linewidth=2))
    leg.append(f'Bridge {b + 1}')
plt.legend(leg, loc='best')
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()
```

```
Min Cost = 4.50
Number of interventions = 6
Number of time intervals with exceeding budget = 0
Number of time intervals with reliability issues = 0.0
```


Reliability Index Over Time
Reliability index, β
Time, t (years)
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
Bridge 1
Bridge 2
Bridge 3
Bridge 4

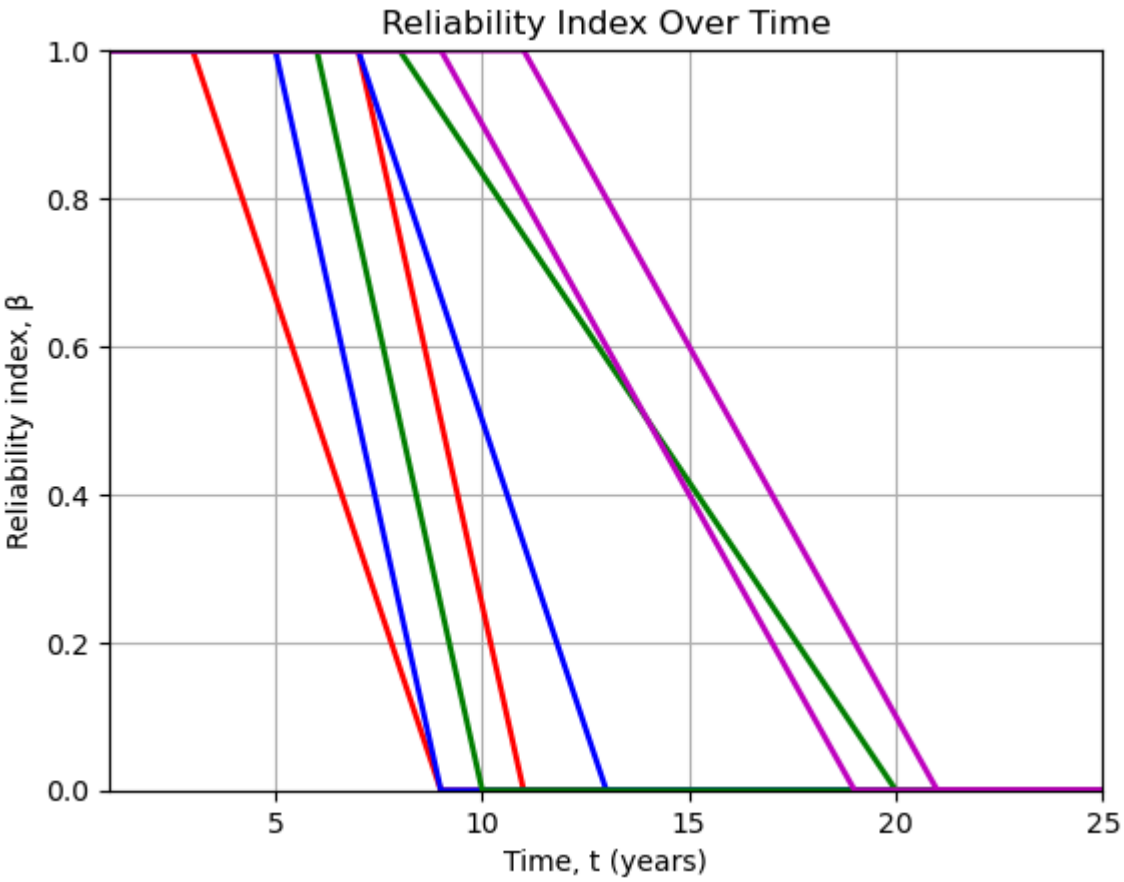
Robust

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution
import scipy as sp

def reliability(beta0_b, d_b, L_b, Thoriz):
    # --- Reliability estimation
    # beta_bt: reliability curve over time for bridge b
    # beta0_b: Initial reliability of bridge b at t=0
    # d_b, L_b: when the degradation starts and reaches beta=0 for each bridge
    # Thoriz: maximum time studied
    nb = len(beta0_b)
    beta_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        #horziontal branch
        timesteps = int(np.floor(d_b[b]))
        beta_bt[b, :timesteps] = beta0_b[b]

        #degradation till L_b
        servicelife = int(np.floor(L_b[b]))
        t = np.arange(timesteps + 1, servicelife + 1)
        beta_bt[b, timesteps:servicelife] = beta0_b[b] / (L_b[b] - d_b[b]) * (L_b[b] - t)

    return beta_bt

def MaintenanceApplication(beta_bt, X_bt, Thoriz):
    # --- Given a maintenance strategy given by X_bt, where 1 means that bridge
    # b undergoes maintenance at time t, this function gives the updated
    # reliability profile
    updatedbeta_bt = beta_bt.copy()
    InterventionTimes = np.where(np.sum(X_bt, axis=0) > 0)[0]

    for t in InterventionTimes:
        intervbrid = np.where(X_bt[:, t] == 1)[0]
        timecells = Thoriz - t - 1
        updatedbeta_bt[intervbrid, t+1:Thoriz] = beta_bt[intervbrid, :timecells]

    return updatedbeta_bt

def Frequency2Schedule(y_b, nb, nt, Thoriz):
    # Transforms a vector indicating the frequency to a schedule matrix [0/1]
    freq_b = y_b.reshape((nt, nb)).T
    cumFreq_b = cumulateFreq(freq_b)
    X_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        idx = cumFreq_b[b, cumFreq_b[b, :] != 0].astype(int)
        idx = idx[idx <= Thoriz]
        if len(idx) > 0:
            X_bt[b, idx - 1] = 1

    return X_bt

def cumulateFreq(freq_b):
    # creates the cumulative frequency
    nb, nt = freq_b.shape
    cumFreq_b = np.zeros((nb, nt))

    for b in range(nb):
        f_b = freq_b[b, freq_b[b, :] != 0]
        cumFreq_b[b, :len(f_b)] = np.cumsum(f_b)
```

```
    return cumFreq_b

def CountNinterv(y_b, nt, nb):
    # from vector y_b, it computes the number of interventions per bridge
    y_b01 = y_b != 0
    freq_b = y_b01.reshape((nt, nb)).T
    ninterv_b = np.sum(freq_b, axis=1)
    return ninterv_b
```

```python
Thoriz = 25
# for each bridge b:  beta0 d1(years) d2(years) L1(years) L2(years) C(Meuros)
data = np.array([
    [1, 3, 7, 9, 11, 0.6],
    [1, 5, 7, 9, 13, 0.8],
    [1, 6, 8, 10, 20, 1.0],
    [1, 9, 11, 19, 21, 0.7]
])
beta0_b, d1_b, d2_b, L1_b, L2_b, C_b = data[:, 0], data[:, 1], data[:, 2], data[:, 3], da

# Annual resources (M euro)
R = 1.4

# Initial reliability estimation
nb = len(beta0_b)
beta_bt01 = reliability(beta0_b, d1_b, L1_b, Thoriz)
beta_bt02 = reliability(beta0_b, d2_b, L2_b, Thoriz)

plt.figure()
col = ['r', 'b', 'g', 'm']
for b in range(nb):
    plt.plot(range(1, Thoriz + 1), beta_bt01[b, :], f'-{col[b]}', linewidth=2)
    plt.plot(range(1, Thoriz + 1), beta_bt02[b, :], f'-{col[b]}', linewidth=2)
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()

# Deterministic Optimization
# Variables ->  e.g., freq=[10 2 0 0 ; 5, 5 0 0; 0 0 0 0];

maxInter_b = np.ceil(Thoriz / L1_b).astype(int) + 1 #max number of interventions expected
nt = max(maxInter_b) #to dimension the matrix, max interventions experienced by the worst
nvars = nb * nt #number of design variables
integers = np.array([True] * nvars) #all variables are integers

#  Range of definition: [0,max number of years with no interventions==L_b]

lb = np.zeros(nvars)
ub = np.zeros(nvars)
pos = 0
for b in range(nb):
    # This guarantees interventions freq. within the service life (if conducted)
    ub[pos:pos + maxInter_b[b]] = L2_b[b] - 1
    # this forces zeros after the max number of interventions expected
    if maxInter_b[b] < nb:
        ub[pos + maxInter_b[b]:pos + nb - 1] = 0
    pos += nt

def nonlcon(y_b):
    #  From frequency to a schedule matrix [0/1]
    X_bt = Frequency2Schedule(y_b, nb, nt, Thoriz)

    # Constraint 1 >>> the last intervention is within Thoriz --> LastInterv<Thoriz --> L
    lastInterv_b = np.max(cumulateFreq(y_b.reshape((nt, nb)).T), axis=1)
    c1 = lastInterv_b - Thoriz - 1

    # Constraint 2 >>> C_b*X_bt <= R   for all t
    c2 = np.sum(C_b[:, None] * X_bt, axis=0) - R
```

```
    # Constraint 3 >>> beta_bt>0 for the lowest reliability
    beta_bt = MaintenanceApplication(beta_bt01, X_bt, Thoriz)
    c3 = np.array([np.sum(beta_bt == 0)])

    # Constraint 4 >>> beta_bt>0 for the highest reliability
    beta_bt = MaintenanceApplication(beta_bt02, X_bt, Thoriz)
    c4 = np.array([np.sum(beta_bt == 0)])

    c = np.concatenate([c1, c2, c3, c4])
    return c

cons = sp.optimize.NonlinearConstraint(nonlcon, np.array([-np.inf] * (nb + Thoriz + 2)),

def fun(y_b):
    #  minimize cost trying to go for the largest time between interventions (== max y_b)
    ninterv_b = CountNinterv(y_b, nt, nb)
    return np.dot(C_b, ninterv_b)

bounds = [[lb[i], ub[i]] for i in range(nvars)]
```




```
result = differential_evolution(fun, bounds, constraints=cons, integrality=integers, disp
print(result)
```

```
differential_evolution step 1: f(x)= inf
differential_evolution step 2: f(x)= inf
differential_evolution step 3: f(x)= inf
differential_evolution step 4: f(x)= inf
differential_evolution step 5: f(x)= inf
differential_evolution step 6: f(x)= inf
differential_evolution step 7: f(x)= inf
differential_evolution step 8: f(x)= inf
differential_evolution step 9: f(x)= inf
differential_evolution step 10: f(x)= inf
differential_evolution step 11: f(x)= 11.7
differential_evolution step 12: f(x)= 11.7
differential_evolution step 13: f(x)= 11.7
differential_evolution step 14: f(x)= 11.7
differential_evolution step 15: f(x)= 11.7
differential_evolution step 16: f(x)= 11.7
differential_evolution step 17: f(x)= 11.7
differential_evolution step 18: f(x)= 11.7
differential_evolution step 19: f(x)= 11.7
differential_evolution step 20: f(x)= 11.7
differential_evolution step 21: f(x)= 11.7
differential_evolution step 22: f(x)= 11.7
differential_evolution step 23: f(x)= 11.7
differential_evolution step 24: f(x)= 11.7
differential_evolution step 25: f(x)= 11.7
differential_evolution step 26: f(x)= 11.7
differential_evolution step 27: f(x)= 11.7
differential_evolution step 28: f(x)= 11.7
differential_evolution step 29: f(x)= 11.7
differential_evolution step 30: f(x)= 11.7
differential_evolution step 31: f(x)= 11.7
differential_evolution step 32: f(x)= 11.7
differential_evolution step 33: f(x)= 11.7
differential_evolution step 34: f(x)= 11.7
differential_evolution step 35: f(x)= 11.7
differential_evolution step 36: f(x)= 11.7
differential_evolution step 37: f(x)= 11.7
differential_evolution step 38: f(x)= 11.7
differential_evolution step 39: f(x)= 11.7
differential_evolution step 40: f(x)= 11.7
differential_evolution step 41: f(x)= 11.7
differential_evolution step 42: f(x)= 11.7
differential_evolution step 43: f(x)= 11.7
differential_evolution step 44: f(x)= 11.7
differential_evolution step 45: f(x)= 11.7
differential_evolution step 46: f(x)= 11.7
differential_evolution step 47: f(x)= 11.7
differential_evolution step 48: f(x)= 11.7
differential_evolution step 49: f(x)= 11.7
differential_evolution step 50: f(x)= 11.7
differential_evolution step 51: f(x)= 11.7
differential_evolution step 52: f(x)= 10.7
differential_evolution step 53: f(x)= 10.7
differential_evolution step 54: f(x)= 10.7
differential_evolution step 55: f(x)= 10.7
differential_evolution step 56: f(x)= 10.7
differential_evolution step 57: f(x)= 10.7
differential_evolution step 58: f(x)= 10.7
differential_evolution step 59: f(x)= 10.7
differential_evolution step 60: f(x)= 10.7
differential_evolution step 61: f(x)= 10.7
differential_evolution step 62: f(x)= 10.7
differential_evolution step 63: f(x)= 10.7
```

```
differential_evolution step 64: f(x)= 10.7
differential_evolution step 65: f(x)= 10.7
differential_evolution step 66: f(x)= 10.7
differential_evolution step 67: f(x)= 10.7
differential_evolution step 68: f(x)= 10.7
differential_evolution step 69: f(x)= 10.7
differential_evolution step 70: f(x)= 10.7
differential_evolution step 71: f(x)= 10.7
differential_evolution step 72: f(x)= 10.7
differential_evolution step 73: f(x)= 10.7
differential_evolution step 74: f(x)= 10.7
differential_evolution step 75: f(x)= 10.7
differential_evolution step 76: f(x)= 10.7
differential_evolution step 77: f(x)= 10.7
differential_evolution step 78: f(x)= 10.7
differential_evolution step 79: f(x)= 10.7
differential_evolution step 80: f(x)= 10.7
differential_evolution step 81: f(x)= 10.7
differential_evolution step 82: f(x)= 10.7
differential_evolution step 83: f(x)= 10.7
differential_evolution step 84: f(x)= 10.7
differential_evolution step 85: f(x)= 10.7
differential_evolution step 86: f(x)= 10.7
differential_evolution step 87: f(x)= 10.7
differential_evolution step 88: f(x)= 10.7
differential_evolution step 89: f(x)= 10.7
differential_evolution step 90: f(x)= 10.7
differential_evolution step 91: f(x)= 10.7
differential_evolution step 92: f(x)= 10.7
differential_evolution step 93: f(x)= 10.7
differential_evolution step 94: f(x)= 10.7
differential_evolution step 95: f(x)= 10.7
differential_evolution step 96: f(x)= 10.7
differential_evolution step 97: f(x)= 10.7
differential_evolution step 98: f(x)= 10.7
differential_evolution step 99: f(x)= 10.7
differential_evolution step 100: f(x)= 10.7
differential_evolution step 101: f(x)= 10.7
differential_evolution step 102: f(x)= 10.7
differential_evolution step 103: f(x)= 10.7
differential_evolution step 104: f(x)= 10.7
differential_evolution step 105: f(x)= 10.7
differential_evolution step 106: f(x)= 10.7
differential_evolution step 107: f(x)= 10.7
differential_evolution step 108: f(x)= 10.7
differential_evolution step 109: f(x)= 10.7
differential_evolution step 110: f(x)= 10.7
differential_evolution step 111: f(x)= 10.1
differential_evolution step 112: f(x)= 10.1
differential_evolution step 113: f(x)= 10.1
differential_evolution step 114: f(x)= 10.1
differential_evolution step 115: f(x)= 10.1
differential_evolution step 116: f(x)= 10.1
differential_evolution step 117: f(x)= 10.0
differential_evolution step 118: f(x)= 10.0
differential_evolution step 119: f(x)= 10.0
differential_evolution step 120: f(x)= 10.0
differential_evolution step 121: f(x)= 10.0
differential_evolution step 122: f(x)= 10.0
differential_evolution step 123: f(x)= 10.0
differential_evolution step 124: f(x)= 10.0
differential_evolution step 125: f(x)= 10.0
differential_evolution step 126: f(x)= 10.0
differential_evolution step 127: f(x)= 10.0
```

```
differential_evolution step 128: f(x)= 10.0
differential_evolution step 129: f(x)= 10.0
differential_evolution step 130: f(x)= 10.0
differential_evolution step 131: f(x)= 10.0
differential_evolution step 132: f(x)= 10.0
differential_evolution step 133: f(x)= 10.0
differential_evolution step 134: f(x)= 10.0
differential_evolution step 135: f(x)= 10.0
differential_evolution step 136: f(x)= 10.0
differential_evolution step 137: f(x)= 10.0
differential_evolution step 138: f(x)= 10.0
differential_evolution step 139: f(x)= 10.0
differential_evolution step 140: f(x)= 10.0
differential_evolution step 141: f(x)= 10.0
differential_evolution step 142: f(x)= 10.0
differential_evolution step 143: f(x)= 10.0
differential_evolution step 144: f(x)= 10.0
differential_evolution step 145: f(x)= 10.0
differential_evolution step 146: f(x)= 10.0
differential_evolution step 147: f(x)= 10.0
differential_evolution step 148: f(x)= 10.0
differential_evolution step 149: f(x)= 10.0
differential_evolution step 150: f(x)= 10.0
differential_evolution step 151: f(x)= 10.0
differential_evolution step 152: f(x)= 10.0
differential_evolution step 153: f(x)= 10.0
differential_evolution step 154: f(x)= 10.0
differential_evolution step 155: f(x)= 10.0
differential_evolution step 156: f(x)= 10.0
differential_evolution step 157: f(x)= 10.0
differential_evolution step 158: f(x)= 10.0
differential_evolution step 159: f(x)= 10.0
differential_evolution step 160: f(x)= 10.0
differential_evolution step 161: f(x)= 10.0
differential_evolution step 162: f(x)= 10.0
differential_evolution step 163: f(x)= 10.0
differential_evolution step 164: f(x)= 10.0
differential_evolution step 165: f(x)= 10.0
differential_evolution step 166: f(x)= 10.0
differential_evolution step 167: f(x)= 10.0
differential_evolution step 168: f(x)= 10.0
differential_evolution step 169: f(x)= 10.0
differential_evolution step 170: f(x)= 10.0
differential_evolution step 171: f(x)= 10.0
differential_evolution step 172: f(x)= 10.0
differential_evolution step 173: f(x)= 10.0
differential_evolution step 174: f(x)= 10.0
differential_evolution step 175: f(x)= 10.0
differential_evolution step 176: f(x)= 10.0
differential_evolution step 177: f(x)= 10.0
differential_evolution step 178: f(x)= 10.0
differential_evolution step 179: f(x)= 10.0
differential_evolution step 180: f(x)= 10.0
differential_evolution step 181: f(x)= 9.200000000000001
differential_evolution step 182: f(x)= 9.200000000000001
differential_evolution step 183: f(x)= 9.200000000000001
differential_evolution step 184: f(x)= 9.200000000000001
differential_evolution step 185: f(x)= 9.200000000000001
differential_evolution step 186: f(x)= 9.200000000000001
differential_evolution step 187: f(x)= 9.200000000000001
differential_evolution step 188: f(x)= 9.200000000000001
differential_evolution step 189: f(x)= 9.200000000000001
differential_evolution step 190: f(x)= 9.200000000000001
differential_evolution step 191: f(x)= 9.200000000000001
```

```
differential_evolution step 192: f(x)= 9.200000000000001
differential_evolution step 193: f(x)= 9.200000000000001
differential_evolution step 194: f(x)= 9.200000000000001
differential_evolution step 195: f(x)= 9.200000000000001
differential_evolution step 196: f(x)= 9.200000000000001
differential_evolution step 197: f(x)= 9.200000000000001
differential_evolution step 198: f(x)= 9.200000000000001
differential_evolution step 199: f(x)= 9.200000000000001
differential_evolution step 200: f(x)= 9.200000000000001
differential_evolution step 201: f(x)= 9.200000000000001
differential_evolution step 202: f(x)= 9.200000000000001
differential_evolution step 203: f(x)= 9.200000000000001
differential_evolution step 204: f(x)= 9.200000000000001
differential_evolution step 205: f(x)= 9.200000000000001
differential_evolution step 206: f(x)= 9.200000000000001
differential_evolution step 207: f(x)= 9.200000000000001
differential_evolution step 208: f(x)= 9.200000000000001
differential_evolution step 209: f(x)= 9.200000000000001
differential_evolution step 210: f(x)= 9.200000000000001
differential_evolution step 211: f(x)= 9.200000000000001
differential_evolution step 212: f(x)= 9.200000000000001
differential_evolution step 213: f(x)= 9.200000000000001
differential_evolution step 214: f(x)= 9.200000000000001
differential_evolution step 215: f(x)= 9.200000000000001
differential_evolution step 216: f(x)= 9.200000000000001
differential_evolution step 217: f(x)= 9.200000000000001
differential_evolution step 218: f(x)= 9.200000000000001
differential_evolution step 219: f(x)= 9.200000000000001
differential_evolution step 220: f(x)= 9.200000000000001
differential_evolution step 221: f(x)= 9.200000000000001
differential_evolution step 222: f(x)= 9.200000000000001
differential_evolution step 223: f(x)= 9.200000000000001
differential_evolution step 224: f(x)= 9.200000000000001
differential_evolution step 225: f(x)= 9.200000000000001
differential_evolution step 226: f(x)= 9.200000000000001
differential_evolution step 227: f(x)= 9.200000000000001
differential_evolution step 228: f(x)= 9.200000000000001
differential_evolution step 229: f(x)= 9.200000000000001
differential_evolution step 230: f(x)= 9.200000000000001
differential_evolution step 231: f(x)= 9.200000000000001
differential_evolution step 232: f(x)= 9.200000000000001
differential_evolution step 233: f(x)= 9.200000000000001
differential_evolution step 234: f(x)= 9.200000000000001
differential_evolution step 235: f(x)= 9.200000000000001
differential_evolution step 236: f(x)= 9.200000000000001
differential_evolution step 237: f(x)= 9.200000000000001
differential_evolution step 238: f(x)= 9.200000000000001
differential_evolution step 239: f(x)= 9.200000000000001
differential_evolution step 240: f(x)= 9.200000000000001
differential_evolution step 241: f(x)= 9.200000000000001
differential_evolution step 242: f(x)= 9.200000000000001
differential_evolution step 243: f(x)= 9.200000000000001
differential_evolution step 244: f(x)= 9.200000000000001
differential_evolution step 245: f(x)= 9.200000000000001
differential_evolution step 246: f(x)= 9.200000000000001
differential_evolution step 247: f(x)= 9.200000000000001
differential_evolution step 248: f(x)= 9.200000000000001
differential_evolution step 249: f(x)= 9.200000000000001
differential_evolution step 250: f(x)= 9.200000000000001
differential_evolution step 251: f(x)= 9.200000000000001
differential_evolution step 252: f(x)= 9.200000000000001
differential_evolution step 253: f(x)= 9.200000000000001
differential_evolution step 254: f(x)= 9.200000000000001
differential_evolution step 255: f(x)= 9.200000000000001
```

```
differential_evolution step 256: f(x)= 9.200000000000001
differential_evolution step 257: f(x)= 9.200000000000001
differential_evolution step 258: f(x)= 9.200000000000001
differential_evolution step 259: f(x)= 9.200000000000001
differential_evolution step 260: f(x)= 9.200000000000001
differential_evolution step 261: f(x)= 9.200000000000001
differential_evolution step 262: f(x)= 9.200000000000001
differential_evolution step 263: f(x)= 9.200000000000001
differential_evolution step 264: f(x)= 9.200000000000001
differential_evolution step 265: f(x)= 9.200000000000001
differential_evolution step 266: f(x)= 9.200000000000001
differential_evolution step 267: f(x)= 9.200000000000001
differential_evolution step 268: f(x)= 9.200000000000001
differential_evolution step 269: f(x)= 9.200000000000001
differential_evolution step 270: f(x)= 9.200000000000001
differential_evolution step 271: f(x)= 9.200000000000001
differential_evolution step 272: f(x)= 9.200000000000001
differential_evolution step 273: f(x)= 9.200000000000001
differential_evolution step 274: f(x)= 9.200000000000001
differential_evolution step 275: f(x)= 9.200000000000001
differential_evolution step 276: f(x)= 9.200000000000001
differential_evolution step 277: f(x)= 9.200000000000001
differential_evolution step 278: f(x)= 9.200000000000001
differential_evolution step 279: f(x)= 9.200000000000001
differential_evolution step 280: f(x)= 9.200000000000001
differential_evolution step 281: f(x)= 9.200000000000001
differential_evolution step 282: f(x)= 9.200000000000001
differential_evolution step 283: f(x)= 9.200000000000001
differential_evolution step 284: f(x)= 9.200000000000001
differential_evolution step 285: f(x)= 9.200000000000001
differential_evolution step 286: f(x)= 9.200000000000001
differential_evolution step 287: f(x)= 9.200000000000001
differential_evolution step 288: f(x)= 9.200000000000001
differential_evolution step 289: f(x)= 9.200000000000001
differential_evolution step 290: f(x)= 9.200000000000001
differential_evolution step 291: f(x)= 9.200000000000001
differential_evolution step 292: f(x)= 9.200000000000001
differential_evolution step 293: f(x)= 9.200000000000001
differential_evolution step 294: f(x)= 9.200000000000001
differential_evolution step 295: f(x)= 9.200000000000001
differential_evolution step 296: f(x)= 9.200000000000001
differential_evolution step 297: f(x)= 9.200000000000001
differential_evolution step 298: f(x)= 9.200000000000001
differential_evolution step 299: f(x)= 9.200000000000001
differential_evolution step 300: f(x)= 9.200000000000001
differential_evolution step 301: f(x)= 9.200000000000001
differential_evolution step 302: f(x)= 9.200000000000001
differential_evolution step 303: f(x)= 9.200000000000001
differential_evolution step 304: f(x)= 9.200000000000001
differential_evolution step 305: f(x)= 9.200000000000001
differential_evolution step 306: f(x)= 9.200000000000001
differential_evolution step 307: f(x)= 9.200000000000001
differential_evolution step 308: f(x)= 9.200000000000001
differential_evolution step 309: f(x)= 8.7
differential_evolution step 310: f(x)= 8.7
differential_evolution step 311: f(x)= 8.7
differential_evolution step 312: f(x)= 8.7
differential_evolution step 313: f(x)= 8.7
differential_evolution step 314: f(x)= 8.7
differential_evolution step 315: f(x)= 8.7
differential_evolution step 316: f(x)= 8.7
differential_evolution step 317: f(x)= 8.7
differential_evolution step 318: f(x)= 8.7
differential_evolution step 319: f(x)= 8.7
```

```
differential_evolution step 320: f(x)= 8.7
differential_evolution step 321: f(x)= 8.7
differential_evolution step 322: f(x)= 8.7
differential_evolution step 323: f(x)= 8.7
differential_evolution step 324: f(x)= 8.7
differential_evolution step 325: f(x)= 8.7
differential_evolution step 326: f(x)= 8.7
differential_evolution step 327: f(x)= 8.7
differential_evolution step 328: f(x)= 8.7
differential_evolution step 329: f(x)= 8.7
differential_evolution step 330: f(x)= 8.7
differential_evolution step 331: f(x)= 8.7
differential_evolution step 332: f(x)= 8.7
differential_evolution step 333: f(x)= 8.7
differential_evolution step 334: f(x)= 8.7
differential_evolution step 335: f(x)= 8.7
differential_evolution step 336: f(x)= 8.7
differential_evolution step 337: f(x)= 8.7
differential_evolution step 338: f(x)= 8.7
differential_evolution step 339: f(x)= 8.7
differential_evolution step 340: f(x)= 8.7
differential_evolution step 341: f(x)= 8.7
differential_evolution step 342: f(x)= 8.7
differential_evolution step 343: f(x)= 8.7
differential_evolution step 344: f(x)= 8.7
differential_evolution step 345: f(x)= 8.7
differential_evolution step 346: f(x)= 8.7
differential_evolution step 347: f(x)= 8.7
differential_evolution step 348: f(x)= 8.7
differential_evolution step 349: f(x)= 8.7
differential_evolution step 350: f(x)= 8.7
differential_evolution step 351: f(x)= 8.7
differential_evolution step 352: f(x)= 8.7
differential_evolution step 353: f(x)= 8.7
differential_evolution step 354: f(x)= 8.7
differential_evolution step 355: f(x)= 8.7
differential_evolution step 356: f(x)= 8.7
differential_evolution step 357: f(x)= 8.7
differential_evolution step 358: f(x)= 8.7
differential_evolution step 359: f(x)= 8.7
differential_evolution step 360: f(x)= 8.7
differential_evolution step 361: f(x)= 8.7
differential_evolution step 362: f(x)= 8.7
differential_evolution step 363: f(x)= 8.7
differential_evolution step 364: f(x)= 8.7
differential_evolution step 365: f(x)= 8.7
differential_evolution step 366: f(x)= 8.7
differential_evolution step 367: f(x)= 8.7
differential_evolution step 368: f(x)= 8.7
differential_evolution step 369: f(x)= 8.7
differential_evolution step 370: f(x)= 8.7
differential_evolution step 371: f(x)= 8.7
differential_evolution step 372: f(x)= 8.7
differential_evolution step 373: f(x)= 8.7
differential_evolution step 374: f(x)= 8.7
differential_evolution step 375: f(x)= 8.7
differential_evolution step 376: f(x)= 8.7
differential_evolution step 377: f(x)= 8.7
differential_evolution step 378: f(x)= 8.7
differential_evolution step 379: f(x)= 8.7
differential_evolution step 380: f(x)= 8.7
differential_evolution step 381: f(x)= 8.7
differential_evolution step 382: f(x)= 8.7
differential_evolution step 383: f(x)= 8.7
```

```
differential_evolution step 384: f(x)= 8.7
differential_evolution step 385: f(x)= 8.7
differential_evolution step 386: f(x)= 8.7
differential_evolution step 387: f(x)= 8.7
differential_evolution step 388: f(x)= 8.7
differential_evolution step 389: f(x)= 8.7
differential_evolution step 390: f(x)= 8.7
differential_evolution step 391: f(x)= 8.7
differential_evolution step 392: f(x)= 8.7
differential_evolution step 393: f(x)= 8.7
differential_evolution step 394: f(x)= 8.7
differential_evolution step 395: f(x)= 8.7
differential_evolution step 396: f(x)= 8.7
differential_evolution step 397: f(x)= 8.7
differential_evolution step 398: f(x)= 8.7
differential_evolution step 399: f(x)= 8.7
differential_evolution step 400: f(x)= 8.7
differential_evolution step 401: f(x)= 8.7
differential_evolution step 402: f(x)= 8.7
differential_evolution step 403: f(x)= 8.7
differential_evolution step 404: f(x)= 8.7
differential_evolution step 405: f(x)= 8.7
differential_evolution step 406: f(x)= 8.7
differential_evolution step 407: f(x)= 8.7
differential_evolution step 408: f(x)= 8.7
differential_evolution step 409: f(x)= 8.7
differential_evolution step 410: f(x)= 8.7
differential_evolution step 411: f(x)= 8.7
differential_evolution step 412: f(x)= 8.7
differential_evolution step 413: f(x)= 8.7
differential_evolution step 414: f(x)= 8.7
differential_evolution step 415: f(x)= 8.7
differential_evolution step 416: f(x)= 8.7
differential_evolution step 417: f(x)= 8.7
differential_evolution step 418: f(x)= 8.7
differential_evolution step 419: f(x)= 8.7
differential_evolution step 420: f(x)= 8.7
differential_evolution step 421: f(x)= 8.7
differential_evolution step 422: f(x)= 8.7
differential_evolution step 423: f(x)= 8.7
differential_evolution step 424: f(x)= 8.7
differential_evolution step 425: f(x)= 8.7
differential_evolution step 426: f(x)= 8.7
differential_evolution step 427: f(x)= 8.7
differential_evolution step 428: f(x)= 8.7
differential_evolution step 429: f(x)= 8.7
differential_evolution step 430: f(x)= 8.7
differential_evolution step 431: f(x)= 8.7
differential_evolution step 432: f(x)= 8.7
differential_evolution step 433: f(x)= 8.7
differential_evolution step 434: f(x)= 8.7
differential_evolution step 435: f(x)= 8.7
differential_evolution step 436: f(x)= 8.7
differential_evolution step 437: f(x)= 8.7
differential_evolution step 438: f(x)= 8.7
differential_evolution step 439: f(x)= 8.7
differential_evolution step 440: f(x)= 8.7
differential_evolution step 441: f(x)= 8.7
differential_evolution step 442: f(x)= 8.7
differential_evolution step 443: f(x)= 8.7
differential_evolution step 444: f(x)= 8.7
differential_evolution step 445: f(x)= 8.7
differential_evolution step 446: f(x)= 8.7
differential_evolution step 447: f(x)= 8.7
```

```
differential_evolution step 448: f(x)= 8.7
differential_evolution step 449: f(x)= 8.7
differential_evolution step 450: f(x)= 8.7
differential_evolution step 451: f(x)= 8.7
differential_evolution step 452: f(x)= 8.7
differential_evolution step 453: f(x)= 8.7
differential_evolution step 454: f(x)= 8.7
differential_evolution step 455: f(x)= 8.7
differential_evolution step 456: f(x)= 8.7
differential_evolution step 457: f(x)= 8.7
differential_evolution step 458: f(x)= 8.7
differential_evolution step 459: f(x)= 8.7
differential_evolution step 460: f(x)= 8.7
differential_evolution step 461: f(x)= 8.7
differential_evolution step 462: f(x)= 8.7
differential_evolution step 463: f(x)= 8.7
differential_evolution step 464: f(x)= 8.7
differential_evolution step 465: f(x)= 8.7
differential_evolution step 466: f(x)= 8.7
differential_evolution step 467: f(x)= 8.7
differential_evolution step 468: f(x)= 8.7
differential_evolution step 469: f(x)= 8.7
differential_evolution step 470: f(x)= 8.7
differential_evolution step 471: f(x)= 8.7
differential_evolution step 472: f(x)= 8.7
differential_evolution step 473: f(x)= 8.7
differential_evolution step 474: f(x)= 8.7
differential_evolution step 475: f(x)= 8.7
differential_evolution step 476: f(x)= 8.7
differential_evolution step 477: f(x)= 8.7
differential_evolution step 478: f(x)= 8.7
differential_evolution step 479: f(x)= 8.7
differential_evolution step 480: f(x)= 8.7
differential_evolution step 481: f(x)= 8.7
differential_evolution step 482: f(x)= 8.7
differential_evolution step 483: f(x)= 8.7
differential_evolution step 484: f(x)= 8.7
differential_evolution step 485: f(x)= 8.7
differential_evolution step 486: f(x)= 8.7
differential_evolution step 487: f(x)= 8.7
differential_evolution step 488: f(x)= 8.7
differential_evolution step 489: f(x)= 8.7
differential_evolution step 490: f(x)= 8.7
differential_evolution step 491: f(x)= 8.7
differential_evolution step 492: f(x)= 8.7
differential_evolution step 493: f(x)= 8.7
differential_evolution step 494: f(x)= 8.7
differential_evolution step 495: f(x)= 8.7
differential_evolution step 496: f(x)= 8.7
differential_evolution step 497: f(x)= 8.7
differential_evolution step 498: f(x)= 8.7
differential_evolution step 499: f(x)= 8.7
differential_evolution step 500: f(x)= 8.7
differential_evolution step 501: f(x)= 8.7
differential_evolution step 502: f(x)= 8.7
differential_evolution step 503: f(x)= 8.7
differential_evolution step 504: f(x)= 8.7
differential_evolution step 505: f(x)= 8.7
differential_evolution step 506: f(x)= 8.7
differential_evolution step 507: f(x)= 8.7
differential_evolution step 508: f(x)= 8.7
differential_evolution step 509: f(x)= 8.7
differential_evolution step 510: f(x)= 8.7
differential_evolution step 511: f(x)= 8.7
```

```
differential_evolution step 512: f(x)= 8.7
differential_evolution step 513: f(x)= 8.7
differential_evolution step 514: f(x)= 8.7
differential_evolution step 515: f(x)= 8.7
differential_evolution step 516: f(x)= 8.7
differential_evolution step 517: f(x)= 8.7
differential_evolution step 518: f(x)= 8.7
differential_evolution step 519: f(x)= 8.7
differential_evolution step 520: f(x)= 8.7
differential_evolution step 521: f(x)= 8.7
differential_evolution step 522: f(x)= 8.7
differential_evolution step 523: f(x)= 8.7
differential_evolution step 524: f(x)= 8.7
differential_evolution step 525: f(x)= 8.7
differential_evolution step 526: f(x)= 8.7
differential_evolution step 527: f(x)= 8.7
differential_evolution step 528: f(x)= 8.7
differential_evolution step 529: f(x)= 8.7
differential_evolution step 530: f(x)= 8.7
differential_evolution step 531: f(x)= 8.7
differential_evolution step 532: f(x)= 8.7
differential_evolution step 533: f(x)= 8.7
differential_evolution step 534: f(x)= 8.7
differential_evolution step 535: f(x)= 8.7
differential_evolution step 536: f(x)= 8.7
differential_evolution step 537: f(x)= 8.7
differential_evolution step 538: f(x)= 8.7
differential_evolution step 539: f(x)= 8.7
differential_evolution step 540: f(x)= 8.7
differential_evolution step 541: f(x)= 8.7
differential_evolution step 542: f(x)= 8.7
differential_evolution step 543: f(x)= 8.7
differential_evolution step 544: f(x)= 8.7
differential_evolution step 545: f(x)= 8.7
differential_evolution step 546: f(x)= 8.7
differential_evolution step 547: f(x)= 8.7
differential_evolution step 548: f(x)= 8.7
differential_evolution step 549: f(x)= 8.7
differential_evolution step 550: f(x)= 8.7
differential_evolution step 551: f(x)= 8.7
differential_evolution step 552: f(x)= 8.7
differential_evolution step 553: f(x)= 8.7
differential_evolution step 554: f(x)= 8.7
differential_evolution step 555: f(x)= 8.7
differential_evolution step 556: f(x)= 8.7
differential_evolution step 557: f(x)= 8.7
differential_evolution step 558: f(x)= 8.7
differential_evolution step 559: f(x)= 8.7
differential_evolution step 560: f(x)= 8.7
differential_evolution step 561: f(x)= 8.7
differential_evolution step 562: f(x)= 8.7
differential_evolution step 563: f(x)= 8.7
differential_evolution step 564: f(x)= 8.7
differential_evolution step 565: f(x)= 8.7
differential_evolution step 566: f(x)= 8.7
differential_evolution step 567: f(x)= 8.7
differential_evolution step 568: f(x)= 8.7
differential_evolution step 569: f(x)= 8.7
differential_evolution step 570: f(x)= 8.7
differential_evolution step 571: f(x)= 8.399999999999999
differential_evolution step 572: f(x)= 8.399999999999999
differential_evolution step 573: f(x)= 8.399999999999999
differential_evolution step 574: f(x)= 8.399999999999999
differential_evolution step 575: f(x)= 8.399999999999999
```

```
differential_evolution step 576: f(x)= 8.399999999999999
differential_evolution step 577: f(x)= 8.399999999999999
differential_evolution step 578: f(x)= 8.399999999999999
differential_evolution step 579: f(x)= 8.399999999999999
differential_evolution step 580: f(x)= 8.399999999999999
differential_evolution step 581: f(x)= 8.399999999999999
differential_evolution step 582: f(x)= 8.399999999999999
differential_evolution step 583: f(x)= 8.399999999999999
differential_evolution step 584: f(x)= 8.399999999999999
differential_evolution step 585: f(x)= 8.399999999999999
differential_evolution step 586: f(x)= 8.399999999999999
differential_evolution step 587: f(x)= 8.399999999999999
differential_evolution step 588: f(x)= 8.399999999999999
differential_evolution step 589: f(x)= 8.399999999999999
differential_evolution step 590: f(x)= 8.399999999999999
differential_evolution step 591: f(x)= 8.399999999999999
differential_evolution step 592: f(x)= 8.399999999999999
differential_evolution step 593: f(x)= 8.399999999999999
differential_evolution step 594: f(x)= 8.399999999999999
differential_evolution step 595: f(x)= 8.399999999999999
differential_evolution step 596: f(x)= 8.399999999999999
differential_evolution step 597: f(x)= 8.399999999999999
differential_evolution step 598: f(x)= 8.399999999999999
differential_evolution step 599: f(x)= 8.399999999999999
differential_evolution step 600: f(x)= 8.399999999999999
differential_evolution step 601: f(x)= 8.399999999999999
differential_evolution step 602: f(x)= 8.399999999999999
differential_evolution step 603: f(x)= 8.399999999999999
differential_evolution step 604: f(x)= 8.399999999999999
differential_evolution step 605: f(x)= 8.399999999999999
differential_evolution step 606: f(x)= 8.399999999999999
differential_evolution step 607: f(x)= 8.399999999999999
differential_evolution step 608: f(x)= 8.399999999999999
differential_evolution step 609: f(x)= 8.399999999999999
differential_evolution step 610: f(x)= 8.399999999999999
differential_evolution step 611: f(x)= 8.399999999999999
differential_evolution step 612: f(x)= 8.3
differential_evolution step 613: f(x)= 8.3
differential_evolution step 614: f(x)= 8.3
differential_evolution step 615: f(x)= 8.3
differential_evolution step 616: f(x)= 8.3
differential_evolution step 617: f(x)= 8.3
differential_evolution step 618: f(x)= 8.3
differential_evolution step 619: f(x)= 8.3
differential_evolution step 620: f(x)= 8.3
differential_evolution step 621: f(x)= 8.3
differential_evolution step 622: f(x)= 8.3
differential_evolution step 623: f(x)= 8.3
differential_evolution step 624: f(x)= 8.3
differential_evolution step 625: f(x)= 8.3
differential_evolution step 626: f(x)= 8.3
differential_evolution step 627: f(x)= 8.3
differential_evolution step 628: f(x)= 8.3
differential_evolution step 629: f(x)= 8.3
differential_evolution step 630: f(x)= 8.3
differential_evolution step 631: f(x)= 8.3
differential_evolution step 632: f(x)= 8.3
differential_evolution step 633: f(x)= 8.3
differential_evolution step 634: f(x)= 8.3
differential_evolution step 635: f(x)= 8.3
differential_evolution step 636: f(x)= 8.3
differential_evolution step 637: f(x)= 8.3
differential_evolution step 638: f(x)= 8.3
differential_evolution step 639: f(x)= 8.3
```

```
differential_evolution step 640: f(x)= 8.3
differential_evolution step 641: f(x)= 8.3
differential_evolution step 642: f(x)= 8.3
differential_evolution step 643: f(x)= 8.3
differential_evolution step 644: f(x)= 8.3
differential_evolution step 645: f(x)= 8.3
differential_evolution step 646: f(x)= 8.3
differential_evolution step 647: f(x)= 8.3
differential_evolution step 648: f(x)= 8.3
differential_evolution step 649: f(x)= 8.3
differential_evolution step 650: f(x)= 8.3
differential_evolution step 651: f(x)= 8.3
differential_evolution step 652: f(x)= 8.3
differential_evolution step 653: f(x)= 8.3
differential_evolution step 654: f(x)= 8.3
differential_evolution step 655: f(x)= 8.3
differential_evolution step 656: f(x)= 8.3
differential_evolution step 657: f(x)= 8.3
differential_evolution step 658: f(x)= 8.3
differential_evolution step 659: f(x)= 8.200000000000001
differential_evolution step 660: f(x)= 8.200000000000001
differential_evolution step 661: f(x)= 8.200000000000001
differential_evolution step 662: f(x)= 8.200000000000001
differential_evolution step 663: f(x)= 8.200000000000001
differential_evolution step 664: f(x)= 8.200000000000001
differential_evolution step 665: f(x)= 8.200000000000001
differential_evolution step 666: f(x)= 8.200000000000001
differential_evolution step 667: f(x)= 8.200000000000001
differential_evolution step 668: f(x)= 8.200000000000001
differential_evolution step 669: f(x)= 8.200000000000001
differential_evolution step 670: f(x)= 8.200000000000001
differential_evolution step 671: f(x)= 8.200000000000001
differential_evolution step 672: f(x)= 8.200000000000001
differential_evolution step 673: f(x)= 8.200000000000001
differential_evolution step 674: f(x)= 8.200000000000001
differential_evolution step 675: f(x)= 8.200000000000001
differential_evolution step 676: f(x)= 8.200000000000001
differential_evolution step 677: f(x)= 8.200000000000001
differential_evolution step 678: f(x)= 8.200000000000001
differential_evolution step 679: f(x)= 8.200000000000001
differential_evolution step 680: f(x)= 8.200000000000001
differential_evolution step 681: f(x)= 8.200000000000001
differential_evolution step 682: f(x)= 8.200000000000001
differential_evolution step 683: f(x)= 8.200000000000001
differential_evolution step 684: f(x)= 8.200000000000001
differential_evolution step 685: f(x)= 8.200000000000001
differential_evolution step 686: f(x)= 8.200000000000001
differential_evolution step 687: f(x)= 8.200000000000001
differential_evolution step 688: f(x)= 8.200000000000001
differential_evolution step 689: f(x)= 8.200000000000001
differential_evolution step 690: f(x)= 8.200000000000001
differential_evolution step 691: f(x)= 8.200000000000001
differential_evolution step 692: f(x)= 8.200000000000001
differential_evolution step 693: f(x)= 7.500000000000001
differential_evolution step 694: f(x)= 7.500000000000001
differential_evolution step 695: f(x)= 7.500000000000001
differential_evolution step 696: f(x)= 7.500000000000001
differential_evolution step 697: f(x)= 7.500000000000001
differential_evolution step 698: f(x)= 7.500000000000001
differential_evolution step 699: f(x)= 7.500000000000001
differential_evolution step 700: f(x)= 7.500000000000001
differential_evolution step 701: f(x)= 7.500000000000001
differential_evolution step 702: f(x)= 7.500000000000001
differential_evolution step 703: f(x)= 7.500000000000001
```

```
differential_evolution step 704: f(x)= 7.500000000000001
differential_evolution step 705: f(x)= 7.500000000000001
differential_evolution step 706: f(x)= 7.500000000000001
differential_evolution step 707: f(x)= 7.500000000000001
differential_evolution step 708: f(x)= 7.500000000000001
differential_evolution step 709: f(x)= 7.500000000000001
differential_evolution step 710: f(x)= 7.500000000000001
differential_evolution step 711: f(x)= 7.500000000000001
differential_evolution step 712: f(x)= 7.500000000000001
differential_evolution step 713: f(x)= 7.500000000000001
differential_evolution step 714: f(x)= 7.500000000000001
differential_evolution step 715: f(x)= 7.500000000000001
differential_evolution step 716: f(x)= 7.500000000000001
differential_evolution step 717: f(x)= 7.500000000000001
differential_evolution step 718: f(x)= 7.500000000000001
differential_evolution step 719: f(x)= 7.500000000000001
differential_evolution step 720: f(x)= 7.500000000000001
differential_evolution step 721: f(x)= 7.500000000000001
differential_evolution step 722: f(x)= 7.500000000000001
differential_evolution step 723: f(x)= 7.500000000000001
differential_evolution step 724: f(x)= 7.500000000000001
differential_evolution step 725: f(x)= 7.500000000000001
differential_evolution step 726: f(x)= 7.500000000000001
differential_evolution step 727: f(x)= 7.500000000000001
differential_evolution step 728: f(x)= 7.500000000000001
differential_evolution step 729: f(x)= 7.500000000000001
differential_evolution step 730: f(x)= 7.500000000000001
differential_evolution step 731: f(x)= 7.500000000000001
differential_evolution step 732: f(x)= 7.500000000000001
differential_evolution step 733: f(x)= 7.500000000000001
differential_evolution step 734: f(x)= 7.500000000000001
differential_evolution step 735: f(x)= 7.500000000000001
differential_evolution step 736: f(x)= 7.500000000000001
differential_evolution step 737: f(x)= 7.500000000000001
differential_evolution step 738: f(x)= 7.500000000000001
differential_evolution step 739: f(x)= 7.500000000000001
differential_evolution step 740: f(x)= 7.500000000000001
differential_evolution step 741: f(x)= 7.500000000000001
differential_evolution step 742: f(x)= 7.500000000000001
differential_evolution step 743: f(x)= 7.500000000000001
differential_evolution step 744: f(x)= 7.500000000000001
differential_evolution step 745: f(x)= 7.500000000000001
differential_evolution step 746: f(x)= 7.500000000000001
differential_evolution step 747: f(x)= 7.500000000000001
differential_evolution step 748: f(x)= 7.500000000000001
differential_evolution step 749: f(x)= 7.500000000000001
differential_evolution step 750: f(x)= 7.500000000000001
differential_evolution step 751: f(x)= 7.500000000000001
differential_evolution step 752: f(x)= 7.500000000000001
differential_evolution step 753: f(x)= 7.500000000000001
differential_evolution step 754: f(x)= 7.500000000000001
differential_evolution step 755: f(x)= 7.500000000000001
differential_evolution step 756: f(x)= 7.500000000000001
differential_evolution step 757: f(x)= 7.500000000000001
differential_evolution step 758: f(x)= 7.500000000000001
differential_evolution step 759: f(x)= 7.500000000000001
differential_evolution step 760: f(x)= 7.500000000000001
differential_evolution step 761: f(x)= 7.500000000000001
differential_evolution step 762: f(x)= 7.500000000000001
differential_evolution step 763: f(x)= 7.500000000000001
differential_evolution step 764: f(x)= 7.500000000000001
differential_evolution step 765: f(x)= 7.500000000000001
differential_evolution step 766: f(x)= 6.9
differential_evolution step 767: f(x)= 6.9
```

```
differential_evolution step 768: f(x)= 6.9
differential_evolution step 769: f(x)= 6.9
differential_evolution step 770: f(x)= 6.9
differential_evolution step 771: f(x)= 6.9
differential_evolution step 772: f(x)= 6.9
differential_evolution step 773: f(x)= 6.9
differential_evolution step 774: f(x)= 6.9
differential_evolution step 775: f(x)= 6.9
differential_evolution step 776: f(x)= 6.9
differential_evolution step 777: f(x)= 6.9
differential_evolution step 778: f(x)= 6.9
differential_evolution step 779: f(x)= 6.9
differential_evolution step 780: f(x)= 6.9
differential_evolution step 781: f(x)= 6.9
differential_evolution step 782: f(x)= 6.9
differential_evolution step 783: f(x)= 6.9
differential_evolution step 784: f(x)= 6.9
differential_evolution step 785: f(x)= 6.9
differential_evolution step 786: f(x)= 6.9
differential_evolution step 787: f(x)= 6.9
differential_evolution step 788: f(x)= 6.9
differential_evolution step 789: f(x)= 6.9
differential_evolution step 790: f(x)= 6.9
differential_evolution step 791: f(x)= 6.9
differential_evolution step 792: f(x)= 6.9
differential_evolution step 793: f(x)= 6.9
differential_evolution step 794: f(x)= 6.9
differential_evolution step 795: f(x)= 6.9
differential_evolution step 796: f(x)= 6.9
differential_evolution step 797: f(x)= 6.9
differential_evolution step 798: f(x)= 6.9
differential_evolution step 799: f(x)= 6.9
differential_evolution step 800: f(x)= 6.9
differential_evolution step 801: f(x)= 6.9
differential_evolution step 802: f(x)= 6.9
differential_evolution step 803: f(x)= 6.9
differential_evolution step 804: f(x)= 6.9
differential_evolution step 805: f(x)= 6.9
differential_evolution step 806: f(x)= 6.9
differential_evolution step 807: f(x)= 6.9
differential_evolution step 808: f(x)= 6.9
differential_evolution step 809: f(x)= 6.9
differential_evolution step 810: f(x)= 6.9
differential_evolution step 811: f(x)= 6.9
differential_evolution step 812: f(x)= 6.9
differential_evolution step 813: f(x)= 6.9
differential_evolution step 814: f(x)= 6.9
differential_evolution step 815: f(x)= 6.9
differential_evolution step 816: f(x)= 6.9
differential_evolution step 817: f(x)= 6.9
differential_evolution step 818: f(x)= 6.9
differential_evolution step 819: f(x)= 6.9
differential_evolution step 820: f(x)= 6.9
differential_evolution step 821: f(x)= 6.9
differential_evolution step 822: f(x)= 6.9
differential_evolution step 823: f(x)= 6.9
differential_evolution step 824: f(x)= 6.9
differential_evolution step 825: f(x)= 6.9
differential_evolution step 826: f(x)= 6.9
differential_evolution step 827: f(x)= 6.9
differential_evolution step 828: f(x)= 6.9
differential_evolution step 829: f(x)= 6.9
differential_evolution step 830: f(x)= 6.9
differential_evolution step 831: f(x)= 6.9
```

```
differential_evolution step 832: f(x)= 6.9
differential_evolution step 833: f(x)= 6.9
differential_evolution step 834: f(x)= 6.9
differential_evolution step 835: f(x)= 6.9
differential_evolution step 836: f(x)= 6.9
differential_evolution step 837: f(x)= 6.9
differential_evolution step 838: f(x)= 6.9
differential_evolution step 839: f(x)= 6.9
differential_evolution step 840: f(x)= 6.9
differential_evolution step 841: f(x)= 6.9
differential_evolution step 842: f(x)= 6.9
differential_evolution step 843: f(x)= 6.9
differential_evolution step 844: f(x)= 6.9
differential_evolution step 845: f(x)= 6.9
differential_evolution step 846: f(x)= 6.9
differential_evolution step 847: f(x)= 6.9
differential_evolution step 848: f(x)= 6.9
differential_evolution step 849: f(x)= 6.9
differential_evolution step 850: f(x)= 6.9
differential_evolution step 851: f(x)= 6.9
differential_evolution step 852: f(x)= 6.9
differential_evolution step 853: f(x)= 6.9
differential_evolution step 854: f(x)= 6.9
differential_evolution step 855: f(x)= 6.9
differential_evolution step 856: f(x)= 6.9
differential_evolution step 857: f(x)= 6.9
differential_evolution step 858: f(x)= 6.9
differential_evolution step 859: f(x)= 6.9
differential_evolution step 860: f(x)= 6.9
differential_evolution step 861: f(x)= 6.9
differential_evolution step 862: f(x)= 6.9
differential_evolution step 863: f(x)= 6.9
differential_evolution step 864: f(x)= 6.9
differential_evolution step 865: f(x)= 6.9
differential_evolution step 866: f(x)= 6.9
differential_evolution step 867: f(x)= 6.9
differential_evolution step 868: f(x)= 6.9
differential_evolution step 869: f(x)= 6.9
differential_evolution step 870: f(x)= 6.9
differential_evolution step 871: f(x)= 6.9
differential_evolution step 872: f(x)= 6.9
differential_evolution step 873: f(x)= 6.9
differential_evolution step 874: f(x)= 6.9
differential_evolution step 875: f(x)= 6.9
differential_evolution step 876: f(x)= 6.9
differential_evolution step 877: f(x)= 6.9
differential_evolution step 878: f(x)= 6.9
differential_evolution step 879: f(x)= 6.9
differential_evolution step 880: f(x)= 6.9
differential_evolution step 881: f(x)= 6.9
differential_evolution step 882: f(x)= 6.9
differential_evolution step 883: f(x)= 6.9
differential_evolution step 884: f(x)= 6.9
differential_evolution step 885: f(x)= 6.9
differential_evolution step 886: f(x)= 6.9
differential_evolution step 887: f(x)= 6.9
differential_evolution step 888: f(x)= 6.9
differential_evolution step 889: f(x)= 6.9
differential_evolution step 890: f(x)= 6.9
differential_evolution step 891: f(x)= 6.9
differential_evolution step 892: f(x)= 6.9
differential_evolution step 893: f(x)= 6.9
differential_evolution step 894: f(x)= 6.9
differential_evolution step 895: f(x)= 6.9
```

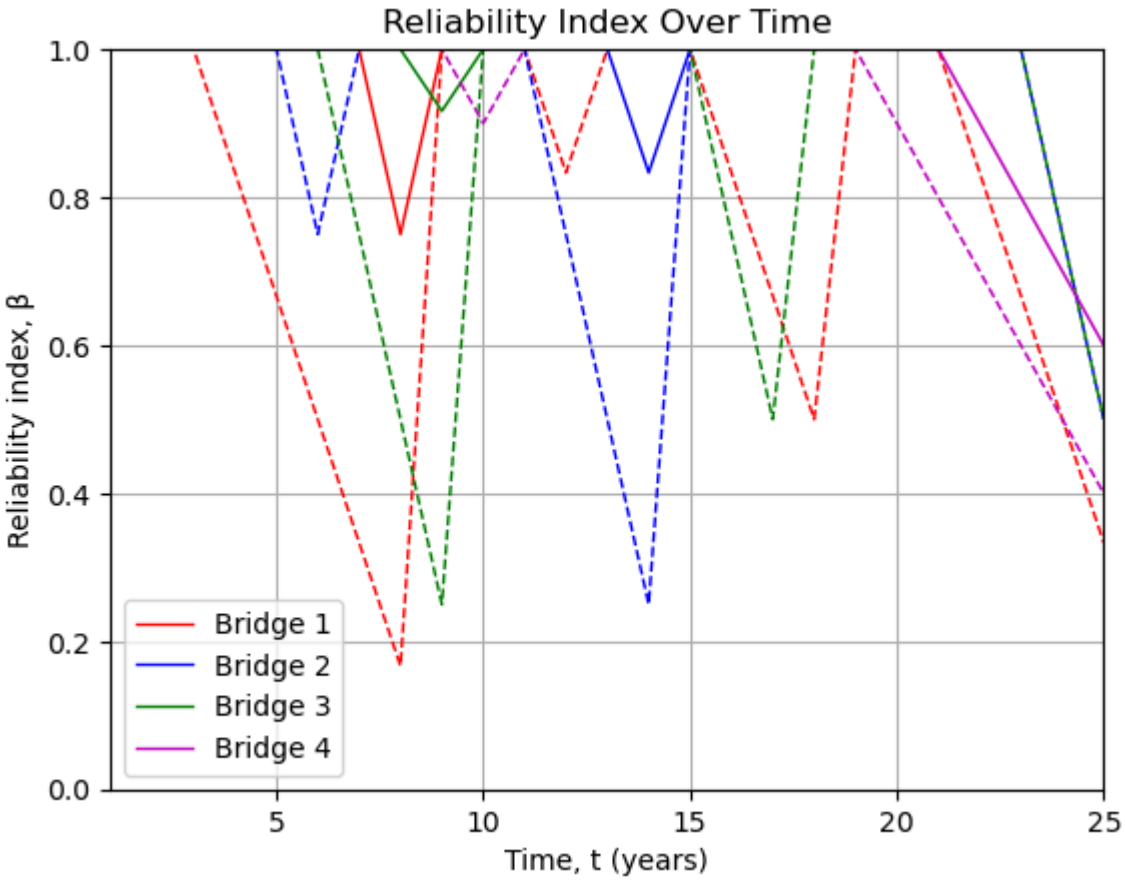
```
differential_evolution step 896: f(x)= 6.9
differential_evolution step 897: f(x)= 6.9
differential_evolution step 898: f(x)= 6.9
differential_evolution step 899: f(x)= 6.9
differential_evolution step 900: f(x)= 6.9
differential_evolution step 901: f(x)= 6.9
differential_evolution step 902: f(x)= 6.9
differential_evolution step 903: f(x)= 6.9
differential_evolution step 904: f(x)= 6.9
differential_evolution step 905: f(x)= 6.9
differential_evolution step 906: f(x)= 6.9
differential_evolution step 907: f(x)= 6.9
differential_evolution step 908: f(x)= 6.9
differential_evolution step 909: f(x)= 6.9
differential_evolution step 910: f(x)= 6.9
differential_evolution step 911: f(x)= 6.9
differential_evolution step 912: f(x)= 6.9
differential_evolution step 913: f(x)= 6.9
differential_evolution step 914: f(x)= 6.9
differential_evolution step 915: f(x)= 6.9
differential_evolution step 916: f(x)= 6.9
differential_evolution step 917: f(x)= 6.9
differential_evolution step 918: f(x)= 6.9
differential_evolution step 919: f(x)= 6.9
differential_evolution step 920: f(x)= 6.9
differential_evolution step 921: f(x)= 6.9
differential_evolution step 922: f(x)= 6.9
differential_evolution step 923: f(x)= 6.9
differential_evolution step 924: f(x)= 6.9
differential_evolution step 925: f(x)= 6.9
differential_evolution step 926: f(x)= 6.9
differential_evolution step 927: f(x)= 6.9
differential_evolution step 928: f(x)= 6.9
differential_evolution step 929: f(x)= 6.9
differential_evolution step 930: f(x)= 6.9
differential_evolution step 931: f(x)= 6.9
differential_evolution step 932: f(x)= 6.9
differential_evolution step 933: f(x)= 6.9
differential_evolution step 934: f(x)= 6.9
differential_evolution step 935: f(x)= 6.9
differential_evolution step 936: f(x)= 6.9
differential_evolution step 937: f(x)= 6.9
differential_evolution step 938: f(x)= 6.9
differential_evolution step 939: f(x)= 6.9
differential_evolution step 940: f(x)= 6.9
differential_evolution step 941: f(x)= 6.9
differential_evolution step 942: f(x)= 6.9
differential_evolution step 943: f(x)= 6.9
differential_evolution step 944: f(x)= 6.9
differential_evolution step 945: f(x)= 6.9
differential_evolution step 946: f(x)= 6.9
differential_evolution step 947: f(x)= 6.9
differential_evolution step 948: f(x)= 6.9
differential_evolution step 949: f(x)= 6.9
differential_evolution step 950: f(x)= 6.9
differential_evolution step 951: f(x)= 6.9
differential_evolution step 952: f(x)= 6.9
differential_evolution step 953: f(x)= 6.9
differential_evolution step 954: f(x)= 6.9
differential_evolution step 955: f(x)= 6.9
differential_evolution step 956: f(x)= 6.9
differential_evolution step 957: f(x)= 6.9
differential_evolution step 958: f(x)= 6.9
differential_evolution step 959: f(x)= 6.9
```

```
differential_evolution step 960: f(x)= 6.9
differential_evolution step 961: f(x)= 6.9
differential_evolution step 962: f(x)= 6.9
differential_evolution step 963: f(x)= 6.9
differential_evolution step 964: f(x)= 6.9
differential_evolution step 965: f(x)= 6.9
differential_evolution step 966: f(x)= 6.9
differential_evolution step 967: f(x)= 6.9
differential_evolution step 968: f(x)= 6.9
differential_evolution step 969: f(x)= 6.9
differential_evolution step 970: f(x)= 6.9
differential_evolution step 971: f(x)= 6.9
differential_evolution step 972: f(x)= 6.9
differential_evolution step 973: f(x)= 6.9
differential_evolution step 974: f(x)= 6.9
differential_evolution step 975: f(x)= 6.9
differential_evolution step 976: f(x)= 6.9
differential_evolution step 977: f(x)= 6.9
differential_evolution step 978: f(x)= 6.9
differential_evolution step 979: f(x)= 6.9
differential_evolution step 980: f(x)= 6.9
differential_evolution step 981: f(x)= 6.9
differential_evolution step 982: f(x)= 6.9
differential_evolution step 983: f(x)= 6.9
differential_evolution step 984: f(x)= 6.9
differential_evolution step 985: f(x)= 6.9
differential_evolution step 986: f(x)= 6.9
differential_evolution step 987: f(x)= 6.9
differential_evolution step 988: f(x)= 6.9
differential_evolution step 989: f(x)= 6.9
differential_evolution step 990: f(x)= 6.9
differential_evolution step 991: f(x)= 6.9
differential_evolution step 992: f(x)= 6.9
differential_evolution step 993: f(x)= 6.9
differential_evolution step 994: f(x)= 6.9
differential_evolution step 995: f(x)= 6.9
differential_evolution step 996: f(x)= 6.9
differential_evolution step 997: f(x)= 6.9
differential_evolution step 998: f(x)= 6.9
differential_evolution step 999: f(x)= 6.9
differential_evolution step 1000: f(x)= 6.9
differential_evolution step 1001: f(x)= 6.9
differential_evolution step 1002: f(x)= 6.9
differential_evolution step 1003: f(x)= 6.9
differential_evolution step 1004: f(x)= 6.9
differential_evolution step 1005: f(x)= 6.9
differential_evolution step 1006: f(x)= 6.9
differential_evolution step 1007: f(x)= 6.9
differential_evolution step 1008: f(x)= 6.9
differential_evolution step 1009: f(x)= 6.9
differential_evolution step 1010: f(x)= 6.9
differential_evolution step 1011: f(x)= 6.9
differential_evolution step 1012: f(x)= 6.9
differential_evolution step 1013: f(x)= 6.9
differential_evolution step 1014: f(x)= 6.9
differential_evolution step 1015: f(x)= 6.9
differential_evolution step 1016: f(x)= 6.9
differential_evolution step 1017: f(x)= 6.9
differential_evolution step 1018: f(x)= 6.9
differential_evolution step 1019: f(x)= 6.9
differential_evolution step 1020: f(x)= 6.9
differential_evolution step 1021: f(x)= 6.9
differential_evolution step 1022: f(x)= 6.9
differential_evolution step 1023: f(x)= 6.9
```

```
differential_evolution step 1024: f(x)= 6.9
differential_evolution step 1025: f(x)= 6.9
differential_evolution step 1026: f(x)= 6.9
differential_evolution step 1027: f(x)= 6.9
differential_evolution step 1028: f(x)= 6.9
differential_evolution step 1029: f(x)= 6.9
differential_evolution step 1030: f(x)= 6.9
differential_evolution step 1031: f(x)= 6.9
differential_evolution step 1032: f(x)= 6.9
differential_evolution step 1033: f(x)= 6.9
differential_evolution step 1034: f(x)= 6.9
             message: Optimization terminated successfully.
             success: True
                 fun: 6.9
                   x: [ 8.000e+00  6.000e+00 ...  8.000e+00  0.000e+00]
                 nit: 1034
                nfev: 11133
          population: [[ 8.000e+00  6.000e+00 ...  8.000e+00  0.000e+00]
                       [ 7.000e+00  7.000e+00 ...  7.000e+00  0.000e+00]
                       ...
                       [ 8.000e+00  7.000e+00 ...  8.000e+00  0.000e+00]
                       [ 8.000e+00  7.000e+00 ...  9.000e+00  0.000e+00]]
 population_energies: [ 6.900e+00  6.900e+00 ...  6.900e+00  6.900e+00]
              constr: [array([ 0.000e+00,  0.000e+00, ...,  0.000e+00,
                               0.000e+00])]
    constr_violation: 0.0
                maxcv: 0.0
```

```
Yopt = result.x
OptInterv = cumulateFreq(Yopt.reshape((nt, nb)).T)
X_bt = Frequency2Schedule(Yopt, nb, nt, Thoriz)
beta1_bt = MaintenanceApplication(beta_bt01, X_bt, Thoriz)
beta2_bt = MaintenanceApplication(beta_bt02, X_bt, Thoriz)

print(f'Min Cost = {result.fun:.2f}')
ninterv = np.sum(CountNinterv(Yopt, nt, nb))
print(f'Number of interventions = {ninterv}')
c = nonlcon(Yopt)
BudgetIssues = np.sum(c[-Thoriz - 1:-1] > 0)
print(f'Number of time intervals with exceeding budget = {BudgetIssues}')
reliabIssues = c[-1]
print(f'Number of time intervals with reliability issues = {reliabIssues}')

plt.figure()
h = []
leg = []
for b in range(nb):
    h.append(plt.plot(range(1, Thoriz + 1), beta1_bt[b, :], f'--{col[b]}', linewidth=1))
    plt.plot(range(1, Thoriz + 1), beta2_bt[b, :], f'-{col[b]}', linewidth=1, label = f'B
plt.legend(loc='best')
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()
```

```
Min Cost = 6.90
Number of interventions = 9
Number of time intervals with exceeding budget = 0
Number of time intervals with reliability issues = 0.0
```



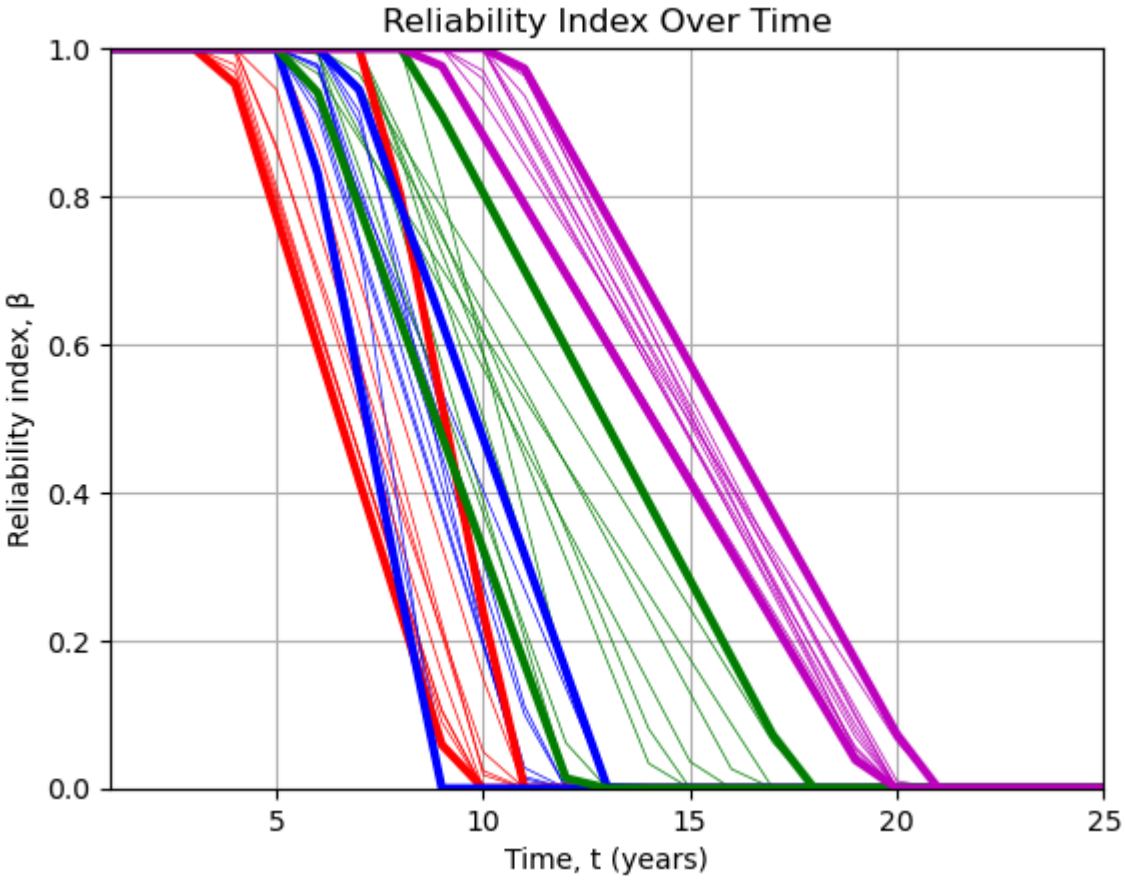
Stochastic

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import differential_evolution
import scipy as sp

def reliability(beta0_b, d_b, L_b, Thoriz):
    # --- Reliability estimation
    # beta_bt: reliability curve over time for bridge b
    # beta0_b: Initial reliability of bridge b at t=0
    # d_b, L_b: when the degradation starts and reaches beta=0 for each bridge
    # Thoriz: maximum time studied
    nb = len(beta0_b)
    beta_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        #horziontal branch
        timesteps = int(np.floor(d_b[b]))
        beta_bt[b, :timesteps] = beta0_b[b]

        #degradation till L_b
        servicelife = int(np.floor(L_b[b]))
        t = np.arange(timesteps + 1, servicelife + 1)
        beta_bt[b, timesteps:servicelife] = beta0_b[b] / (L_b[b] - d_b[b]) * (L_b[b] - t)

    return beta_bt

def MaintenanceApplication(beta_bt, X_bt, Thoriz):
    # --- Given a maintenance strategy given by X_bt, where 1 means that bridge
    # b undergoes maintenance at time t, this function gives the updated
    # reliability profile
    updatedbeta_bt = beta_bt.copy()
    InterventionTimes = np.where(np.sum(X_bt, axis=0) > 0)[0]

    for t in InterventionTimes:
        intervbrid = np.where(X_bt[:, t] == 1)[0]
        timecells = Thoriz - t - 1
        updatedbeta_bt[intervbrid, t+1:Thoriz] = beta_bt[intervbrid, :timecells]

    return updatedbeta_bt

def Frequency2Schedule(y_b, nb, nt, Thoriz):
    # Transforms a vector indicating the frequency to a schedule matrix [0/1]
    freq_b = y_b.reshape((nt, nb)).T
    cumFreq_b = cumulateFreq(freq_b)
    X_bt = np.zeros((nb, Thoriz))

    for b in range(nb):
        idx = cumFreq_b[b, cumFreq_b[b, :] != 0].astype(int)
        idx = idx[idx <= Thoriz]
        if len(idx) > 0:
            X_bt[b, idx - 1] = 1

    return X_bt

def cumulateFreq(freq_b):
    # creates the cumulative frequency
    nb, nt = freq_b.shape
    cumFreq_b = np.zeros((nb, nt))

    for b in range(nb):
        f_b = freq_b[b, freq_b[b, :] != 0]
        cumFreq_b[b, :len(f_b)] = np.cumsum(f_b)
```

```
    return cumFreq_b

def CountNinterv(y_b, nt, nb):
    # from vector y_b, it computes the number of interventions per bridge
    y_b01 = y_b != 0
    freq_b = y_b01.reshape((nt, nb)).T
    ninterv_b = np.sum(freq_b, axis=1)
    return ninterv_b

def ProbFulfilment(beta_bti0, X_bt, Thoriz):
    nb, _, nMCS = beta_bti0.shape

    # --- MCS
    Achivement_bi = np.zeros((nb, nMCS))
    for i in range(nMCS):
        beta_bt = MaintenanceApplication(beta_bti0[:, :, i], X_bt, Thoriz)
        failures_b = np.sum(beta_bt == 0, axis=1)
        Achivement_bi[:, i] = (failures_b == 0)

    ProbRel_b = np.sum(Achivement_bi, axis=1) / nMCS
    return ProbRel_b
```

```
Thoriz = 25
# for each bridge b:  beta0 mu_d(years) sigma_d(years) mu_L(years) sigma_L(years) C(Meuro
data = np.array([
    [1, 5, 2 / 1.96, 10, 1 / 1.96, 0.6],
    [1, 6, 1 / 1.96, 11, 2 / 1.96, 0.8],
    [1, 7, 1 / 1.96, 15, 5 / 1.96, 1.0],
    [1, 10, 1 / 1.96, 20, 1 / 1.96, 0.7]
])
beta0_b, mud_b, sigd_b, muL_b, sigL_b, C_b = data[:, 0], data[:, 1], data[:, 2], data[:,

# Annual resources (M euro)
R = 1.4

# Sample size of the probabilistic analysis
nMCS=10

# --- Initial reliability estimation
nb = len(beta0_b)  # number of bridges

# nMCS cases of d and L
d_bi, L_bi = np.zeros((nb, nMCS)), np.zeros((nb, nMCS))
for b in range(nb):
    d_bi[b, :] = np.random.normal(mud_b[b], sigd_b[b], nMCS)
    L_bi[b, :] = np.random.normal(muL_b[b], sigL_b[b], nMCS)

beta_bti0 = np.zeros((nb, Thoriz, nMCS))
for i in range(nMCS):
    beta_bti0[:, :, i] = reliability(beta0_b, d_bi[:, i], L_bi[:, i], Thoriz)  # reliabil

plt.figure()
col = ['r', 'b', 'g', 'm']
for i in range(nMCS):
    for b in range(nb):
        plt.plot(range(1, Thoriz + 1), beta_bti0[b, :, i], f'-{col[b]}', linewidth=0.5)
for b in range(nb):
    minbeta_t0 = reliability([beta0_b[b]], [np.min(d_bi[b, :])], [np.min(L_bi[b, :])], Th
    maxbeta_t0 = reliability([beta0_b[b]], [np.max(d_bi[b, :])], [np.max(L_bi[b, :])], Th
    plt.plot(range(1, Thoriz + 1), minbeta_t0, f'-{col[b]}', linewidth=3)
    plt.plot(range(1, Thoriz + 1), maxbeta_t0, f'-{col[b]}', linewidth=3)
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()

# Deterministic Optimization
# Variables ->  e.g., freq=[10 2 0 0 ; 5, 5 0 0; 0 0 0 0];

maxInter_b = np.ceil(Thoriz / np.min(L_bi, axis=1)).astype(int) + 1  # max number of inte
nt = max(maxInter_b)  # to dimension the matrix, max interventions experienced by the wor
nvars = nb * nt #number of design variables
integers = np.array([True] * nvars) #all variables are integers

#  Range of definition: [0,max number of years with no interventions==L_b]

lb = np.zeros((nvars, 1))
# upper bound
ub = lb.copy()
pos = 0
for b in range(nb):
    # This guarantees interventions freq. within the service life (if conducted)
```

```
    ub[pos:pos + maxInter_b[b]] = np.max(L_bi[b, :], axis=0) - 1  # the largest service l
    # this forces zeros after the max number of interventions expected
    if maxInter_b[b] < nb:
        ub[pos + maxInter_b[b]:pos + nb - 1] = 0
    pos += nt
ub = np.ceil(ub)

def nonlcon(y_b):
    #  From frequency to a schedule matrix [0/1]
    X_bt = Frequency2Schedule(y_b, nb, nt, Thoriz)

    # Constraint 1 >>> the last intervention is within Thoriz --> LastInterv<Thoriz --> L
    lastInterv_b = np.max(cumulateFreq(y_b.reshape((nt, nb)).T), axis=1)
    c1 = lastInterv_b - Thoriz - 1

    # Constraint 2 >>> C_b*X_bt <= R   for all t
    c2 = np.sum(C_b[:, None] * X_bt, axis=0) - R

    # Constraint 3 >>> Prob(beta_bt>0)>=0.90  --> sum(sum(beta_bt))<=0 for all b
    ProbRel_b = ProbFulfilment(beta_bti0, X_bt, Thoriz)
    c3 = 0.90 - ProbRel_b

    c = np.concatenate([c1, c2, c3])
    return c

cons = sp.optimize.NonlinearConstraint(nonlcon, np.array([-np.inf] * (nb + Thoriz + nb)),

def fun(y_b):
    #  minimize cost trying to go for the largest time between interventions (== max y_b)
    ninterv_b = CountNinterv(y_b, nt, nb)
    return np.dot(C_b, ninterv_b)

bounds = [[lb[i][0], ub[i][0]] for i in range(nvars)]
```



```
result = differential_evolution(fun, bounds, constraints=cons, integrality=integers, disp
print(result.x)
```

```
differential_evolution step 1: f(x)= 11.7
differential_evolution step 2: f(x)= 11.7
differential_evolution step 3: f(x)= 11.7
differential_evolution step 4: f(x)= 9.3
differential_evolution step 5: f(x)= 9.3
differential_evolution step 6: f(x)= 9.3
differential_evolution step 7: f(x)= 9.3
differential_evolution step 8: f(x)= 9.3
differential_evolution step 9: f(x)= 9.3
differential_evolution step 10: f(x)= 9.3
differential_evolution step 11: f(x)= 9.3
differential_evolution step 12: f(x)= 9.3
differential_evolution step 13: f(x)= 9.3
differential_evolution step 14: f(x)= 9.3
differential_evolution step 15: f(x)= 9.3
differential_evolution step 16: f(x)= 9.3
differential_evolution step 17: f(x)= 9.3
differential_evolution step 18: f(x)= 9.3
differential_evolution step 19: f(x)= 9.3
differential_evolution step 20: f(x)= 9.3
differential_evolution step 21: f(x)= 9.3
differential_evolution step 22: f(x)= 9.3
differential_evolution step 23: f(x)= 9.3
differential_evolution step 24: f(x)= 9.3
differential_evolution step 25: f(x)= 9.3
differential_evolution step 26: f(x)= 9.3
differential_evolution step 27: f(x)= 9.3
differential_evolution step 28: f(x)= 9.3
differential_evolution step 29: f(x)= 9.3
differential_evolution step 30: f(x)= 9.3
differential_evolution step 31: f(x)= 9.3
differential_evolution step 32: f(x)= 9.3
differential_evolution step 33: f(x)= 9.3
differential_evolution step 34: f(x)= 9.3
differential_evolution step 35: f(x)= 9.3
differential_evolution step 36: f(x)= 9.3
differential_evolution step 37: f(x)= 9.3
differential_evolution step 38: f(x)= 9.3
differential_evolution step 39: f(x)= 9.3
differential_evolution step 40: f(x)= 9.3
differential_evolution step 41: f(x)= 9.3
differential_evolution step 42: f(x)= 9.3
differential_evolution step 43: f(x)= 9.3
differential_evolution step 44: f(x)= 9.3
differential_evolution step 45: f(x)= 9.3
differential_evolution step 46: f(x)= 9.3
differential_evolution step 47: f(x)= 9.3
differential_evolution step 48: f(x)= 9.3
differential_evolution step 49: f(x)= 9.3
differential_evolution step 50: f(x)= 9.3
differential_evolution step 51: f(x)= 9.3
differential_evolution step 52: f(x)= 9.3
differential_evolution step 53: f(x)= 9.3
differential_evolution step 54: f(x)= 9.3
differential_evolution step 55: f(x)= 9.3
differential_evolution step 56: f(x)= 9.3
differential_evolution step 57: f(x)= 9.3
differential_evolution step 58: f(x)= 9.3
differential_evolution step 59: f(x)= 9.3
differential_evolution step 60: f(x)= 9.3
differential_evolution step 61: f(x)= 9.3
differential_evolution step 62: f(x)= 9.3
differential_evolution step 63: f(x)= 9.3
```

```
differential_evolution step 64: f(x)= 9.3
differential_evolution step 65: f(x)= 9.3
differential_evolution step 66: f(x)= 9.3
differential_evolution step 67: f(x)= 9.3
differential_evolution step 68: f(x)= 9.3
differential_evolution step 69: f(x)= 9.3
differential_evolution step 70: f(x)= 9.3
differential_evolution step 71: f(x)= 9.3
differential_evolution step 72: f(x)= 9.3
differential_evolution step 73: f(x)= 9.3
differential_evolution step 74: f(x)= 9.3
differential_evolution step 75: f(x)= 9.3
differential_evolution step 76: f(x)= 9.3
differential_evolution step 77: f(x)= 9.3
differential_evolution step 78: f(x)= 9.3
differential_evolution step 79: f(x)= 9.3
differential_evolution step 80: f(x)= 9.3
differential_evolution step 81: f(x)= 9.3
differential_evolution step 82: f(x)= 9.3
differential_evolution step 83: f(x)= 9.3
differential_evolution step 84: f(x)= 9.3
differential_evolution step 85: f(x)= 9.3
differential_evolution step 86: f(x)= 9.3
differential_evolution step 87: f(x)= 9.3
differential_evolution step 88: f(x)= 9.3
differential_evolution step 89: f(x)= 9.3
differential_evolution step 90: f(x)= 9.3
differential_evolution step 91: f(x)= 9.3
differential_evolution step 92: f(x)= 9.3
differential_evolution step 93: f(x)= 9.3
differential_evolution step 94: f(x)= 9.3
differential_evolution step 95: f(x)= 9.3
differential_evolution step 96: f(x)= 9.1
differential_evolution step 97: f(x)= 9.1
differential_evolution step 98: f(x)= 9.1
differential_evolution step 99: f(x)= 9.1
differential_evolution step 100: f(x)= 9.1
differential_evolution step 101: f(x)= 9.1
differential_evolution step 102: f(x)= 9.1
differential_evolution step 103: f(x)= 9.1
differential_evolution step 104: f(x)= 9.1
differential_evolution step 105: f(x)= 9.1
differential_evolution step 106: f(x)= 9.1
differential_evolution step 107: f(x)= 9.1
differential_evolution step 108: f(x)= 9.1
differential_evolution step 109: f(x)= 9.1
differential_evolution step 110: f(x)= 9.1
differential_evolution step 111: f(x)= 9.1
differential_evolution step 112: f(x)= 9.1
differential_evolution step 113: f(x)= 9.1
differential_evolution step 114: f(x)= 9.1
differential_evolution step 115: f(x)= 9.1
differential_evolution step 116: f(x)= 9.1
differential_evolution step 117: f(x)= 9.1
differential_evolution step 118: f(x)= 9.1
differential_evolution step 119: f(x)= 9.1
differential_evolution step 120: f(x)= 9.1
differential_evolution step 121: f(x)= 9.1
differential_evolution step 122: f(x)= 8.9
differential_evolution step 123: f(x)= 8.9
differential_evolution step 124: f(x)= 8.9
differential_evolution step 125: f(x)= 8.9
differential_evolution step 126: f(x)= 8.9
differential_evolution step 127: f(x)= 8.9
```

```
differential_evolution step 128: f(x)= 8.9
differential_evolution step 129: f(x)= 8.9
differential_evolution step 130: f(x)= 8.9
differential_evolution step 131: f(x)= 8.9
differential_evolution step 132: f(x)= 8.9
differential_evolution step 133: f(x)= 8.9
differential_evolution step 134: f(x)= 8.9
differential_evolution step 135: f(x)= 8.9
differential_evolution step 136: f(x)= 8.9
differential_evolution step 137: f(x)= 8.9
differential_evolution step 138: f(x)= 8.9
differential_evolution step 139: f(x)= 8.9
differential_evolution step 140: f(x)= 8.9
differential_evolution step 141: f(x)= 8.9
differential_evolution step 142: f(x)= 8.9
differential_evolution step 143: f(x)= 8.9
differential_evolution step 144: f(x)= 8.9
differential_evolution step 145: f(x)= 8.9
differential_evolution step 146: f(x)= 8.9
differential_evolution step 147: f(x)= 8.9
differential_evolution step 148: f(x)= 8.9
differential_evolution step 149: f(x)= 8.9
differential_evolution step 150: f(x)= 8.9
differential_evolution step 151: f(x)= 8.9
differential_evolution step 152: f(x)= 8.9
differential_evolution step 153: f(x)= 8.9
differential_evolution step 154: f(x)= 8.9
differential_evolution step 155: f(x)= 8.9
differential_evolution step 156: f(x)= 8.9
differential_evolution step 157: f(x)= 8.9
differential_evolution step 158: f(x)= 8.9
differential_evolution step 159: f(x)= 8.9
differential_evolution step 160: f(x)= 8.9
differential_evolution step 161: f(x)= 8.9
differential_evolution step 162: f(x)= 8.9
differential_evolution step 163: f(x)= 8.9
differential_evolution step 164: f(x)= 8.9
differential_evolution step 165: f(x)= 8.9
differential_evolution step 166: f(x)= 8.9
differential_evolution step 167: f(x)= 8.9
differential_evolution step 168: f(x)= 8.9
differential_evolution step 169: f(x)= 8.9
differential_evolution step 170: f(x)= 8.9
differential_evolution step 171: f(x)= 8.9
differential_evolution step 172: f(x)= 8.9
differential_evolution step 173: f(x)= 8.9
differential_evolution step 174: f(x)= 8.9
differential_evolution step 175: f(x)= 8.9
differential_evolution step 176: f(x)= 8.9
differential_evolution step 177: f(x)= 8.9
differential_evolution step 178: f(x)= 8.9
differential_evolution step 179: f(x)= 8.9
differential_evolution step 180: f(x)= 8.9
differential_evolution step 181: f(x)= 8.9
differential_evolution step 182: f(x)= 8.9
differential_evolution step 183: f(x)= 8.9
differential_evolution step 184: f(x)= 8.9
differential_evolution step 185: f(x)= 8.9
differential_evolution step 186: f(x)= 8.9
differential_evolution step 187: f(x)= 8.9
differential_evolution step 188: f(x)= 8.9
differential_evolution step 189: f(x)= 8.9
differential_evolution step 190: f(x)= 8.9
differential_evolution step 191: f(x)= 8.9
```

```
differential_evolution step 192: f(x)= 8.9
differential_evolution step 193: f(x)= 8.9
differential_evolution step 194: f(x)= 8.9
differential_evolution step 195: f(x)= 8.9
differential_evolution step 196: f(x)= 8.9
differential_evolution step 197: f(x)= 8.9
differential_evolution step 198: f(x)= 8.9
differential_evolution step 199: f(x)= 8.9
differential_evolution step 200: f(x)= 8.9
differential_evolution step 201: f(x)= 8.9
differential_evolution step 202: f(x)= 8.9
differential_evolution step 203: f(x)= 8.9
differential_evolution step 204: f(x)= 8.9
differential_evolution step 205: f(x)= 8.9
differential_evolution step 206: f(x)= 8.9
differential_evolution step 207: f(x)= 8.9
differential_evolution step 208: f(x)= 8.9
differential_evolution step 209: f(x)= 8.9
differential_evolution step 210: f(x)= 8.9
differential_evolution step 211: f(x)= 8.9
differential_evolution step 212: f(x)= 8.9
differential_evolution step 213: f(x)= 8.9
differential_evolution step 214: f(x)= 8.9
differential_evolution step 215: f(x)= 8.9
differential_evolution step 216: f(x)= 8.9
differential_evolution step 217: f(x)= 8.9
differential_evolution step 218: f(x)= 8.9
differential_evolution step 219: f(x)= 8.9
differential_evolution step 220: f(x)= 8.9
differential_evolution step 221: f(x)= 8.9
differential_evolution step 222: f(x)= 8.9
differential_evolution step 223: f(x)= 8.9
differential_evolution step 224: f(x)= 8.9
differential_evolution step 225: f(x)= 8.9
differential_evolution step 226: f(x)= 8.9
differential_evolution step 227: f(x)= 8.9
differential_evolution step 228: f(x)= 8.9
differential_evolution step 229: f(x)= 8.9
differential_evolution step 230: f(x)= 8.9
differential_evolution step 231: f(x)= 8.9
differential_evolution step 232: f(x)= 8.9
differential_evolution step 233: f(x)= 8.9
differential_evolution step 234: f(x)= 8.9
differential_evolution step 235: f(x)= 8.9
differential_evolution step 236: f(x)= 8.9
differential_evolution step 237: f(x)= 8.9
differential_evolution step 238: f(x)= 8.9
differential_evolution step 239: f(x)= 8.9
differential_evolution step 240: f(x)= 8.9
differential_evolution step 241: f(x)= 8.9
differential_evolution step 242: f(x)= 8.9
differential_evolution step 243: f(x)= 8.9
differential_evolution step 244: f(x)= 8.9
differential_evolution step 245: f(x)= 8.9
differential_evolution step 246: f(x)= 8.9
differential_evolution step 247: f(x)= 8.9
differential_evolution step 248: f(x)= 8.9
differential_evolution step 249: f(x)= 8.9
differential_evolution step 250: f(x)= 8.9
differential_evolution step 251: f(x)= 8.9
differential_evolution step 252: f(x)= 8.9
differential_evolution step 253: f(x)= 8.9
differential_evolution step 254: f(x)= 8.9
differential_evolution step 255: f(x)= 8.9
```

```
differential_evolution step 256: f(x)= 8.9
differential_evolution step 257: f(x)= 8.9
differential_evolution step 258: f(x)= 8.9
differential_evolution step 259: f(x)= 8.9
differential_evolution step 260: f(x)= 8.9
differential_evolution step 261: f(x)= 8.9
differential_evolution step 262: f(x)= 8.9
differential_evolution step 263: f(x)= 8.9
differential_evolution step 264: f(x)= 8.9
differential_evolution step 265: f(x)= 8.9
differential_evolution step 266: f(x)= 8.9
differential_evolution step 267: f(x)= 8.9
differential_evolution step 268: f(x)= 8.9
differential_evolution step 269: f(x)= 8.9
differential_evolution step 270: f(x)= 8.9
differential_evolution step 271: f(x)= 8.9
differential_evolution step 272: f(x)= 8.9
differential_evolution step 273: f(x)= 8.9
differential_evolution step 274: f(x)= 8.9
differential_evolution step 275: f(x)= 8.9
differential_evolution step 276: f(x)= 8.9
differential_evolution step 277: f(x)= 8.9
differential_evolution step 278: f(x)= 8.9
differential_evolution step 279: f(x)= 8.9
differential_evolution step 280: f(x)= 8.4
differential_evolution step 281: f(x)= 8.4
differential_evolution step 282: f(x)= 8.4
differential_evolution step 283: f(x)= 8.4
differential_evolution step 284: f(x)= 8.4
differential_evolution step 285: f(x)= 8.4
differential_evolution step 286: f(x)= 8.4
differential_evolution step 287: f(x)= 8.4
differential_evolution step 288: f(x)= 8.4
differential_evolution step 289: f(x)= 8.4
differential_evolution step 290: f(x)= 8.4
differential_evolution step 291: f(x)= 8.4
differential_evolution step 292: f(x)= 8.4
differential_evolution step 293: f(x)= 8.4
differential_evolution step 294: f(x)= 8.4
differential_evolution step 295: f(x)= 8.4
differential_evolution step 296: f(x)= 8.4
differential_evolution step 297: f(x)= 8.4
differential_evolution step 298: f(x)= 8.4
differential_evolution step 299: f(x)= 8.4
differential_evolution step 300: f(x)= 8.4
differential_evolution step 301: f(x)= 8.4
differential_evolution step 302: f(x)= 8.4
differential_evolution step 303: f(x)= 8.4
differential_evolution step 304: f(x)= 8.4
differential_evolution step 305: f(x)= 8.4
differential_evolution step 306: f(x)= 8.4
differential_evolution step 307: f(x)= 8.4
differential_evolution step 308: f(x)= 8.4
differential_evolution step 309: f(x)= 8.4
differential_evolution step 310: f(x)= 8.4
differential_evolution step 311: f(x)= 8.4
differential_evolution step 312: f(x)= 8.4
differential_evolution step 313: f(x)= 8.4
differential_evolution step 314: f(x)= 8.4
differential_evolution step 315: f(x)= 8.4
differential_evolution step 316: f(x)= 8.4
differential_evolution step 317: f(x)= 8.4
differential_evolution step 318: f(x)= 8.4
differential_evolution step 319: f(x)= 8.4
```

```
differential_evolution step 320: f(x)= 8.4
differential_evolution step 321: f(x)= 8.4
differential_evolution step 322: f(x)= 8.4
differential_evolution step 323: f(x)= 8.4
differential_evolution step 324: f(x)= 8.4
differential_evolution step 325: f(x)= 8.4
differential_evolution step 326: f(x)= 8.4
differential_evolution step 327: f(x)= 8.4
differential_evolution step 328: f(x)= 8.200000000000001
differential_evolution step 329: f(x)= 8.200000000000001
differential_evolution step 330: f(x)= 8.200000000000001
differential_evolution step 331: f(x)= 8.200000000000001
differential_evolution step 332: f(x)= 8.200000000000001
differential_evolution step 333: f(x)= 8.200000000000001
differential_evolution step 334: f(x)= 8.200000000000001
differential_evolution step 335: f(x)= 8.200000000000001
differential_evolution step 336: f(x)= 8.200000000000001
differential_evolution step 337: f(x)= 8.200000000000001
differential_evolution step 338: f(x)= 8.200000000000001
differential_evolution step 339: f(x)= 8.200000000000001
differential_evolution step 340: f(x)= 8.200000000000001
differential_evolution step 341: f(x)= 8.200000000000001
differential_evolution step 342: f(x)= 8.200000000000001
differential_evolution step 343: f(x)= 8.200000000000001
differential_evolution step 344: f(x)= 8.200000000000001
differential_evolution step 345: f(x)= 8.200000000000001
differential_evolution step 346: f(x)= 8.200000000000001
differential_evolution step 347: f(x)= 8.200000000000001
differential_evolution step 348: f(x)= 8.200000000000001
differential_evolution step 349: f(x)= 8.200000000000001
differential_evolution step 350: f(x)= 8.200000000000001
differential_evolution step 351: f(x)= 8.200000000000001
differential_evolution step 352: f(x)= 8.200000000000001
differential_evolution step 353: f(x)= 8.200000000000001
differential_evolution step 354: f(x)= 8.200000000000001
differential_evolution step 355: f(x)= 8.200000000000001
differential_evolution step 356: f(x)= 8.200000000000001
differential_evolution step 357: f(x)= 8.200000000000001
differential_evolution step 358: f(x)= 8.200000000000001
differential_evolution step 359: f(x)= 8.200000000000001
differential_evolution step 360: f(x)= 8.200000000000001
differential_evolution step 361: f(x)= 8.200000000000001
differential_evolution step 362: f(x)= 8.200000000000001
differential_evolution step 363: f(x)= 8.200000000000001
differential_evolution step 364: f(x)= 8.200000000000001
differential_evolution step 365: f(x)= 8.200000000000001
differential_evolution step 366: f(x)= 8.200000000000001
differential_evolution step 367: f(x)= 8.200000000000001
differential_evolution step 368: f(x)= 8.200000000000001
differential_evolution step 369: f(x)= 8.200000000000001
differential_evolution step 370: f(x)= 8.200000000000001
differential_evolution step 371: f(x)= 8.200000000000001
differential_evolution step 372: f(x)= 8.200000000000001
differential_evolution step 373: f(x)= 8.200000000000001
differential_evolution step 374: f(x)= 8.200000000000001
differential_evolution step 375: f(x)= 8.200000000000001
differential_evolution step 376: f(x)= 8.200000000000001
differential_evolution step 377: f(x)= 8.200000000000001
differential_evolution step 378: f(x)= 8.200000000000001
differential_evolution step 379: f(x)= 8.200000000000001
differential_evolution step 380: f(x)= 8.200000000000001
differential_evolution step 381: f(x)= 8.200000000000001
differential_evolution step 382: f(x)= 8.200000000000001
differential_evolution step 383: f(x)= 8.200000000000001
```

```
differential_evolution step 384: f(x)= 8.200000000000001
differential_evolution step 385: f(x)= 8.200000000000001
differential_evolution step 386: f(x)= 8.200000000000001
differential_evolution step 387: f(x)= 8.200000000000001
differential_evolution step 388: f(x)= 8.200000000000001
differential_evolution step 389: f(x)= 8.200000000000001
differential_evolution step 390: f(x)= 8.200000000000001
differential_evolution step 391: f(x)= 8.200000000000001
differential_evolution step 392: f(x)= 8.200000000000001
differential_evolution step 393: f(x)= 8.200000000000001
differential_evolution step 394: f(x)= 8.200000000000001
differential_evolution step 395: f(x)= 8.200000000000001
differential_evolution step 396: f(x)= 8.200000000000001
differential_evolution step 397: f(x)= 8.200000000000001
differential_evolution step 398: f(x)= 8.200000000000001
differential_evolution step 399: f(x)= 8.200000000000001
differential_evolution step 400: f(x)= 8.200000000000001
differential_evolution step 401: f(x)= 8.200000000000001
differential_evolution step 402: f(x)= 8.200000000000001
differential_evolution step 403: f(x)= 8.200000000000001
differential_evolution step 404: f(x)= 8.200000000000001
differential_evolution step 405: f(x)= 8.200000000000001
differential_evolution step 406: f(x)= 8.200000000000001
differential_evolution step 407: f(x)= 8.200000000000001
differential_evolution step 408: f(x)= 8.200000000000001
differential_evolution step 409: f(x)= 8.200000000000001
differential_evolution step 410: f(x)= 8.200000000000001
differential_evolution step 411: f(x)= 8.200000000000001
differential_evolution step 412: f(x)= 8.200000000000001
differential_evolution step 413: f(x)= 8.200000000000001
differential_evolution step 414: f(x)= 8.200000000000001
differential_evolution step 415: f(x)= 8.200000000000001
differential_evolution step 416: f(x)= 8.200000000000001
differential_evolution step 417: f(x)= 8.200000000000001
differential_evolution step 418: f(x)= 8.200000000000001
differential_evolution step 419: f(x)= 8.200000000000001
differential_evolution step 420: f(x)= 8.200000000000001
differential_evolution step 421: f(x)= 8.200000000000001
differential_evolution step 422: f(x)= 8.200000000000001
differential_evolution step 423: f(x)= 8.200000000000001
differential_evolution step 424: f(x)= 8.200000000000001
differential_evolution step 425: f(x)= 8.200000000000001
differential_evolution step 426: f(x)= 8.200000000000001
differential_evolution step 427: f(x)= 8.200000000000001
differential_evolution step 428: f(x)= 8.200000000000001
differential_evolution step 429: f(x)= 8.200000000000001
differential_evolution step 430: f(x)= 8.200000000000001
differential_evolution step 431: f(x)= 8.200000000000001
differential_evolution step 432: f(x)= 8.200000000000001
differential_evolution step 433: f(x)= 8.200000000000001
differential_evolution step 434: f(x)= 8.200000000000001
differential_evolution step 435: f(x)= 8.200000000000001
differential_evolution step 436: f(x)= 8.200000000000001
differential_evolution step 437: f(x)= 8.200000000000001
differential_evolution step 438: f(x)= 8.200000000000001
differential_evolution step 439: f(x)= 8.200000000000001
differential_evolution step 440: f(x)= 8.200000000000001
differential_evolution step 441: f(x)= 8.200000000000001
differential_evolution step 442: f(x)= 8.200000000000001
differential_evolution step 443: f(x)= 8.200000000000001
differential_evolution step 444: f(x)= 8.200000000000001
differential_evolution step 445: f(x)= 8.200000000000001
differential_evolution step 446: f(x)= 8.200000000000001
differential_evolution step 447: f(x)= 8.200000000000001
```

```
differential_evolution step 448: f(x)= 8.200000000000001
differential_evolution step 449: f(x)= 8.200000000000001
differential_evolution step 450: f(x)= 8.200000000000001
differential_evolution step 451: f(x)= 8.200000000000001
differential_evolution step 452: f(x)= 8.200000000000001
differential_evolution step 453: f(x)= 8.200000000000001
differential_evolution step 454: f(x)= 8.200000000000001
differential_evolution step 455: f(x)= 8.200000000000001
differential_evolution step 456: f(x)= 8.200000000000001
differential_evolution step 457: f(x)= 8.200000000000001
differential_evolution step 458: f(x)= 8.200000000000001
differential_evolution step 459: f(x)= 8.200000000000001
differential_evolution step 460: f(x)= 8.200000000000001
differential_evolution step 461: f(x)= 8.200000000000001
differential_evolution step 462: f(x)= 8.200000000000001
differential_evolution step 463: f(x)= 8.200000000000001
differential_evolution step 464: f(x)= 8.200000000000001
differential_evolution step 465: f(x)= 8.200000000000001
differential_evolution step 466: f(x)= 8.200000000000001
differential_evolution step 467: f(x)= 8.200000000000001
differential_evolution step 468: f(x)= 8.200000000000001
differential_evolution step 469: f(x)= 8.200000000000001
differential_evolution step 470: f(x)= 8.200000000000001
differential_evolution step 471: f(x)= 8.200000000000001
differential_evolution step 472: f(x)= 8.200000000000001
differential_evolution step 473: f(x)= 8.200000000000001
differential_evolution step 474: f(x)= 8.200000000000001
differential_evolution step 475: f(x)= 8.200000000000001
differential_evolution step 476: f(x)= 8.200000000000001
differential_evolution step 477: f(x)= 8.200000000000001
differential_evolution step 478: f(x)= 8.200000000000001
differential_evolution step 479: f(x)= 8.200000000000001
differential_evolution step 480: f(x)= 8.200000000000001
differential_evolution step 481: f(x)= 8.200000000000001
differential_evolution step 482: f(x)= 8.200000000000001
differential_evolution step 483: f(x)= 8.200000000000001
differential_evolution step 484: f(x)= 7.500000000000001
differential_evolution step 485: f(x)= 7.500000000000001
differential_evolution step 486: f(x)= 7.500000000000001
differential_evolution step 487: f(x)= 7.500000000000001
differential_evolution step 488: f(x)= 7.500000000000001
differential_evolution step 489: f(x)= 7.500000000000001
differential_evolution step 490: f(x)= 7.500000000000001
differential_evolution step 491: f(x)= 7.500000000000001
differential_evolution step 492: f(x)= 7.500000000000001
differential_evolution step 493: f(x)= 7.500000000000001
differential_evolution step 494: f(x)= 7.500000000000001
differential_evolution step 495: f(x)= 7.500000000000001
differential_evolution step 496: f(x)= 7.500000000000001
differential_evolution step 497: f(x)= 7.500000000000001
differential_evolution step 498: f(x)= 7.500000000000001
differential_evolution step 499: f(x)= 7.500000000000001
differential_evolution step 500: f(x)= 7.500000000000001
differential_evolution step 501: f(x)= 7.500000000000001
differential_evolution step 502: f(x)= 7.500000000000001
differential_evolution step 503: f(x)= 7.500000000000001
differential_evolution step 504: f(x)= 7.500000000000001
differential_evolution step 505: f(x)= 7.500000000000001
differential_evolution step 506: f(x)= 7.500000000000001
differential_evolution step 507: f(x)= 7.500000000000001
differential_evolution step 508: f(x)= 7.500000000000001
differential_evolution step 509: f(x)= 7.500000000000001
differential_evolution step 510: f(x)= 7.500000000000001
differential_evolution step 511: f(x)= 7.500000000000001
```

```
differential_evolution step 512: f(x)= 7.500000000000001
differential_evolution step 513: f(x)= 7.500000000000001
differential_evolution step 514: f(x)= 7.500000000000001
differential_evolution step 515: f(x)= 7.500000000000001
differential_evolution step 516: f(x)= 7.500000000000001
differential_evolution step 517: f(x)= 7.500000000000001
differential_evolution step 518: f(x)= 7.500000000000001
differential_evolution step 519: f(x)= 7.500000000000001
differential_evolution step 520: f(x)= 7.500000000000001
differential_evolution step 521: f(x)= 7.500000000000001
differential_evolution step 522: f(x)= 7.500000000000001
differential_evolution step 523: f(x)= 7.500000000000001
differential_evolution step 524: f(x)= 7.500000000000001
differential_evolution step 525: f(x)= 7.500000000000001
differential_evolution step 526: f(x)= 7.500000000000001
differential_evolution step 527: f(x)= 7.500000000000001
differential_evolution step 528: f(x)= 7.500000000000001
differential_evolution step 529: f(x)= 7.500000000000001
differential_evolution step 530: f(x)= 7.500000000000001
differential_evolution step 531: f(x)= 7.500000000000001
differential_evolution step 532: f(x)= 7.500000000000001
differential_evolution step 533: f(x)= 7.500000000000001
differential_evolution step 534: f(x)= 7.500000000000001
differential_evolution step 535: f(x)= 7.500000000000001
differential_evolution step 536: f(x)= 7.500000000000001
differential_evolution step 537: f(x)= 7.500000000000001
differential_evolution step 538: f(x)= 7.500000000000001
differential_evolution step 539: f(x)= 7.500000000000001
differential_evolution step 540: f(x)= 7.500000000000001
differential_evolution step 541: f(x)= 7.500000000000001
differential_evolution step 542: f(x)= 7.500000000000001
differential_evolution step 543: f(x)= 7.500000000000001
differential_evolution step 544: f(x)= 7.500000000000001
differential_evolution step 545: f(x)= 7.500000000000001
differential_evolution step 546: f(x)= 7.500000000000001
differential_evolution step 547: f(x)= 7.500000000000001
differential_evolution step 548: f(x)= 7.500000000000001
differential_evolution step 549: f(x)= 7.500000000000001
differential_evolution step 550: f(x)= 7.500000000000001
differential_evolution step 551: f(x)= 7.500000000000001
differential_evolution step 552: f(x)= 7.500000000000001
differential_evolution step 553: f(x)= 7.500000000000001
differential_evolution step 554: f(x)= 7.500000000000001
differential_evolution step 555: f(x)= 7.500000000000001
differential_evolution step 556: f(x)= 7.500000000000001
differential_evolution step 557: f(x)= 7.500000000000001
differential_evolution step 558: f(x)= 7.500000000000001
differential_evolution step 559: f(x)= 7.500000000000001
differential_evolution step 560: f(x)= 7.500000000000001
differential_evolution step 561: f(x)= 7.500000000000001
differential_evolution step 562: f(x)= 7.500000000000001
differential_evolution step 563: f(x)= 7.500000000000001
differential_evolution step 564: f(x)= 7.500000000000001
differential_evolution step 565: f(x)= 7.500000000000001
differential_evolution step 566: f(x)= 7.500000000000001
differential_evolution step 567: f(x)= 7.500000000000001
differential_evolution step 568: f(x)= 7.500000000000001
differential_evolution step 569: f(x)= 7.500000000000001
differential_evolution step 570: f(x)= 7.500000000000001
differential_evolution step 571: f(x)= 7.500000000000001
differential_evolution step 572: f(x)= 7.500000000000001
differential_evolution step 573: f(x)= 7.500000000000001
differential_evolution step 574: f(x)= 7.500000000000001
differential_evolution step 575: f(x)= 7.500000000000001
```

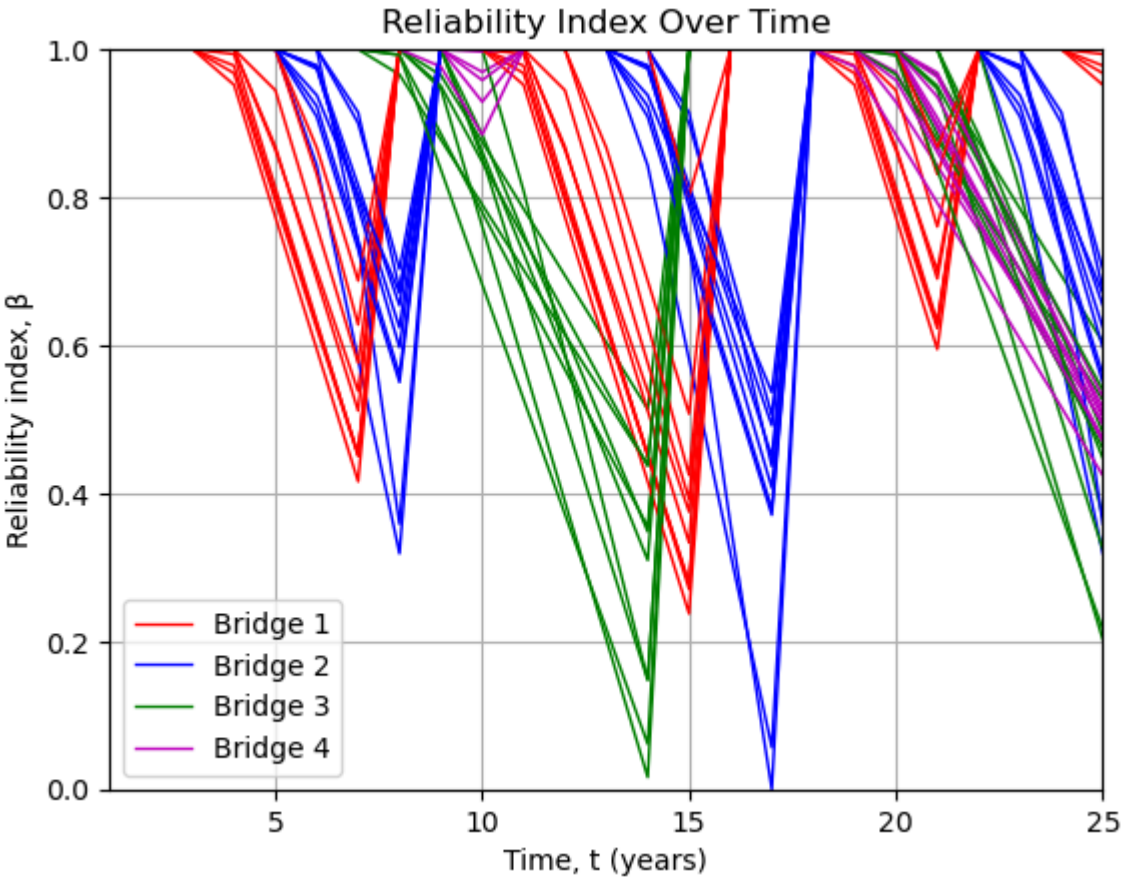
```
differential_evolution step 576: f(x)= 7.500000000000001
differential_evolution step 577: f(x)= 7.500000000000001
differential_evolution step 578: f(x)= 7.500000000000001
differential_evolution step 579: f(x)= 7.500000000000001
differential_evolution step 580: f(x)= 7.500000000000001
differential_evolution step 581: f(x)= 7.500000000000001
differential_evolution step 582: f(x)= 7.500000000000001
differential_evolution step 583: f(x)= 7.500000000000001
differential_evolution step 584: f(x)= 7.500000000000001
differential_evolution step 585: f(x)= 7.500000000000001
differential_evolution step 586: f(x)= 7.500000000000001
differential_evolution step 587: f(x)= 7.500000000000001
differential_evolution step 588: f(x)= 7.500000000000001
differential_evolution step 589: f(x)= 7.500000000000001
differential_evolution step 590: f(x)= 7.500000000000001
differential_evolution step 591: f(x)= 6.9
differential_evolution step 592: f(x)= 6.9
differential_evolution step 593: f(x)= 6.9
differential_evolution step 594: f(x)= 6.9
differential_evolution step 595: f(x)= 6.9
differential_evolution step 596: f(x)= 6.9
differential_evolution step 597: f(x)= 6.9
differential_evolution step 598: f(x)= 6.9
differential_evolution step 599: f(x)= 6.9
differential_evolution step 600: f(x)= 6.9
differential_evolution step 601: f(x)= 6.9
differential_evolution step 602: f(x)= 6.9
differential_evolution step 603: f(x)= 6.9
differential_evolution step 604: f(x)= 6.9
differential_evolution step 605: f(x)= 6.9
differential_evolution step 606: f(x)= 6.9
differential_evolution step 607: f(x)= 6.9
differential_evolution step 608: f(x)= 6.9
differential_evolution step 609: f(x)= 6.9
differential_evolution step 610: f(x)= 6.9
differential_evolution step 611: f(x)= 6.9
differential_evolution step 612: f(x)= 6.9
differential_evolution step 613: f(x)= 6.9
differential_evolution step 614: f(x)= 6.9
differential_evolution step 615: f(x)= 6.9
differential_evolution step 616: f(x)= 6.9
differential_evolution step 617: f(x)= 6.9
differential_evolution step 618: f(x)= 6.9
differential_evolution step 619: f(x)= 6.9
differential_evolution step 620: f(x)= 6.9
differential_evolution step 621: f(x)= 6.9
differential_evolution step 622: f(x)= 6.9
differential_evolution step 623: f(x)= 6.9
differential_evolution step 624: f(x)= 6.9
differential_evolution step 625: f(x)= 6.9
differential_evolution step 626: f(x)= 6.9
differential_evolution step 627: f(x)= 6.9
differential_evolution step 628: f(x)= 6.9
differential_evolution step 629: f(x)= 6.9
differential_evolution step 630: f(x)= 6.9
differential_evolution step 631: f(x)= 6.9
differential_evolution step 632: f(x)= 6.9
differential_evolution step 633: f(x)= 6.9
differential_evolution step 634: f(x)= 6.9
differential_evolution step 635: f(x)= 6.9
differential_evolution step 636: f(x)= 6.9
differential_evolution step 637: f(x)= 6.9
differential_evolution step 638: f(x)= 6.9
differential_evolution step 639: f(x)= 6.9
```

```
differential_evolution step 640: f(x)= 6.9
differential_evolution step 641: f(x)= 6.9
differential_evolution step 642: f(x)= 6.9
differential_evolution step 643: f(x)= 6.9
differential_evolution step 644: f(x)= 6.9
differential_evolution step 645: f(x)= 6.9
differential_evolution step 646: f(x)= 6.9
differential_evolution step 647: f(x)= 6.9
differential_evolution step 648: f(x)= 6.9
differential_evolution step 649: f(x)= 6.9
differential_evolution step 650: f(x)= 6.9
differential_evolution step 651: f(x)= 6.9
differential_evolution step 652: f(x)= 6.9
differential_evolution step 653: f(x)= 6.9
differential_evolution step 654: f(x)= 6.9
differential_evolution step 655: f(x)= 6.9
differential_evolution step 656: f(x)= 6.9
differential_evolution step 657: f(x)= 6.9
differential_evolution step 658: f(x)= 6.9
differential_evolution step 659: f(x)= 6.9
differential_evolution step 660: f(x)= 6.9
differential_evolution step 661: f(x)= 6.9
differential_evolution step 662: f(x)= 6.9
differential_evolution step 663: f(x)= 6.9
differential_evolution step 664: f(x)= 6.9
differential_evolution step 665: f(x)= 6.9
differential_evolution step 666: f(x)= 6.9
differential_evolution step 667: f(x)= 6.9
differential_evolution step 668: f(x)= 6.9
differential_evolution step 669: f(x)= 6.9
differential_evolution step 670: f(x)= 6.9
differential_evolution step 671: f(x)= 6.9
differential_evolution step 672: f(x)= 6.9
differential_evolution step 673: f(x)= 6.9
differential_evolution step 674: f(x)= 6.9
differential_evolution step 675: f(x)= 6.9
differential_evolution step 676: f(x)= 6.9
differential_evolution step 677: f(x)= 6.9
differential_evolution step 678: f(x)= 6.1
differential_evolution step 679: f(x)= 6.1
differential_evolution step 680: f(x)= 6.1
differential_evolution step 681: f(x)= 6.1
differential_evolution step 682: f(x)= 6.1
differential_evolution step 683: f(x)= 6.1
differential_evolution step 684: f(x)= 6.1
differential_evolution step 685: f(x)= 6.1
differential_evolution step 686: f(x)= 6.1
differential_evolution step 687: f(x)= 6.1
differential_evolution step 688: f(x)= 6.1
differential_evolution step 689: f(x)= 6.1
differential_evolution step 690: f(x)= 6.1
differential_evolution step 691: f(x)= 6.1
differential_evolution step 692: f(x)= 6.1
differential_evolution step 693: f(x)= 6.1
differential_evolution step 694: f(x)= 6.1
differential_evolution step 695: f(x)= 6.1
differential_evolution step 696: f(x)= 6.1
differential_evolution step 697: f(x)= 6.1
differential_evolution step 698: f(x)= 6.1
differential_evolution step 699: f(x)= 6.1
differential_evolution step 700: f(x)= 6.1
differential_evolution step 701: f(x)= 6.1
differential_evolution step 702: f(x)= 6.1
differential_evolution step 703: f(x)= 6.1
```

```
differential_evolution step 704: f(x)= 6.1
differential_evolution step 705: f(x)= 6.1
differential_evolution step 706: f(x)= 6.1
differential_evolution step 707: f(x)= 6.1
differential_evolution step 708: f(x)= 6.1
differential_evolution step 709: f(x)= 6.1
differential_evolution step 710: f(x)= 6.1
differential_evolution step 711: f(x)= 6.1
differential_evolution step 712: f(x)= 6.1
differential_evolution step 713: f(x)= 6.1
differential_evolution step 714: f(x)= 6.1
differential_evolution step 715: f(x)= 6.1
differential_evolution step 716: f(x)= 6.1
differential_evolution step 717: f(x)= 6.1
differential_evolution step 718: f(x)= 6.1
differential_evolution step 719: f(x)= 6.1
differential_evolution step 720: f(x)= 6.1
differential_evolution step 721: f(x)= 6.1
differential_evolution step 722: f(x)= 6.1
differential_evolution step 723: f(x)= 6.1
differential_evolution step 724: f(x)= 6.1
differential_evolution step 725: f(x)= 6.1
differential_evolution step 726: f(x)= 6.1
differential_evolution step 727: f(x)= 6.1
differential_evolution step 728: f(x)= 6.1
differential_evolution step 729: f(x)= 6.1
differential_evolution step 730: f(x)= 6.1
differential_evolution step 731: f(x)= 6.1
differential_evolution step 732: f(x)= 6.1
differential_evolution step 733: f(x)= 6.1
differential_evolution step 734: f(x)= 6.1
differential_evolution step 735: f(x)= 6.1
differential_evolution step 736: f(x)= 6.1
differential_evolution step 737: f(x)= 6.1
differential_evolution step 738: f(x)= 6.1
differential_evolution step 739: f(x)= 6.1
differential_evolution step 740: f(x)= 6.1
[ 7.  0.  2.  0.  8.  8.  0.  0.  6.  9. 12. 10.  0.  0.  0.  0.]
```

```python
Yopt = result.x
OptInterv = cumulateFreq(Yopt.reshape((nt, nb)).T)
X_bt = Frequency2Schedule(Yopt, nb, nt, Thoriz)

print(f'Min Cost = {result.fun:.2f}')
ninterv = np.sum(CountNinterv(Yopt, nt, nb))
print(f'Number of interventions = {ninterv}')
c = nonlcon(Yopt)
BudgetIssues = np.sum(c[-Thoriz - 1:-1] > 0)
print(f'Number of time intervals with exceeding budget = {BudgetIssues}')
for b in range(1, nb + 1):
    probfulfilment = 0.90 - c[-nb + b - 1]
    print(f'Prob(reliability>0) for bridge {b} = {probfulfilment:.4f}')

plt.figure()
h = []
leg = []
for i in range(nMCS):
    beta_bt = MaintenanceApplication(beta_bti0[:, :, i], X_bt, Thoriz)  # optimal mainten
    for b in range(nb):
        h.append(plt.plot(range(1, Thoriz + 1), beta_bt[b, :], f'-{col[b]}', linewidth=1)
        if i == 0:
            leg.append(f'Bridge {b + 1}')
plt.legend(leg, loc='best')
plt.grid()
plt.xlim([1, Thoriz])
plt.ylim([0, 1])
plt.xlabel('Time, t (years)')
plt.ylabel('Reliability index, β')
plt.title('Reliability Index Over Time')
plt.show()
```

```
Min Cost = 6.10
Number of interventions = 8
Number of time intervals with exceeding budget = 0
Prob(reliability>0) for bridge 1 = 1.0000
Prob(reliability>0) for bridge 2 = 0.9000
Prob(reliability>0) for bridge 3 = 1.0000
Prob(reliability>0) for bridge 4 = 1.0000
```


Reliability Index Over Time
Reliability index, β
Time, t (years)
1.0
0.8
0.6
0.4
0.2
0.0
5
10
15
20
25
Bridge 1
Bridge 2
Bridge 3
Bridge 4

Bibliography

[Car23] Carpentries. Workshop template. 2023. URL: https://carpentries.github.io/workshop-template/#python.

[MUD23] MUDE. Getting started: week 1 software installation. 2023. URL: https://mude.citg.tudelft.nl/software/getting_started/.

[SP97] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997. doi:10.1023/A:1008202821328.

[BlankDeb20] J. Blank and K. Deb. Pymoo: multi-objective optimization in python. *IEEE Access*, 8():89497–89509, 2020.

[TheScommunity24] The SciPy community. Scipy user guide. 2024. URL: https://docs.scipy.org/doc/scipy/tutorial/optimize.html.

Credits and License

Contents

You can refer to this book as:

> Tom van Woudenberg and Maria Nogal from Delft University of Technology (2025) *Engineering Systems Optimization*. GitHub/Zenodo. 10.5281/zenodo.15099966

You can refer to individual chapters or pages within this book as:

> `<Title of Chapter or Page>`. In Tom van Woudenberg and Maria Nogal from Delft University of Technology (2025) *Engineering Systems Optimization*. GitHub/Zenodo. 10.5281/zenodo.15099966. Accessed `date`.

We anticipate that the content of this book will change significantly. Therefore, we recommend using the source code directly with the citation above that refers to the GitHub repository and lists the date and name of the file. Although content will be added over time, chapter titles and URL's in this book are expected to remain relatively static. However, we make no guarantee, so if it is important for you to reference a specific location/commit within the book.

How the book is made

This book is created using open source tools: it is a JupyterBook that is written using Markdown and Jupyter notebooks. Additional tooling is used from the TeachBooks initiative to enhance the editing and reading experience. The files are stored on a public GitHub repository. The website can be viewed at https://oit.tudelft.nl/CME4501. View the repository README file or contact the authors for additional information.

Changelog

Contents

This changelog will include all changes, except for minor adjustments like typos.

v2024.3.0 After class February 26th

- Added solutions to text and downloads Coding of optimization problems - Multi-objective optimization using `scipy` - Class exercise: Ice cream cone
- See full changelog here

v2024.2.0 After class February 26th

- Added solutions to text and downloads Coding of optimization problems - Nonlinear constrained optimization using `scipy` - Class exercise: Suspended beams
- See full changelog here

v2024.1.0 After class February 19th

- Added solutions to text and downloads Coding of optimization problems - Linear constrained optimization using `scipy` - Class exercise: Book distribution
- See full changelog here

v2024.0.0 Start course

- Restructured book with part on introduction, coding of optimization problems and miscellaneous
- Added class exercises to book
- Various improvements of student-experience
- See full changelog here