Parameter and state estimation of a groundwater model using the Ensemble Kalman Filter

by

Moos Castelijn

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Wednesday June 1, 2022 at 4:00 PM.

Student number:	4734459	
Project duration:	February 28, 2022 – June 1, 2022	
Thesis committee:	Prof. dr. ir. M. Verlaan,	TU Delft, supervisor
	Prof. dr. ir. C. Vuik,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Groundwater is an essential ingredient in farming, knowledge about how this is expected to change over time can help farmers improve yields and save water. Predictions about the groundwater level can be made using a mathematical model. This model takes into account the precipitation and evaporation, the flux towards a deeper aquifer and flux to lateral open structures, with parameters for the different resistances and the storage. This model needs input data which is acquired from among others the KNMI (Koningklijk Nederlands Meteorologisch Instituut), with data about the groundwater level at the points of interest acquired from the measurements from the Water board Drenths Overijsselse Delta. The input from the model can be used to generate predictions, whilst the measurement data can be compare with the predictions to obtain more accurate predictions and calibrate the parameters in the model. Combining these factors can be done with the Kalman Filter, a mathematical tool from Data Assimilation which can be used to combine a Mathematical Model and Data into an optimal estimate. Furthermore, it is the wish to improve the parameters in the mathematical model, this causes the state to expand thereby enlarging the problem, and the parameters included in the state causes the problem to become non-linear. The Kalman Filter is only suitable for small linear models but an extension of the regular Kalman Filter can account for these problems, the Ensemble Kalman Filter. To apply all this information for groundwater modeling in a single location, first the numerical errors associated with the different model solutions are tested. It was found that a semi-analytical solution which assumed the input to stay constant over a day to produce the best result. After this the Kalman Filter is compared with the Ensemble Kalman Filter to see whether the Ensemble Kalman Filter can perform comparably in state estimation, it was found that en ensemble size of $n = 200$ this was the case. To test if the parameter calibration scheme works as desired and to test the quality of the parameters found, artificial data is created using the model with all parameters set to one. Then, the parameters are shifted to a different value and the parameter Ensemble Kalman Filter is used to try to find the original parameters back. It is found that in almost all cases the parameter calibration cause the parameter to change in the direction of one, within the first 2/3 months most change within the parameter takes place and the standard deviation of the parameters decreases. After this initial period the parameter remains more stable and the standard deviation changes little. Since the model uses multiple parameters for the resistances, the possibility of calibrating these separately was investigated. It was discovered that in the data usually one resistance parameter is lot smaller than the other, it was enough to calibrate only this parameter to account for the resistance. After this, the parameters in the model are calibrated with actual data. The model always becomes a lot closer to the measurements, and interestingly all parameter become larger than 1. Finally the quality of the predictions is tested. Here we saw that the quality improved fast, already 1/3th year was enough to find parameter far better than originally.

Acknowledgements

I would like to thank Martin Verlaan for pointing me in the right direction when necessary, for reading drafts and for providing commentary. Also, I would like to thank my colleagues Stefan Korenberg from the company Acacia water. He was there for any technical question I had and was always available for some mathematical sparring.
The subject for this thesis was chosen because of the authors work at the company Acacia Water. Here, I came into contact with their groundwater model and the Kalman Filter was a good option to improve their predictions. Because of this, I worked 2 days a week in their company on this Thesis, asking clarification about their data or model when necessary. A branch of this company called Fixeau provided provided the measurement data, this was obtained using their Aqua Pin. Furthermore, while the input data for the model was obtained from the KNMI, Acacia had an easy way to import this data.

Contents

1 Introduction

In this chapter the motivation for using the Ensemble Kalman Filter in the context of groundwater modelling to estimate the state and calibrate the parameters is given, along with a small literature study. The goal of this thesis is defined and the structure of this thesis is outlines.

1.1. Motivation

The number of people in the world is still increasing rapidly, so much so that it is projected that in 2050 the global population will have reached 9.7 billion [27]. This fact, coupled with the development of developing nations which will change their consumption patterns into more consumption of resource-intensive foods, most noteworthy animal-related products, will place increasingly higher pressure on our global food system and by extension, the global farming system. Estimates show that in 2050, agricultural production will have to increase by 60 % to meet the increased demand [1]. Whilst it should be said that this figure is highly contested since many things can happen before 2050 -less food waste and loss, consumption patterns can change etc. - it is clear that the agricultural sector has a large task ahead of itself.

Providing further difficulties for the agricultural sector is the problem of climate change, which will result in more extreme and more fluctuating weather patterns [2], which in turn will result in less reliable and more random water availability. To grow plants, plants need to absorb water through their roots from the ground from the surrounding soil moisture and groundwater. Since this groundwater is an essential ingredient in farming, it is very useful to know the amount of water in the ground at any particular time and to have some insight into how the groundwater level is expected to change over time given the weather prediction. With this, the farmer could choose to either irrigate more or less, which then results in more healthy plants but also in the conservation of water. This is the goal of the company Acacia Water, to this end they developed their Aqua Pins.

These Pins will collect measurements of the groundwater state with intervals of 1 day. The goal of these measurements is to provide farmers with real time information about their groundwater state, and to make predictions about the future to see how the groundwater level will change given the current weather prediction. For the prediction, the measurements are used to obtain a correct mathematical model for the groundwater evolution in this particular area.

A mathematical model is an approximation of reality which takes the current state, other known factors of importance and returns a prediction of the future. In the case of groundwater, given the current groundwater level and the expected precipitation one can predict what the groundwater level will be in the future. But clearly, the model is always only an approximation of reality, there could be other factors playing a role which are not captured in the model, the input for the model contains errors or parameters in the model are not correct. With groundwater these factors could be, the water level in nearby rivers and streams could also play a significant role, the precipitation is measured with a certain error or the parameter responsible for how much the groundwater level changes with a certain precipitation is uncertain and difficult to determine.

With the Measurements on the location the quality of the mathematical model can be evaluated; how close are the measurements to the model prediction of the current state started from the previous day. Also, the error in the measurements, the error in the model and the error in the input can be used to make perhaps an even better prediction of the groundwater level. This uncertainty in all factors is of importance, and it would be good to also quantify the uncertainty associated with the prediction. Finally, the data can be used to "train" the mathematical model. As mentioned before, the model contains parameters which are not perfect for the current area. Using the measurements, these parameters can be calibrated, giving new parameter with which the mathematical model comes closer to the measurements.

This is done in the mathematical field of Data Assimilation. From this field, a widely used framework called the Kalman Filter [15] [10] [22] is used often in hydrological applications [20]. The main idea behind the Kalman Filter is to make an optimal estimate of the state along with its uncertainty by combining data with a model. The strength of the Kalman Filter is that data from different sources can be combined with the model easily, and the uncertainty associated with the measurement, input data and model can all be taken into account. Also, the Kalman Filter can be extended a lot, such that it is also suitable for parameter calibration [8].

The Kalman Filter (KF) however has 2 drawbacks;
1) It can only work with linear systems. To this end the Extended Kalman Filter (EnKF) [14] was introduced. This method relies of linearization of the model using first order approximation of the Taylor series. However, when the non-linearity in the systems is strong this can lead to unbounded propagation of the error statistics [6].
2) When working with a larger state the computational costs of the original Kalman Filter becomes too great. To this solve these problems, the Ensemble Kalman Filter [7] was created. Using an ensemble, which captures the prediction and uncertainty by the mean and variance of the ensemble, an nonlinear model can be used directly for the prediction, and many different sources of uncertainty can be taken into account by simply adding this at the relevant points to the ensemble members. When a sufficient number of Ensemble members is chosen, the EnKF performs very similar to the KF [9].

When modelling hydrological processes the model contains physical parameters like resistances and storage capacities. These can be estimated for different soil types and areas but are often still very uncertain and rough [25]. In a particular location, these parameter estimates can be improved using the data available here, this estimating of physical parameters using data is called inverse modelling. Since these parameters play an important role in the mathematical model, using inverse modeling to improve these parameters improves the eventual predictions a lot, therefore parameter estimation along with state estimation is done often when applying Kalman Filter in hydrological frameworks[20]. [21] proposed a method for estimating the state and the parameters in parallel. Multiple methods for parameter estimation were investigated in [11], dubbed the non-iterative approach, iterative and RestartEnKF approach. [11] concluded that in groundwater problems without strong non-linearity there was not a substantial difference between the different methods, therefore the non-iterative approach was suited best since it was computationally the least demanding. For this reason, in this work the non-iterative approach was chosen. This augments the state vector to include the parameters, and updates the entire vector in the prediction step. The state and parameters are jointly updated. In this way, the model parameters are calibrated as to best fit the model.

1.2. Thesis Statement

The goal of this thesis will be to calibrate the parameters in a groundwater model with the Ensemble Kalman Filter to make accurate predictions about the future groundwater level. This Ensemble Kalman Filter is chosen since all relevant uncertainties are included and propagate over time. To this end, a groundwater model needs to be developed and solved, the relevant data needs to be obtained and the augmented Ensemble Kalman Filter for parameter estimation needs to understood and implemented.

1.3. Thesis Outline

In chapter 2 a mathematical model for the propagation of the groundwater level in a single point is derived. Here, the precipitation is considered, along with discharge and recharge from lateral and deep water struc-

tures. We will arrive at a differential equation for the change in groundwater level over time. This differential equation needs to be solved, we will give the Euler solution and, with a few assumptions in place, arrive at a semi-analytical solution.
In chapter 3, the data used is described, along with a derivation off the uncertainty associated with each data type.
In chapter 4, the different tools used from Data Assimilation will be explained. First, the regular Kalman Filter will be considered, then the transition to the Ensemble Kalman Filter will be made, and finally the augmented Ensemble Kalman Filter for the estimation of state and parameters will be given.
In chapter 5 we will combine the model, data and data assimilation techniques in our own experiments. First, the different solutions to our groundwater model are compared. Then the Kalman Filter is compared with the Ensemble Kalman Filter, since we wish to know if the approximated solutions of the Ensemble Kalman Filter are comparable with the solutions found in the Kalman Filter. After this the augmented Ensemble Kalman Filter is validated on artificial data and then used to calibrate parameters of real data. Finally, we will investigate how much better the model with calibrated parameters is compared to the uncalibrated model at predicting the groundwater level.
In chapter 6, we will reflect on the results from the previous chapter and draw our final conclusion.

2

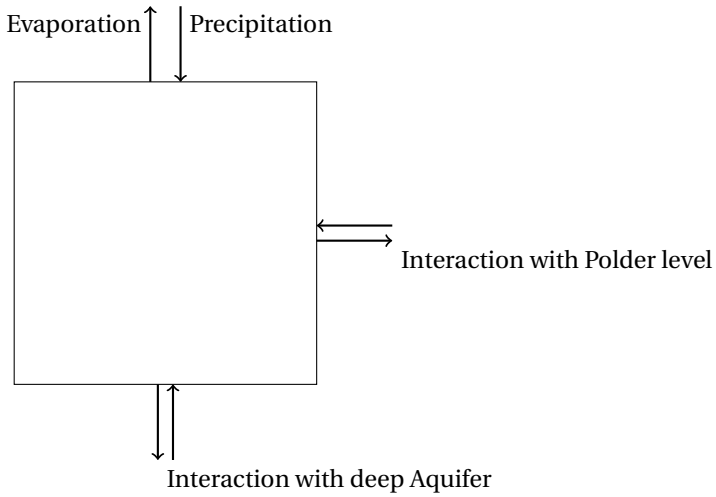
Groundwater Model

In any context where a prediction is attempted, a mathematical model is useful. In this section a model for the change in groundwater level is constructed. The model we derive in this section is the same as the model used by acacia, the document with their model and justification of input can be found here [26].

2.1. Model Derivation

First, is useful to know how we will describe the groundwater level. The measurements we will use are given in mNAP, metres of difference with NAP. The NAP stands for Normaal Amsterdams Peil and is equal to the normal sea level.
To construct a model, it is necessary to carefully consider the physics of a groundwater system. We will do this gradually, justifying each term as they make their way into our model. The model is derived from [29].

We will model the groundwater in a single location as a box. In this box groundwater enters and leaves from different sources. Below you can see the different fluxes, and after this we will describe each source separately.



We are interesting in how the groundwater changes over time, therefore we will write the groundwater model as differential equation. When nothing happens, the change over time is 0, so the most basic expression is:

$$\frac{\partial h}{\partial t} = 0$$

When it rains, the groundwater level increases, when it is warm the groundwater evaporates, therefore let p_n denote the net precipitation, precipitation minus evaporation. But how much the groundwater level changes with a certain p_n depends on how much groundwater can be stored in the soil, you can imagine that sandy

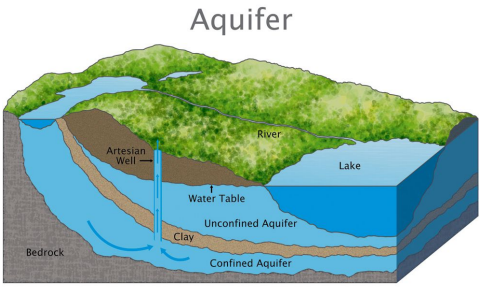


Figure 2.1: Interaction with Deep aquifer

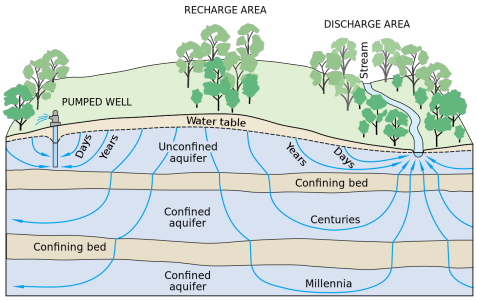


Figure 2.2: Interaction with Lateral Structures

includ

soil which is packed together very tightly leaves less room for water that soil of pebbles which leave more room for water. This is captured by the storage factor s. With these new variables the differential equation becomes:

$$\frac{\partial h}{\partial t} = 1/s(p_n)$$

There are 2 ways remaining in which the groundwater level can change: Interaction with a deep groundwater system or interaction with lateral structures such as streams. The first interaction can be seen as the horizontal interactions, the second as vertical. To understand how interaction with a deep groundwater system takes place it is important to understand the larger dynamics of groundwater. The groundwater level is usually more or less equal in a large structure called an aquifer, which is a region of soil with a nearly impermeable boundary. What this looks like can be seen in figure 2.1, here, clay and bedrock represent the nearly impermeable layers. In this image it can be seen that the confined aquifer gets his water from a river up high, and is then bordered by clay. This results in pressure upwards, therefore if a hole is made in the clay, water will begin to flow upwards from the deeper system. If this deeper aquifer would not be saturated, a hole would result in water seeping downwards. The pressure from the deep system can simply be seen as the groundwater level it wishes to attain, if a you place a well in the deep aquifer what would the resulting water level be. We denote this pressure head by h_d. How fast this interaction is depends on the resistance groundwater experiences when transferring between systems, this resistance is denoted by c_d. The equation now becomes:

$$\frac{\partial h}{\partial t} = 1/s(p_n + \frac{h_d - h}{c_d})$$

The final means that the groundwater level in a system can change is through incoming and out flowing water, rivers, canals, streams, etc as seen in figure 2.2. If the level of the streams are higher than the groundwater level, water flows into the system, with lower water level groundwater leaves the system. In the Netherlands, most ground consists of polders, in these polders the waterlevel of the flowing structures is regulated. This is water level is called the "polder peil" or polder level. Therefore, we denote the open water by h_p, and since this is again dependent on the resistance water experiences on entering and leaving the groundwater system we again introduce a resistance term c_d. The final formula would now be:

$$s\frac{\partial h}{\partial t} = \frac{h_d - h}{c_d} + p_n + \frac{h_p - h}{c_p} \tag{2.1}$$

Where the variables are:

- h: groundwater level (m)
- s: storage (-)
- h_d: pressure head from deep system (m)
- p_n: net precipitation (m/d)
- h_p: polder water level (m)
- c_p, c_d: resistances (d)

This model is - as any model - not perfect, a few points about this can be found in the discussion in chapter 6

2.2. Numerical Solution for our model

We are interested in how the groundwater table changes over time, given initial state h_0, we would like to know the change of the system in a time step, predict h_{t+1} given h_t. The easiest way to do this is to simply use the Euler's method:

$$\begin{aligned} y_t &= y_{t-1} + \Delta t y'_{t-1} \\ h_t &= h_{t-1} + \Delta t(1/s(\frac{h_d - h_{t-1}}{c_d} + p_n + \frac{h_p - h_{t-1}}{c_p})) \\ &= h_{t-1} + \Delta t(1/s(h_d/c_d - h_{t-1}/c_d + p_n + h_p/c_p - h_{t-1}/c_p)) \\ &= (1 - \frac{\Delta t}{s}(1/c_d + 1/c_p))h_{t-1} + (\frac{\Delta t}{s}\begin{bmatrix}1/c_d & 1 & 1/c_p\end{bmatrix})\begin{bmatrix}h_d \\ p_n \\ h_p\end{bmatrix} \end{aligned} \tag{2.2}$$

Were in the third step we write it in the form as required by the Kalman Filter. For Δt usually 1 day is chosen.

2.3. Drawbacks of Numerical Solution

It is known that Euler has a large truncation error, so checking if this falls within the bounds of reason is a good starting point for investigating whether our model is any good. It is known that the error from the first order Euler method can be described to be

$$E(y_t) = 1/2 y''_t (\Delta t)^2 + O((\Delta t)^3)$$

For our model this becomes:

$$E(h_t) = 1/s * (-1/c_d - 1/c_p) + O(1)$$

However, since the resistances are all very big, we expect the error to be still reasonable. This can be seen as the fact that the system changes slowly, therefore numerical solution with time step of a day is expected to capture the system behaviour correctly.

2.4. Semi-Analytical Solution

With a few assumptions in place, it is actually possible to solve this differential equation exactly. This can be done using the exponential integrator method explained here [13]. The idea is, if you can write the differential equation in the following form:

$$\begin{aligned} y' &= -Ay + N(y) \\ y(n) &= y_n \end{aligned}$$

Then, for $y_{t+1} = y_t + \Delta t$ you can write the exact solution as:

$$y_{n+1} = e^{-A\Delta t} y_n + \int_0^{\Delta t} e^{-(\Delta t - \tau)A} N\big(y(t_n + \tau)\big)\, d\tau.$$

If you furthermore assume that $N\big(y(t_n + \tau)\big)$ is constant over the entire interval then the solution becomes:

$$y_{n+1} = e^{(-A\Delta t)} y_n + 1/A(1 - e^{(-A\Delta t)})N$$

Very nice, we can indeed write our model in form needed:

$$\begin{aligned} s\frac{\partial h}{\partial t} &= \frac{h_d - h}{c_d} + p_n + \frac{h_p - h}{c_p} \\ \frac{\partial h}{\partial t} &= -1/s(1/c_d + 1/c_p)h + 1/s(h_d/c_d + p_n + h_p/c_p) \end{aligned} \tag{2.3}$$

With $N = 1/s(h_d/c_d + p_n + h_p/c_p)$.
Under the assumption than N is constant for each day, we can obtain the exact solution for h_{t+1} like this

where, for the readers sake, we let $1/c = (1/c_d + 1/c_p)$:

$$\begin{aligned} h_{t+1} &= e^{(-\Delta t/cs)} h_t + \frac{1}{1/cs} 1/s(h_d/c_d + p_n + h_p/c_p)(1 - e^{(-\Delta t/cs)}) \\ &= e^{(-\Delta t/cs)} h_t + (c(1 - e^{(-\Delta t/cs)}) \begin{bmatrix} 1/c_d & 1 & 1/c_p \end{bmatrix}) \begin{bmatrix} h_d \\ p_n \\ h_p \end{bmatrix} \end{aligned} \tag{2.4}$$

The assumption that N is equal over the entire interval is not really realistic, a discussion of this assumption can be found in the Discussion in chapter 6.
Also, we add the constraint that h_{t+1} is equal to the minimum of the output as specified above **or the surface level**, since the groundwater level cannot be above ground.

We now have 2 solutions of the model, if they both behave correctly we expect that their solutions look (nearly) identical. This is a good way to see if either solution is correct, if they are similar than we can be sure that both solutions are derived correctly and that their numerical accuracy is sufficient for our purposes. If one of the solutions is different we can have to investigate if this is due to numerical errors or something else, and then which of the two models is showing numerical errors. Therefore, our first experiment in chapter 5 will be to compare the different numerical solutions.
The code for both model solutions can be found in the appendix B.2

3

Data

As stated above, the mathematical model requires data on multiple things. In this section it is explained where the data used comes from. Since groundwater is a relatively slowly changing system it was chosen to take a measurement once every day, since you want to use data over a long period and the calculation times to still remain relatively short. Since the model wants to use these measurements as reference, the model also requires its input information for each day. Therefore all information is gather with a time step of 1 day: $\Delta t = 1$ day. The period over which we will validate our model is 2019, so it is necessary to obtain information for each day of 2019. Also, since we are using the Kalman Filter we need to have information on the uncertainty associated with all information, to this end a derivation of this uncertainty is given for each information type. The general area where the information is coming from is the water board the Drents Overijsselse Delta (WDOD), where in the Netherlands this is can be viewed in figure 3.1.

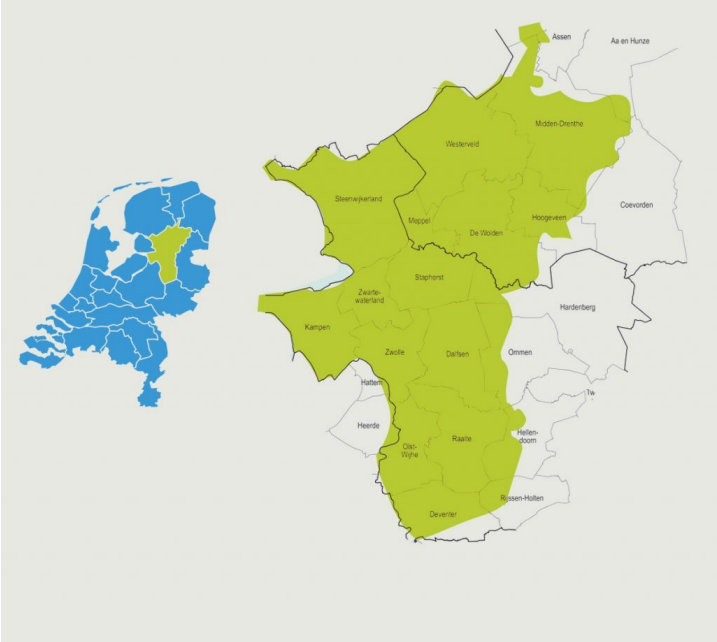


Figure 3.1: Water board Drents Overijsselse Delta Location

3.1. Measurements

The measurements can be retrieved from WDOD Kiwis, this is the database of WDODelta, here there the groundwater is measured daily in multiple locations using measurements in the ground. This data can be accessed trough the dashboard of Fixeau, the subbranch of Acacia Water which concerns itself with the Aqua Pins.

To test and calibrate our model, data from arbitrary locations which have the required information for each time step needs to be selected. We select 8 measurement points, they are located at random locations through the Water board, 3 lie close to a farm in Meppel (9016, 9111, 8907), 3 others are arbitrarily chosen (8890, 9012, 9024), for each location the measurement data for each day of 2019 is retrieved, such a series of measurements is called a time series. An example of what this looks like can be seen in figure 3.3.

To obtain a measure for the uncertainty associated with the measurements, the data from two adjacent measuring points is compared:

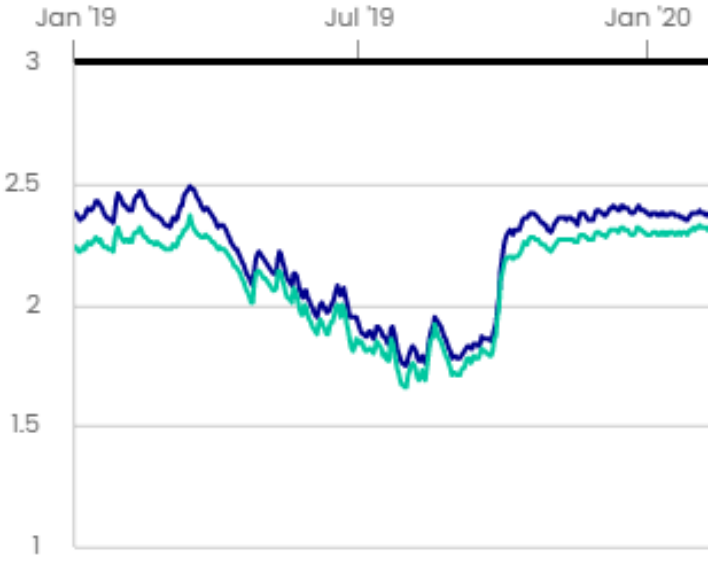


Figure 3.2: Comparison of two adjacent measuring devices, the two different time series are 21ER707D and 21ER707E from the KIWIS WDOD database, and they are located near Staphorst.

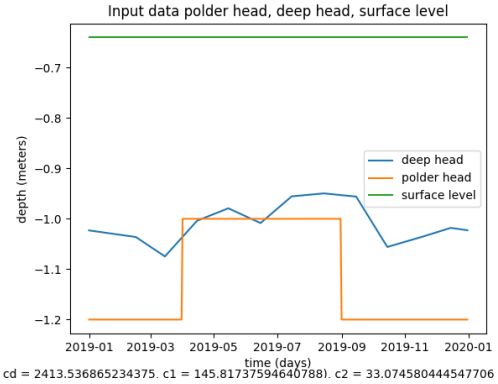
The measurements are not equal, therefore it is possible to obtain uncertainty from the above image. Suppose the true value is exactly the middle of the 2 measured values. Then, you can derive an upper bound the the standard deviation by looking for the point where the measurements deviate the most from their mean, and taking this deviation to be the standard deviation. Looking at the image we can see that the largest deviation is at the beginning of the year. Here, the difference between the 2 values is 0.14, therefore the standard deviation of the measurements is taken to be 0.07. Now the variance becomes:

$$\Sigma^y = 0.0049 \tag{3.1}$$

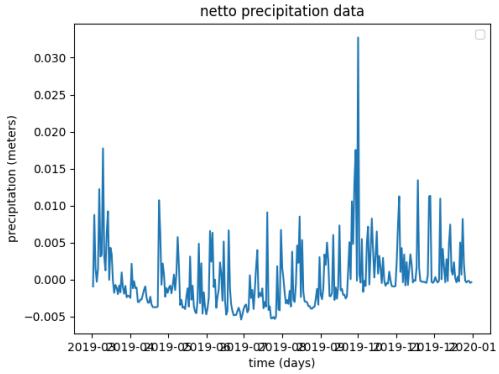
A small side note, when viewing the figure it is possible to think that the difference between the two time series comes from a systematic error rather than a random error. This is valid, and the consequences of this are discussed in the discussion in chapter 6.

3.2. Input data

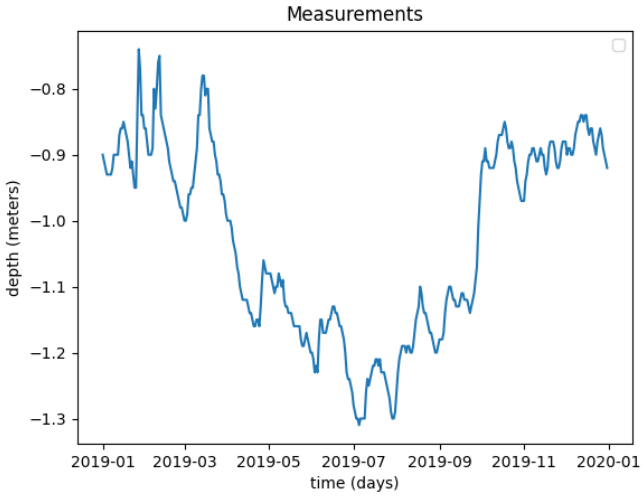
We have multiple types of input data. An example of the values the input data will take can be seen in figure 3.3. First, a look is taken at the precipitation and evaporation, since this is expected to have the highest impact on the model results. Then, the information about the polder level and the pressure from the deep aquifer is described, along with the collection of constants. Finally, the storage is retrieved.



(a) An example of the input data, not including the precipitation because this is measured in mm and not in meters.



(b) Precipitation data, given in meters since this is what the model requires.



(c) Measurement data

Figure 3.3: Example of all the relevant data, retrieved over 2019 from location 8907, a measurement point near the small town Hasselt.

3.2.1. Precipitation and Evaporation

For the precipitation and evaporation data from the Dutch KNMI (Koninklijk Nederlands Meteorologisch Instituut) is used, the RH24 [17] and EV24 [16] respectively. The data for the Precipitation is initially obtained using data from 2 radars placed in Den Helder and in Herwijnen, and has a resolution of 1000x1000 square meters. The radars work by transmitting electromagnetic radiation, and from the amount of particles and the time they take to return back to the antenna it is possible to obtain information of the location and amount of droplets in the air, which can be translated to the precipitation. This radar data is then corrected by the 325 rain gauges installed across the country by the KNMI. A rain gauge in a cylinder that collects rainwater in a particular location over a certain amount of time. This is very accurate information about the rainwater and can therefore be used as very accurate measurements to correct the radar information. The rainwater is given as the sum over a 24 hour period, and since the model also uses time steps of a single day, this data is used immediately as model input.
For evaporation the Makkink Evaporation is calculated and interpolated with a resolution of 1000 m. This requires the daily temperature and incoming shortwave radiation [12].
Since the precipitation is rarely constant over a single day and since it is also usually varies in an area of 1000x1000 square meters, the data is not 100 % accurate. This inaccuracy needs to be taken into account when using the input data. According to [28], an optimal combination of rain gauge and radar data allows for an accuracy up to 5 %. The evaporation is dependent on two terms which can be measured very accurately, therefore the error associated with this is way lower than the 5% for the precipitation, so this error term is ignored.

3.2.2. Deep Groundwater, Resistances and Polder levels

For the deep head, the result of a deep groundwater model was used, the model used is called MIPWA (version 4.0) and the data can be found here [5]. The uncertainty associated with this model is not public information, and since this value changes little over time and becomes indirectly calibrated by the parameter calibration, this uncertainty is not that important and we set it to 0. An uncertainty of 0 is never realistic however, so it is important to note that this is an assumption.

The information about the Surface level is the absolute height of the surface in this location compared to NAP.

The information about the resistances the following data set withing the NHI (Nationaal Hydrologisch Instrumentarium) [4]. This is a map of the Netherlands with for every location 315 parameters for the soil. All these parameters are computed using a mathematical model and their resolution is 1:50.000, therefore it is expected that these parameters are quite crude. For this reason, and their large impact on the model results, calibration of these parameters is of real importance.

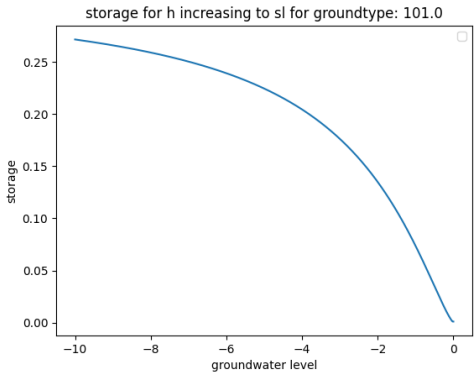
The polder levels are obtained from the Waterboard [30], they try to maintain this different level throughout the summer and the winter, so it is possible that the level deviates. However, it is not necessary to quantify this uncertainty since this will also be indirectly captured by the calibration of the resistance parameter associated with the polder level.

3.2.3. Storage

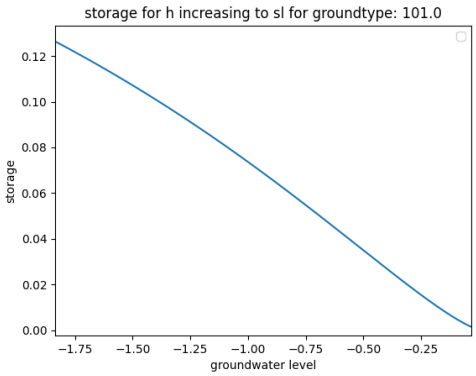
The storage factor is a measure of how much the groundwater level changes in response to incoming or outgoing water. It can be interpreted as the water volume necessary to increase the groundwater level a little. This is determined for different areas of the Netherlands depending on the type of ground available here. Such a region is defined in the BOFEK map [24], this divides all of the Netherlands in different regions, where each region is supposed to behave according to the same hydrological constants. For each BOFEK region, a model for the storage was created based on the hydrological constants present in this region [26]. The model is based on the realization that the possible groundwater storage increases as you get deeper in the soil since here there is no soil moisture to compete with the groundwater level, the storage is a function of h. However in the model s was given as a constant. This is because otherwise the numerical solution would become way more complex, and we can circumvent this by calculating s(h) in each individual time step and then treating it as constant at that time, this is again not a perfect solution and is discussed in the discussion in chapter 6. Since h is given relative to NAP and we want h relative to the surface, we subtract h from the surface level. Here it is assumed that h is below the surface, since otherwise the result could become complex valued. The function for the storage has the form:

$$s(h) = \frac{a * b + c * (sl - h)^d}{b + (sl - h)^d} \tag{3.2}$$

Where the constants a, b, c, d were computed for each BOFEK region. Since we want the storage to be greater than 0, we clip the storage above 0.001. To get a general idea what this function looks like we will plot this function for $sl - h$ decreasing from 10 to 0.



(a) storage function, plotted from a groundwater level of -10 to 0, relative to the surface level.



(b) Zoomed look at the storage function, here it can be seen that on a small section, the function is nearly a straight line

Figure 3.4: Storage function for BOFEK region 9012

Whilst the function in equation 3.2 looks very complex, figure 3.4b shows that on a groundwater range of 1.5 the function becomes very close to a straight line. This is what we would expect, as you get closer to the surface the storage steadily decreases until it is 0 at the surface. The groundwater range of 1.5 is justified by the example of the measurements as seen in figure 3.3c, these are all in a range of 1. Therefore, a future improvement could be to simplify the function for the storage to just a straight line, more on this in the discussion in chapter 6.

3.3. Remarks about the data

We look at all the data for each selected point, and we note some similarities. The measurements all fall in a range of 1.5/2 meters and there is always a lower groundwater table in the summer than in the spring and winter. The polder head is a constant value but sometimes changes from a winter level to a summer level and back twice a year. The deep head fluctuates less than the groundwater, and usually has a higher value than the polder level, because they stay relatively constant we expect this term to raise the predicted groundwater level by a constant throughout the year. The netto precipitation changes a lot from day to day and has a negative baseline in the summer due to the increased temperature leading to increased evaporation. Also, we note that the resistance coefficients can differ a lot in different regions. They can be very big, sometimes they have a value greater than 50000, then you expect their corresponding variable (deep pressure, polder level) to have very little impact on the changing groundwater. The input data is loaded using python, the code for this is found in appendix B.1

4

Data Assimilation

Having created the mathematical model and collected all the data previously, the final object we need before we can start with combining mathematics and data to calibrate parameters is the Data Assimilation scheme. In this section the methods we will use from Data Assimilation will be explained. First, a general description of the Kalman Filter will be given. This will use a Kalman Gain matrix which will serve as way to combine the measurement with the data optimally. Here "optimally" will mean that the result has a lowest possible variance. The mathematics behind the optimal fusing of different estimates can be read in appendix A. After the Kalman Filter, the transition to the Ensemble Kalman Filter is given. This will allow us to tackle larger and nonlinear problems, and allows for an easy transition to parameter estimation. The final thing then will be to describe the Ensemble Kalman Filter with augmented state which includes the parameters for parameter calibration. The notation used in this chapter is obtained from the papers [22] and [21]. All estimates will be assumed to be unbiased normally distributed random variables of the true state $\boldsymbol{x}_t$.

4.1. The Kalman Filter

The idea behind the Kalman Filter is that it combines measurements at different times and a mathematical model describing the evolution of a system over time into a single optimal estimate at each of the measurement points. The model predicts the state of a system over time, and when you have a measurement you shift the prediction using the measurement. With the knowledge from appendix A, we know how to optimally fuse different estimates. This can be used in the case when there are 2 estimates: a measurement and a model prediction. This is precisely what the Kalman Filter wants to accomplish so those mathematical tools translate nicely to our situation here.

The initial state can be denoted by a random variable $\boldsymbol{x}_0$, with mean $\hat{\boldsymbol{x}}_0$ and covariance Σ_0. The propagation of this random variable through time is then captured by a mathematical model of the form $\boldsymbol{x}_t = f_t(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t, \boldsymbol{\theta}) + \boldsymbol{w}_t$.

The idea behind the model is that in a single time step the state of a system is a function of the state at a previous time step and the control inputs (relevant changes in the environment) applied to the system during the interval, with some random error since the model is never perfect. Here, $\boldsymbol{u}_t$ represents the control input, $\boldsymbol{\theta}$ the parameters used in the model and $\boldsymbol{w}_t \sim N(0, \Sigma_t^m)$ is a random variable with Σ_t^m the covariance matrix of the model error. For the Kalman Filter to work, it is required that the relationship between the previous state and the control input is linear, in that case the function takes the following shape:

$$\boldsymbol{x}_t = F_t \boldsymbol{x}_{t-1} + B_t \boldsymbol{u}_t + \boldsymbol{w}_t \tag{4.1}$$

Where F_t and B_t are time-dependent matrices, F_t the effect the system has on the previous estimate and B_t the effect on the input. Both F_t and B_t use the parameters $\boldsymbol{\theta}$.

However propagation of a random variable can be split in two to make things easier. To this end the mean and covariance are updated separately over time, which together capture the entire random variable.

$\hat{\boldsymbol{x}}_{0|0}$ is an estimate of the initial state, with uncertainty captured by covariance matrix $\Sigma_{0|0}$, the model than becomes of the form:

$$\hat{\boldsymbol{x}}_{t|t-1} = F_t \hat{\boldsymbol{x}}_{t-1|t-1} + B_t \boldsymbol{u}_t \tag{4.2}$$

Now, $\hat{\boldsymbol{x}}_{t|t-1}$ is a prediction as generated by the model, also called an **a priori state estimate**, and $\hat{\boldsymbol{x}}_{t-1|t-1}$ is the model prediction at the previous time step fused with the measurement at that time, also called the **a posteriori state estimate**.
The **a priori estimate error covariance** is obtained from the previous **a posteriori estimate error coveriance** using the same model:

$$\Sigma_{t|t-1} = F_t \Sigma_{t-1|t-1} F_t^T + \Sigma_t^m \tag{4.3}$$

The above equation follows from the mathematical in appendix A, see the section on the combination of 2 vector estimates.
We have now generated a prediction for the current time step. Next, we will fuse this prediction with the measurement. Let us denote a measurement at time t be denoted by $\boldsymbol{y}_t$, with covariance matrix covariance matrix Σ_t^y. For the fusing an optimal combination can be found using the covariance matrices as described in appendix A. This combination is called the Kalman Gain matrix:

$$K_t = \Sigma_{t|t-1}(\Sigma_{t|t-1} + \Sigma_t^y)^{-1} \tag{4.4}$$

Then, to compare the state and the measurement, translate the state towards the measurements, this is called the Measurement Equation:

$$\hat{\boldsymbol{y}}_t = H_t \hat{\boldsymbol{x}}_{t|t-1} \tag{4.5}$$

Where H_t is called the **Observation Matrix**.
And now, fuse the estimates into a single optimal estimate using the Kalman Gain matrix gives us the a posteriori state estimate, this becomes:

$$\hat{\boldsymbol{x}}_{t|t} = \hat{\boldsymbol{x}}_{t|t-1} + K_t(\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \tag{4.6}$$

Here, the difference $\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t$ is called the Innovation, and reflects how much the predicted measurement differs from the actual measurement. The larger their difference the more the prediction is corrected, taking into account the magnitude of the Kalman Gain matrix.
Finally, the new a posteriori estimate error covariance matrix is obtained:

$$\Sigma_{t|t} = \Sigma_{t|t-1}(\Sigma_{t|t-1} + \Sigma_t^y)^{-1}\Sigma_t^y = K\Sigma_t^y \tag{4.7}$$

Why this estimate is optimal and why the Covariance matrix is found in this way can be found in appendix A. The following form is also shown sometimes , here it is used that $A_1 + A_2 = I$

$$\begin{aligned}
\Sigma_{t|t} &= \Sigma_{t|t-1}(\Sigma_{t|t-1} + \Sigma_t^y)^{-1}\Sigma_t^y \\
&= ((\Sigma_{t|t-1})^{-1} + (\Sigma_t^y)^{-1})^{-1} \\
&= \Sigma_t^y(\Sigma_{t|t-1} + \Sigma_t^y)^{-1}\Sigma_{t|t-1} \\
&= (I - K)\Sigma_{t|t-1}
\end{aligned} \tag{4.8}$$

The flow of the Kalman Filter can be seen below, here Σ_t^y is replaced by R_t and $\boldsymbol{y}_t$ by $\boldsymbol{z}_t$.

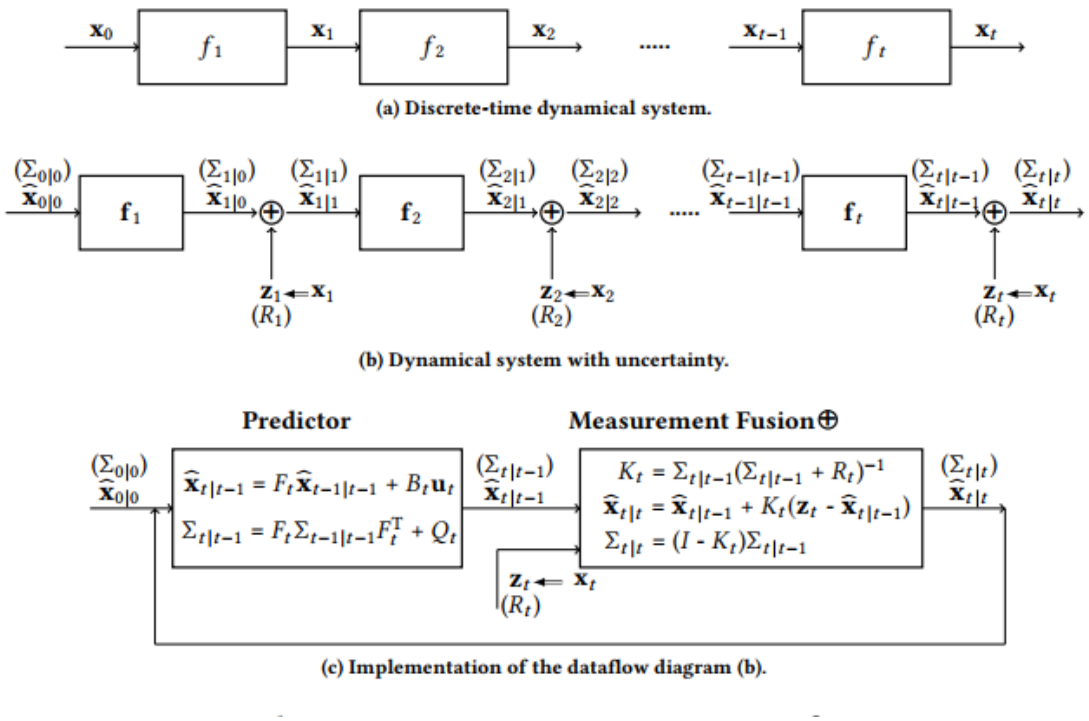


Figure 4.1: Flow of Kalman Filter, this figure is obtained from [22]

The Kalman Filter however has 2 drawbacks;
1) It can only work with linear systems, this goes wrong when considering the problem of parameter estimation, the estimated parameters interact with the state in a nonlinear way. To solve this problem, initially the Extended Kalman Filter [14] was introduced. This method relies on linearization of the model using first order approximation of the Taylor series. However, when the non-linearity in the systems is strong this can lead to unbounded propagation of the error statistics [6].
2) When working with a larger state the computational costs of the original Kalman Filter becomes too great since the covariance matrices to be computed will become very large.
To this solve these problems, the Ensemble Kalman Filter [7] was created. Here, many different sources of uncertainty can be taken into account, large problems can be handled easily and parameter estimation is readily available.

4.2. The Ensemble Kalman Filter

The Ensemble Kalman Filter works with an ensemble, and propagates this ensemble over time. An ensemble is a collection of predictions of a state, the mean of the ensemble is expected to be the best prediction, the variance of the ensemble captures the uncertainty associated with the prediction. For example, suppose you take the standard normal distribution. You can represent this by the probability density function, or you can approximate this with a collection (ensemble) of 100 samples. When computing the sample mean and variance you can then assume that you obtain (an approximation of) the standard normal distribution back. The idea behind this is that you can replace the actual covariance matrices with the sample covariance matrix, and in large problems this is way less expensive to compute. To this end, all uncertainty is added as noise to the ensemble members.

The following section is based on the mathematics as stated in [21]. The initial ensemble is computed by taking samples from a normal distribution $N(m_0, V_0)$ where m_0 is the first measurement and V_0 is the initial variance. Let n denote the ensemble size, we take n samples $\boldsymbol{x}_0^1, \cdots, \boldsymbol{x}_0^n$. Now, each member of the ensemble is propagated through time in the same way as in the Kalman Filter, with a few differences:

First, calculate the a priori state estimate with the evolution equation as before, however now the uncertainty associated with the model prediction is included in the estimate:

$$\boldsymbol{x}^i_{t|t-1} = f(\boldsymbol{x}^i_{t-1|t-1}, \boldsymbol{u}^i_t, \boldsymbol{\theta}) + \boldsymbol{w}^t_i, \qquad \boldsymbol{w}^i_t \sim N(\boldsymbol{0}, \Sigma^m_t) \tag{4.9}$$

Where now, linearity is not required and with $\boldsymbol{\theta}$ the parameters as used in the model. u^i_t is now also given for an ensemble element, this is attained by adding noise to each of the forcing data:

$$\boldsymbol{u}^i_t = \boldsymbol{u}_t + \boldsymbol{\xi}^t_i, \qquad \boldsymbol{\xi}^i_t \sim N(\boldsymbol{0}, \Sigma^u_t) \tag{4.10}$$

If $\boldsymbol{x}$ is not directly related to the observation, we can transform $\boldsymbol{x}$ to the measured values $\boldsymbol{y}$ as:

$$\hat{\boldsymbol{y}}^i_t = h(\boldsymbol{x}^i_{t|t-1}, \boldsymbol{\theta}) \tag{4.11}$$

Since measurements can also be uncertain, for each ensemble element we introduce uncertainty in the measurement as follows:

$$\boldsymbol{y}^i_t = \boldsymbol{y}_t + \boldsymbol{\eta}^i_t, \qquad \boldsymbol{\eta}^i_t \sim N(\boldsymbol{0}, \Sigma^y_t) \tag{4.12}$$

Now the postiori state estimates are obtained in the same way as in the regular Kalman Filter:

$$\boldsymbol{x}^i_{t|t} = \boldsymbol{x}^i_{t|t-1} + K_t(\boldsymbol{y}^i_t - \hat{\boldsymbol{y}}^i_t) \tag{4.13}$$

Where the Kalman Gain matrix is given by:

$$K_t = \Sigma^{xy}_t (\Sigma^{yy}_t + \Sigma^y_t)^{-1} \tag{4.14}$$

What is important to note is that the Kalman Gain matrix is the same for the fusing of each ensemble member. Where Σ^y_t was given before, Σ^{yy}_t denotes the a priori error covariance matrix of the predictions $\hat{\boldsymbol{y}}^i_t$. This matrix is given exactly for vector random variables X as $E((X - E(X))(X - E(X))^T)$, but since we are using a sample simply to $\overline{(X - \overline{X})(X - \overline{X})^T}$.
Therefore the a priori error covariance matrix becomes:

$$\Sigma^{yy}_t = \frac{1}{n-1} \sum_{i=1}^{n} [(\hat{\boldsymbol{y}}_t - 1/n \sum_{i=1}^{n} \hat{\boldsymbol{y}}^i_t)(\hat{\boldsymbol{y}}_t - 1/n \sum_{i=1}^{n} \hat{\boldsymbol{y}}^i_t)^T] \tag{4.15}$$

analogously, Σ^{xy}_t the a priori cross covariance matrix becomes:

$$\Sigma^{xy}_t = \frac{1}{n-1} \sum_{i=1}^{n} [(\boldsymbol{x}^i_{t|t-1} - 1/n \sum_{i=1}^{n} \boldsymbol{x}^i_{t|t-1})(\hat{\boldsymbol{y}}_t - 1/n \sum_{i=1}^{n} \hat{\boldsymbol{y}}^i_t)^T] \tag{4.16}$$

The code associated with both these filters is implemented in Python, the code can be found in appendix B.

4.3. Parameter and State Estimation

Until now we have only been concerned in estimating the state, but we would like to improve the parameters from the model as well. This can be done using the method of **state augmentation** [18]. Here, the state of the system gets updated such that the parameters are present in the state vector. Now, the initial ensemble members look as follows:

$$\boldsymbol{x}^i_0 = \begin{bmatrix} \boldsymbol{x}^i_{s,0} \\ \boldsymbol{\theta}^i_0 \end{bmatrix} \tag{4.17}$$

Where $\boldsymbol{x}_s$ denotes the state and $\boldsymbol{\theta}$ the parameters.
$\boldsymbol{\theta}^i$ is initialized as:

$$\boldsymbol{\theta}^i = \boldsymbol{\theta} + \boldsymbol{\nu}^i \tag{4.18}$$

with $\nu^i_j \sim N(0, \Sigma^p)$. This means, the parameters are also given as elements of the ensemble with noise associated with them.
An a priori state and parameter estimate is given as before, only now the parameters are included in the ensemble member and are retrieved from the previous time step:

$$\boldsymbol{x}^i_{t|t-1} = \begin{bmatrix} \boldsymbol{x}^i_{s,t|t-1} \\ \boldsymbol{\theta}^i_{t|t-1} \end{bmatrix} = \begin{bmatrix} f(\boldsymbol{x}^i_{t-1|t-1}, \boldsymbol{u}^i_t, \boldsymbol{\theta}^i_{t-1}) + \boldsymbol{w}^t_i \\ \boldsymbol{\theta}^i_{t-1|t-1} * I \end{bmatrix}, \qquad \boldsymbol{w}^i_t \sim N(\boldsymbol{0}, \Sigma^m_t) \tag{4.19}$$

Above, noise is added to $\boldsymbol{u}_t^i$ as before. Also, noise is added to the measurement exactly as above with uncertainty Σ^y. Also, the a priori parameter estimate is the same as the a posteriori parameter estimate from the previous step
Now, the a priori estimates are transformed to the measurements, to be comparable with the measurements. This is done by the function h:

$$h(\boldsymbol{x}_{t|t-1}^i, \boldsymbol{\theta}) = \boldsymbol{x}_{s,t|t-1}^i = \hat{\boldsymbol{y}}_t^i \tag{4.20}$$

Now finally, the a priori estimates are now fused with the measurement to obtain the a posteriori estimates as follows:

$$\boldsymbol{x}_{t|t}^i = \begin{bmatrix} \boldsymbol{x}_{s,t|t}^i \\ \boldsymbol{\theta}_{t|t}^i \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_{s,t|t-1}^i \\ \boldsymbol{\theta}_{t|t-1}^i \end{bmatrix} + K_t(\boldsymbol{y}_t^i - \hat{\boldsymbol{y}}_t^i) \tag{4.21}$$

Since the Kalman Gain is the same for each ensemble member, now the parameters are updated with a size corresponding to how close the model solution was to the measurement. When the model and measurement are very close the, the parameters stay nearly the same, if they are far apart, the parameter are changed by a larger value.

the Kalman Gain matrix is still:

$$K_t = \Sigma_t^{xy}(\Sigma_t^{yy} + \Sigma_t^y)^{-1} \tag{4.22}$$

Where Σ_t^{yy} is the a priori state error covariance, the variance of the $\hat{\boldsymbol{y}}$. Σ_t^{xy} is now the covariance between the prediction of both the state and parameters and the state. This looks like this:

$$\Sigma_t^{xy} = \frac{1}{n-1}\sum_{i=1}^{n}[(\boldsymbol{x}_{t|t-1}^i - 1/n\sum_{i=1}^{n}\boldsymbol{x}_{t|t-1}^i)(\hat{\boldsymbol{y}}_t - 1/n\sum_{i=1}^{n}\hat{\boldsymbol{y}}_t^i)^T] \tag{4.23}$$

In this final matrix the covariance between the state and itself, and the state with the parameters is captured. When investigating the Kalman Gain matrix for 1 dimensional state with n parameters, it can be noticed that it looks like this:

$$K_t = \begin{bmatrix} K_{s,t} \\ K_{\theta_1,t} \\ K_{\theta_2,t} \\ \vdots \\ K_{\theta_n,t} \end{bmatrix} \tag{4.24}$$

Where $K_{s,t}$ has the same value as the Kalman Gain matrix for the regular Ensemble Kalman Filter. The $K_{\theta_i,t}$ determine how much the parameters are changed. If its value is big and the function deviates a lot from the measurement, the associated parameter value is changed a lot.

5

Experiments

In this chapter the knowledge accumulated in the previous chapters will be used in the context of groundwater modelling in a single spot. First, The mathematical model described in chapter 2 will be used with the data from chapter 3. There, two solutions for this model were derived. Therefore, first, we will compare the two numerical solutions found in chapter 2 to verify that they are both more or less correct, and we will make the decision of which solution to use in the remainder of this project. Then, the Kalman Filter and the Ensemble Kalman Filter described in chapter 4 can be computed. We will again compare the two results, since first the problem is small and exactly in the linear form described in chapter 4, the Kalman Filter is the optimal solution, but how does the Ensemble Kalman Filter compare? After this we can proceed with the Ensemble Kalman Filter, and we will use this in the context of Parameter Calibration. To test the ability of the Ensemble Kalman Filter to estimate parameters, Artificial Data will be created and the quality of the parameters obtained will be quantified. Then, we will use the Ensemble Kalman Filter to calibrate the parameters for actual data, and see how much better the estimation becomes. From this it is easy to transition to our final goal, estimate parameters on a piece of the data and use the calibrated parameters to make a prediction about the groundwater level for the rest of the data.

5.1. Comparison of model Solutions

In this section we will simply run the two solutions (the numerical and the semi-analytical) with the input values specified before, and we will compare them with the measurements. The results are obtained using python, the actual code can be found in appendix B. The goal is to see if the solutions are similar. If this is the case, we can be quite sure that the both solutions are reasonably correct. The performance will be judged on a visual inspection of the results, and by computing the RMSE (root mean square error) between the Euler results and semi-analytical results. When we simply run the different models with input parameters as obtained for the arbitrarily chosen location 9111 we get the following results:

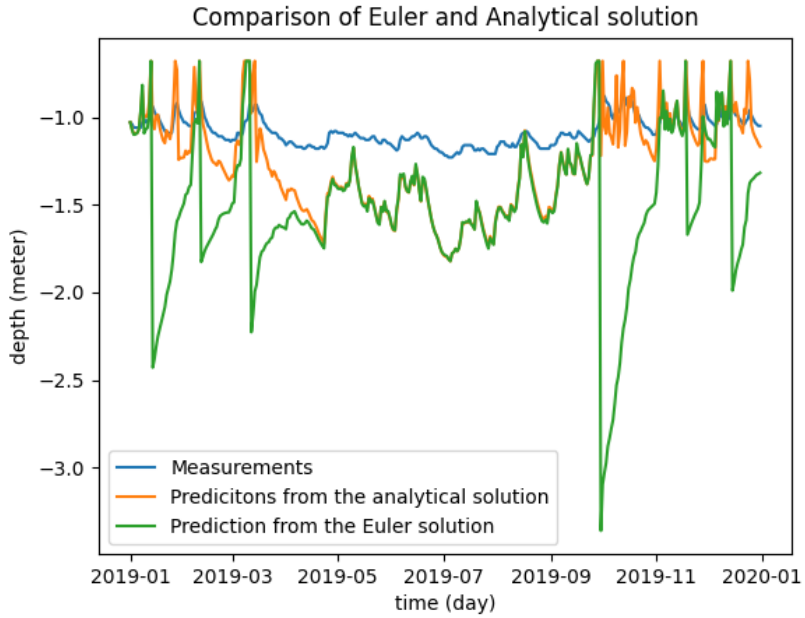


Figure 5.1: Model solved with both the Euler method and semi-analytically, they deviate sharply at certain points. Time step of 1 day.

At certain points the two solutions are quite similar and at certain points they diverge very strongly. Since they overlap at certain points, here the input and the previous value yield the same prediction so the derivation of both solution is indeed correct. At some points they are not similar, this can also be seen in the RMSE between the two solution which is still quite high: $0.51m$. Since Euler is probably the most unstable of the two solutions, we investigate the truncation error of this solution, this can be seen in figure 5.2

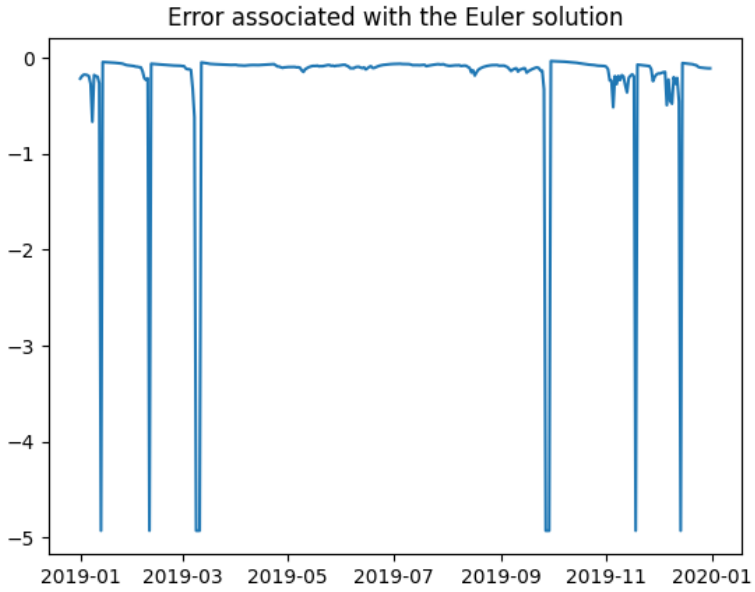


Figure 5.2: The truncation error of the Euler solution. Interestingly, the spikes in the error are exactly the points where the Euler solution deviates from the semi-analytical solution

As seen in figure 5.2, the Truncation error attains very large values at some points. Interestingly, these are exactly the points where the Euler solution deviates from the semi-analytical model as seen in figure 5.1! Since

the Euler model follows the semi-analytical solution except for the points where the truncation error is large, it is not unreasonable to that the semi-analytical solution works more accurately than the Euler solution. To further test this, in figure 5.3 the Euler solution is now computed with $\Delta t = 1$ hour.

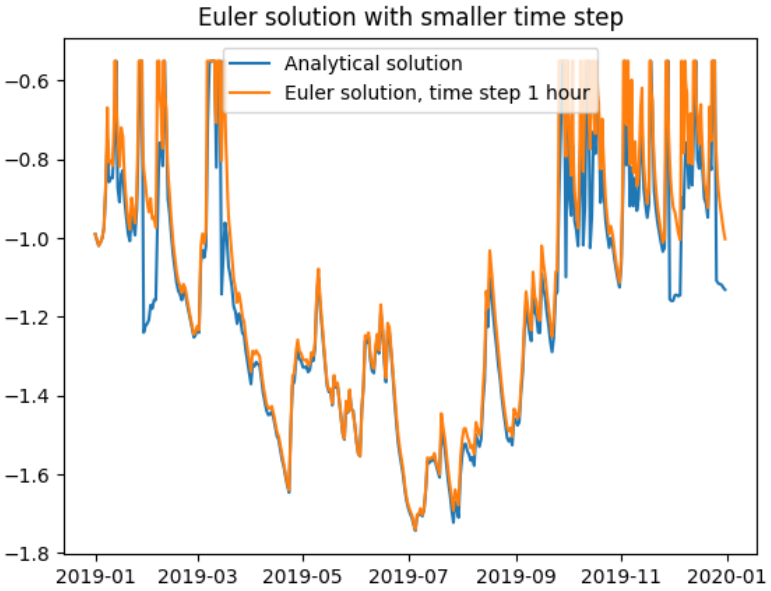


Figure 5.3: Euler solution with a smaller time step compared with the semi-analytical solution.

As seen above, the Euler solution is now a little bit better than the semi-analytical solution. This is because there is also a numerical error in the semi-analytical solution. What is the reason for this can also be found in the discussion in chapter 6. That they both converge to the same solution can now be investigated by also taking smaller time steps for the semi-analytical solution. This is found in figure 5.4

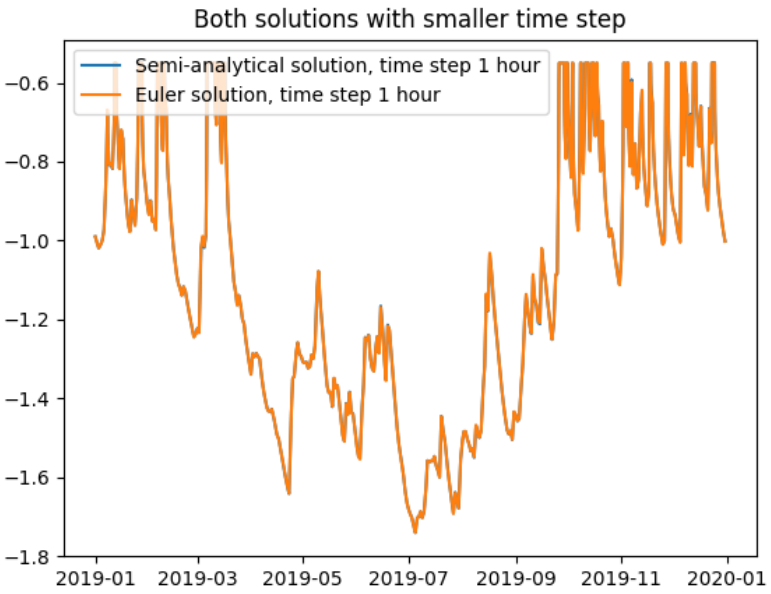


Figure 5.4: Both the semi-analytical and Euler model solution with time step of an hour

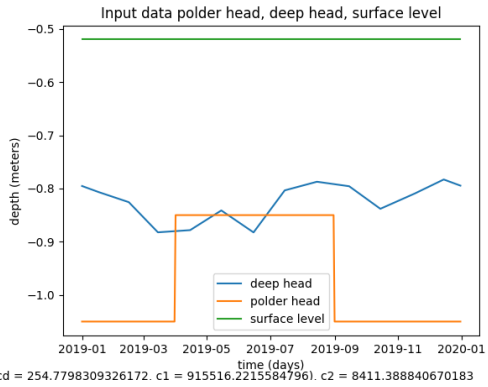
It is good to see that they both converge towards the same value for all time steps. This now tells us that both solutions provide the same answers, but they only differ because of numerical errors. The semi-analytical model takes input over the entire day, except for the storage, this was now calculated at each time step again. Since this already improves the model, this tells that taking constant storage over a single day is imperfect.

Since the input data is now all given with time steps of a single day, improvements with a smaller time step could also use other input data. These improvements would result in a more accurate model, but this is out of the scope of this project, Since the semi-analytical solution is shown to produce better results with a time step of one day, we will use the semi-analytical solution further in this report.

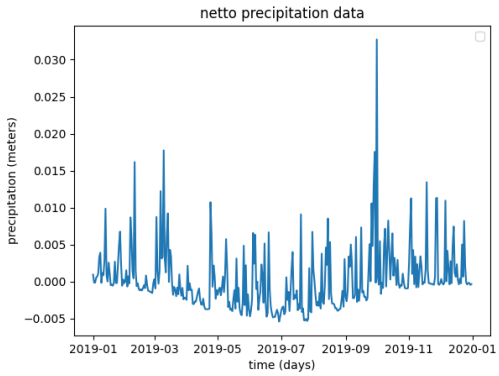
The predictions of the model solutions seems to follow the measurements quite accurately. It is not perfect of course; for example in the summer when there is less precipitation the groundwater level is structurally too low and it seems the model is always more spiky than the measurements. These could simply be problems with the parameters, it would seem that if the water in the ground didn't react so severely to changes in input the result would be better. This could easily be attributed to a storage that is too low. However, given that the parameters are uncertain and nothing yet is tweaked, the model follows the measurements quite well for now. The Root Mean Square error between the model and the measurements is $0.275m$. This would seem to indicate that the model is good to use for the rest of the project. Now that we know that the model we created is follows the measurements enough to suggest sufficient physical correctness, we can use this model to implement both the regular Kalman Filter and the Ensemble Kalman Filter and compare between these two methods.
Code for models in Appendix B.2
Code for the comparison in Appendix B.3

5.2. Comparison of Filtering methods

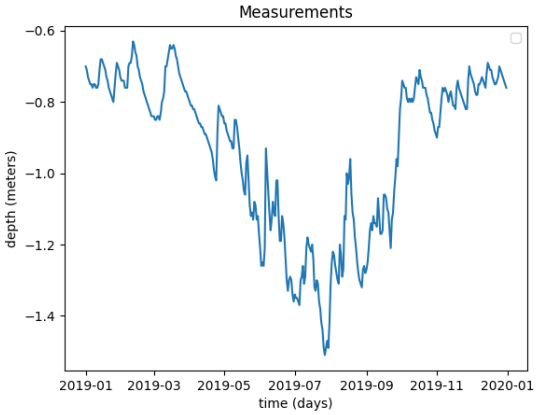
All the information from the previous sections will now be used to implement the Kalman Filter and the Ensemble Kalman Filter. The uncertainties associated with most factors are already derived in chapter 3, the final elements the Filtering methods need are derived below. In this section, location 9012 was used. The input and measurements of this location can be seen in figure 5.5



(a) Input data from location 9012



(b) Precipitation data from location 9012



(c) Measurements from location 9012

Figure 5.5: All information from location 9012, this is the information we will use in this section about data assimilation

5.2.1. Starting Values and Uncertainties

To start using the Filters, the first thing that is necessary is an initial state x_0, which is chosen to be the first measurement. Also, an initial variance has to be chosen, here this is equal to the measurement uncertainty.

Also, the Filters allows for multiple sources of uncertainty. One uncertainty not yet derived is the uncertainty associated with the model. Since we know that this model is always imperfect and for now still contains uncalibrated parameters we expect this to be quite high, a lot higher than the measurement uncertainty, which are assumed to come very close to the reality in comparison with the model. Therefore, for the model uncertainty we can use the measurements as a reference point. We look for the place when the model accumulates the most difference from the solution in the smallest time, this then given an upper bound for the model error. A suitable candidate can be seen around the month march. When enlarging the figure it can be seen that the model accumulates up a difference of 0.58 with the measurements in four days. Therefore, we say the standard deviation associated with 4 days is equal to 0.58, $s.d_4 = 0.58$. The variance associated with for days is $(s.d_4)^2 = 0.336$, the variance of one day therefore becomes $(s.d_1)^2 = 0.0841$ and the standard deviation of a single day becomes: $s.d_1 = 0.29$. A few side notes to the derivation of the uncertainties can be found in the discussion in chapter 6.

Now, all the uncertainties are known, we place them in table 5.1:

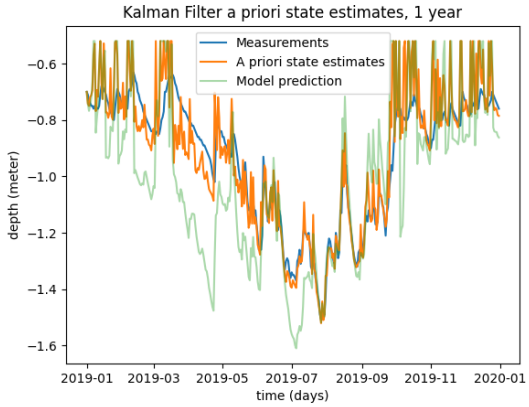
Name	Symbol	Variance	Standard Deviation
Model uncertainty	Σ_t^m	$0.0841\ m^2$	$0.29\ m$
Measurement uncertainty	Σ_t^y	$0.0049\ m^2$	$0.07\ m$
Precipitation uncertainty	Σ_t^u	$0.0025 * p_t^2\ m^2$	$0.05 * p_t\ m$

Table 5.1: Table containing the different sources of uncertainty

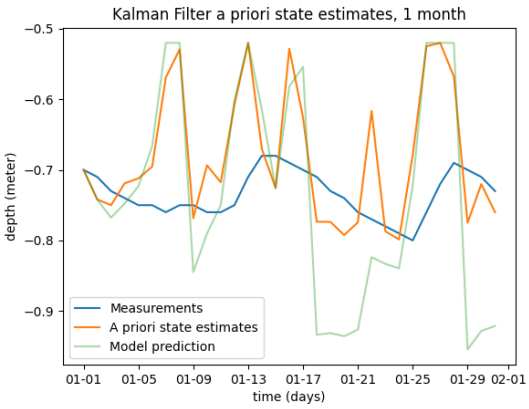
The Kalman Filter will only be able to use the model uncertainty and the measurement uncertainty, the Ensemble Kalman Filter easily handles all the sources of uncertainty. With this information known, we can begin our investigation about the comparison between the regular and Ensemble Kalman Filter.

5.2.2. Compare regular and Ensemble Kalman Filter

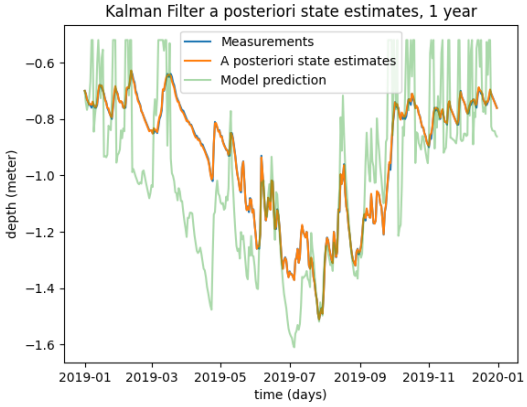
First, we will look at the Kalman Filter and Ensemble Kalman Filter results for the estimation of the state using the uncertainties from table 5.1, the mathematical model from chapter 2 with data from chapter 3. Since input uncertainty is not easy to implement inside the regular Kalman Filter, first we will ignore this uncertainty for both Filters to keep the results comparable. For the Ensemble Kalman Filter, an ensemble size of 200 was chosen. The mathematical background for the Filters is described in chapter 4. We will use python to implement the Filters. The implementation for all code can be found in appendix B.
For the regular Kalman Filter, we obtain the following result:



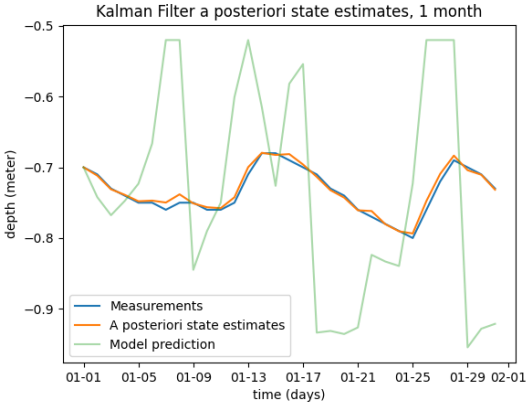
(a) Kalman Filter a priori state estimates over the entire year 2019. Each step starts with the previous a posteriori state estimate, which is nearly the measurement, and then computes the model prediction. So, it can only deviate as much as the model error in a single step, there are no accumulation of model errors.



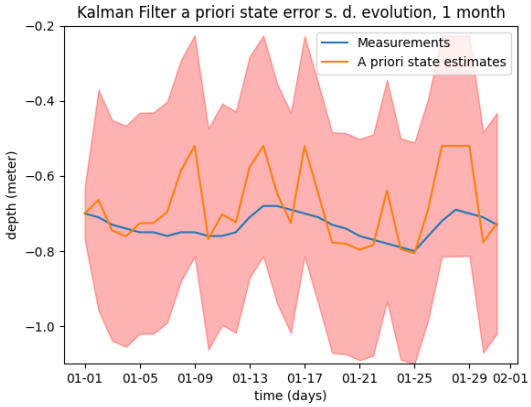
(b) Kalman Filter a priori state estimates over the month January of the year 2019. Same points as in figure 5.6a, the estimates deviate from the measurements, but less since there is no accumulation of model errors.



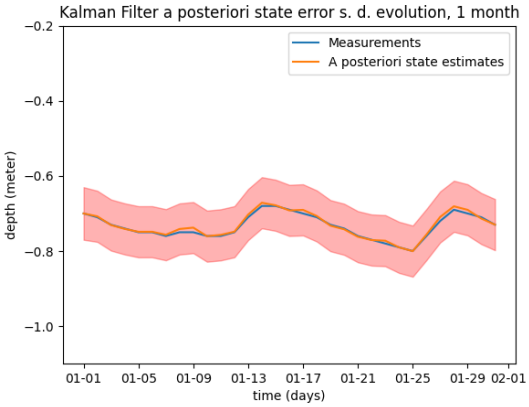
(c) Kalman Filter a posteriori state estimates over the entire year 2019. The estimates follow the measurements quite closely, no extreme deviation visible.



(d) Kalman Filter a posteriori state estimates over the month January of the year 2019. Again only small deviations from the measurements. The deviations are noticeable in the locations where the change in model prediction from the previous time step to the current time step deviates sharply from the change in measurement value from the previous to the current time step.

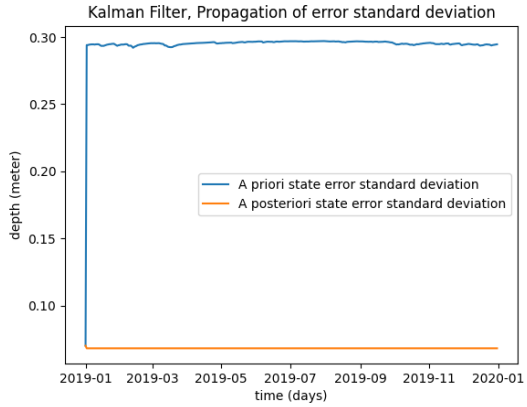


(e) Kalman Filter a priori state estimates with the associated a priori estimate error standard deviation over the month January of the year 2019.

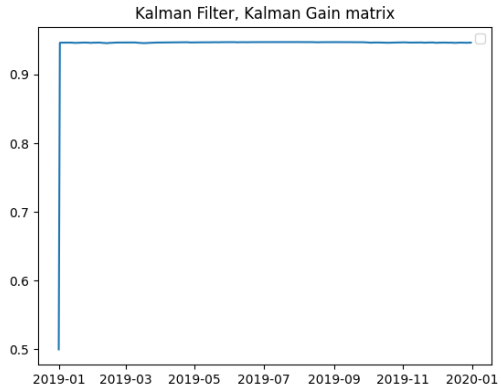


(f) Kalman Filter a posteriori state estimates with the associated a posteriori estimate error standard deviation over the month January of the year 2019.

Figure 5.6: Kalman Filter state estimates in location 9012. The a priori and a posteriori state estimates are viewed over 1 year, and 1 month. Also, the a priori and a posteriori state error covariance are viewed over 1 month. For comparison, the model predictions can be seen in green.



(a) Both the a priori and a posteriori state error covariance plotted for each time step.



(b) The Kalman Gain matrix for each time step of 2019.

Figure 5.7: Background information used in the Kalman Filter scheme: the propagation of the a priori and a posteriori state error standard deviation matrices, and the value of the Kalman Gain matrix. Since we have a one dimensional state, all these matrices simply are numbers.

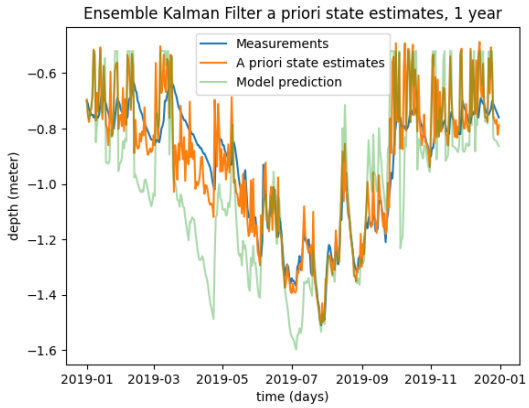
In each time step, the a priori state estimate uses the previous a posteriori state estimate to make a model prediction which is always very close to the measurements. As we have seen in the previous section, the model prediction deviates sharply from the measurements, the erratic behaviour seen in figure 5.6a and figure 5.6b comes from this deviation, to quantify this difference the RMSE was computed: $0.107m$. However, it is not always that the a priori estimate differs from the measurements, in July until September for example they follow the measurements quite closely. Therefore, the model behaves incorrectly to the input at certain days, but not all days.

The a posteriori state estimate is at each time step very close to the measurements. To have a measure of how close this is, the RMSE was computed: $0.00572m$, which is indeed way lower than the RMSE associated with the a priori state estimates. This closeness is to be expected, since the covariance associated with the measurements is far lower the covariance associated with the model. As seen in figure 5.6b, when the change in model value in the time step is very different from the change in measurement value in that time step, the a posteriori estimate deviates most from the estimate. This can also be interpreted as that if when the current a priori state estimate is far from the measurements, the current a posteriori estimate also deviates more from the measurements.

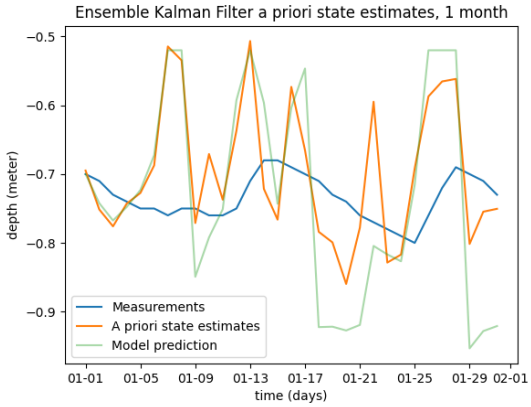
The a priori state error covariance seen in figure 5.6e is quite high in every step. This is because it is equal to the model error Σ^m with a part of the model which is multiplied by the previous a posteriori state error covariance, as described in chapter 4. Therefore, it is never lower than 0.0841, and fluctuates depending on the input. The large error in the estimates is therefore to be expected, and it is nice to see that for both the a priori and the a posteriori estimates the measurements are within the uncertainty range.

The Kalman Gain value for each time step can be seen in figure 5.7b, it is quite close to 1, and fluctuates little. The mean of this value is equal to 0.946. That this value is close to one means that the measurement gets trusted the most in each step when you fuse the a priori estimate and the measurement, so much so that indeed little of the model prediction is left after fusing.

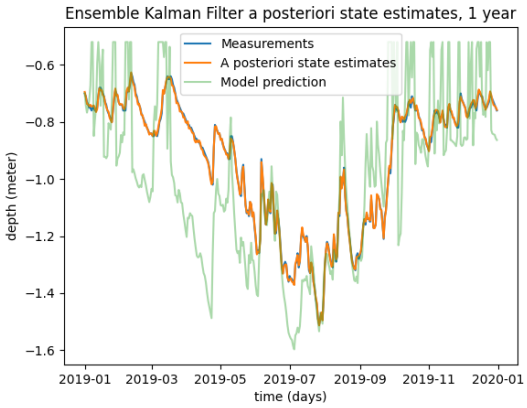
Here, the covariance of the measurements is far smaller than the covariance of the model. This results in an a posteriori estimate which is almost equal to the measurement. It is good that uncertainty can be quantified using this method but for state estimation the measurements by itself are sufficient. The Kalman Filter therefore is a little superfluous in the case when you have such a large gap in covariances.



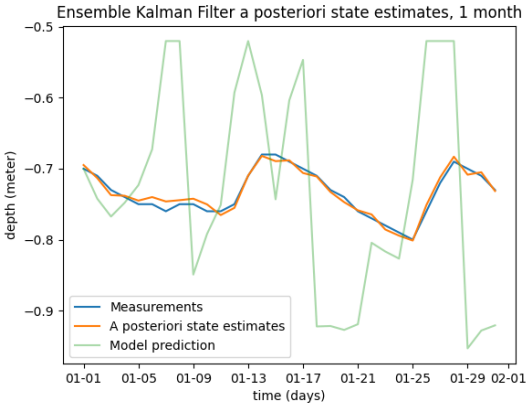
(a) Ensemble Kalman Filter a priori state estimates over the entire year 2019.



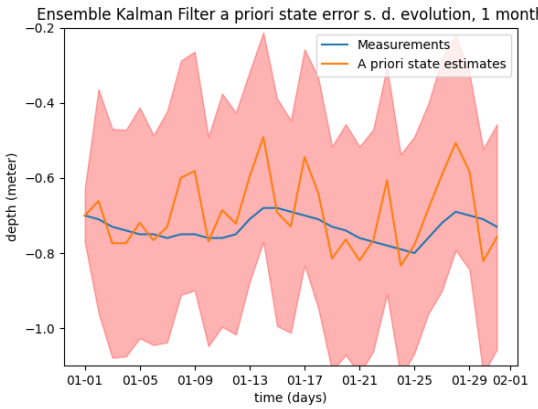
(b) Ensemble Kalman Filter a priori state estimates over the month January of the year 2019.



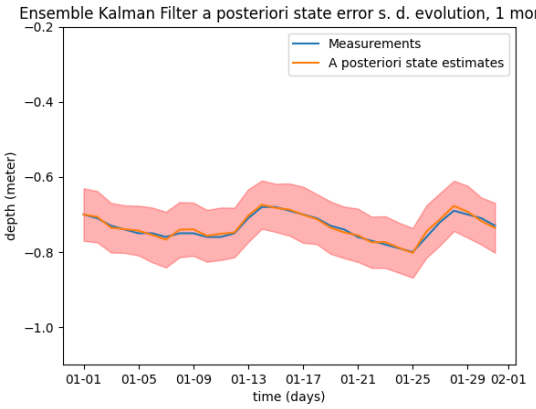
(c) Ensemble Kalman Filter a posteriori state estimates over the entire year 2019.



(d) Ensemble Kalman Filter a posteriori state estimates over the month January of the year 2019.

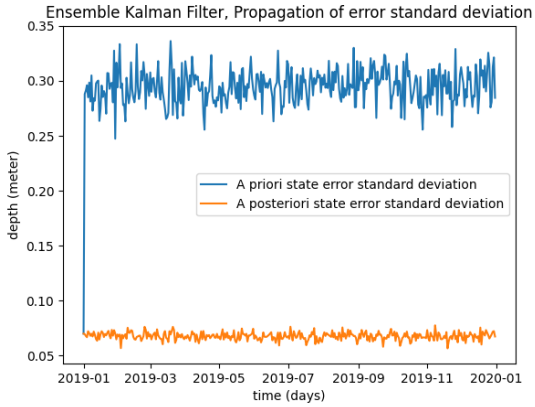


(e) Ensemble Kalman Filter a priori state estimates with the associated a priori estimate standard deviation over the month January of the year 2019.

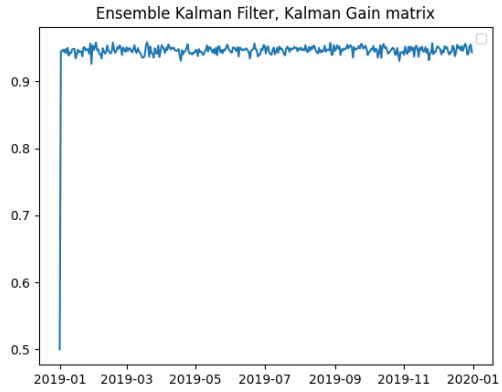


(f) Ensemble Kalman Filter a posteriori state estimates with the associated a posteriori estimate error standard deviation over the month January of the year 2019.

Figure 5.8: Ensemble Kalman Filter state estimates in location 9012. The a priori and a posteriori state estimates are viewed over 1 year, and 1 month. Also, the a priori and a posteriori state error covariance are viewed over 1 month.



(a) Both the a priori and a posteriori state error covariance estimates plotted for each time step.



(b) The approximated Kalman Gain matrix for each time step of 2019.

Figure 5.9: Background information used in the Ensemble Kalman Filter scheme: the propagation of the a priori and a posteriori state error standard deviation matrices, and the value of the Kalman Gain matrix. Since we have a one dimensional state, all these matrices simply are numbers.

When comparing the Kalman Filter and the Ensemble Kalman Filter, the first thing that is noticed is that the differences are not that big. In figure 5.6a and figure 5.8a it can be seen that there are a few differences, some peaks lower, some peaks higher, but nothing that would make the approximation unusable. On the monthly pictures in figure 5.6b and figure 5.8b those differences can be viewed for a single month. Here also, there are a few points where the EnKF estimates differ from the KF estimates, but nothing unreasonable. All differences arise from the fact that the Kalman Filter is the optimal solution, in the sense that it produces the most accurate prediction with the exact covariance. The Ensemble Kalman Filter is an approximation of this. The goal was to see if the approximation is good enough to suit our further purposes, these figures suggest that they are.

All of the figures are extremely similar, except for the final two. Figure 5.7a and figure 5.9a contain the error covariance matrix of the Kalman Filter and the Ensemble Kalman Filter respectively. What is interesting is that the second image shows way more fluctuation that the first. The mean is the same, in both images the a priori state error covariance lies around the 0.8, but in the first image it is very stable and in the second image it fluctuates a lot. This can be explained by noting that in the first case, the Covariance is computed exactly by Σ^m plus something small. In the second case, Σ^m is approximated, so it can be all sorts of values around Σ^m. Since the sample size is quite large the approximation is pretty good, but not as clean as in the Kalman Filter. But, when you take the scale of the y axis into account, you can see that the values are always small compared to the model and measurements and the fluctuations don't affect the state predictions too much, but just enough to sometimes trust the model a bit more compared to previously which contributes to the slight fluctuations. The more noisy Kalman Gain matrix from figure 5.9b compared to figure 5.7b is due to the same phenomenon. Here it is clear that the noise doesn't really affect the results, since it is always above 0.9 still the measurements simply are trusted way more than the model.

To really convince ourselves that the difference between the Kalman Filter and the Ensemble Kalman Filter are small, we compute the RMSE between the a priori state estimates, the a posteriori estimates, the a priori state error covariance and the a posteriori state error covariance:

a priori state estimate	$0.0244m$
a posteriori state estimate	$0.00489m$
a priori state error standard deviation	$0.0146m$
a posteriori state error standard deviation	$0.00344m$

Table 5.2: Root Mean Square Error of the Kalman Filter and the Ensemble Kalman Filter of all the estimates and covariances.

This confirms what we already suspected based on the figures; in this case the difference between the

Kalman Filter and the Ensemble Kalman Filter is negligible, we can work with the Ensemble Kalman Filter without having to worry that due to the fact that it is an approximation it produces incorrect state predictions.

But as noted before, the Ensemble Kalman Filter is now simply quite a crude tool for shifting the model to the measurements. In this case when the covariances are so far apart the measurements are dominant and trusted almost completely. However, the parameter estimation is more interesting and what we will investigate now.

5.3. Estimating Parameters

Now we arrive at what we wanted to do all along, the estimation of the different parameters present in the groundwater model. We want to find optimal parameters such that the model best fits the data, when this happens, the parameters are said to be calibrated on the data. We will do this is by using all the available data and use the Ensemble Kalman Filter to find the parameters as described in chapter 4 to try to find the parameters. Then, we use the means of the parameters in the ensemble that are found in the final time step and evaluate the model again using these new parameters. The idea is that now, the parameters are better than what they initially were. We will evaluate how much better these new parameters are by comparing the RMSE between the model and the measurements, and look at the different model trajectories in plots.

We will do this in two steps. First, we will create artificial data ourselves and see if the parameter calibration scheme works at all. We will test the calibration of multiple different combinations of parameters, and try to discover when parameter calibration is, and is not possible. Then, having knowledge of when parameter calibration is fruitful and realistic, we will calibrate the parameters of multiple data sets and look at the quality of the results.

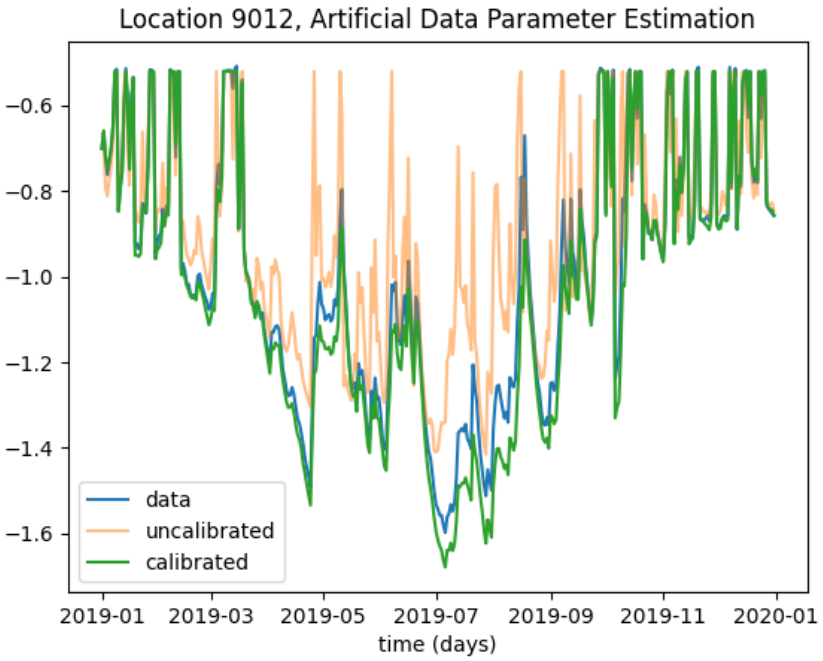
5.3.1. Calibration of Artificial Data

To begin to see if estimating parameter is at all possible we will first try to estimate the parameters from artificial data. From this, we hope to be able to answer the following questions;
1) Which parameters are suitable for calibration.
2) How much can we rely on the parameter found, is there a way to say with certainty that it corresponds with the physical parameters present in this location.
What we do is the following, using the model with the parameter as given by the input data, we generate the prediction over the year 2019 with the random noise associated with the measurements, $N(0, \Sigma_y)$, added at each prediction. This error is chosen since we want the calibrated model to fit the measurements and be able to correct for their associated error. Then, we will shift the initial value of one or more parameters. After this, we let our the augmented Ensemble Kalman Filter run, with the uncertainties equal to what they were before so to the values in table 5.1. We expect that the parameter shift back towards in direction of their initial value and stabilize there, with a small fluctuations as a consequence of the random error associated with the measurements, since it is certain that moving towards the initial value of the parameter fits the artificial data better.
But, how do you shift the parameters? We do this by placing a weight in front of each parameter that we wish to calibrate. First, The artificial data is obtained as described above. Then, the starting weights are shifted to a new value, say 0.5, so they halve the corresponding parameter. Now, the Ensemble Kalman Filter is run to find the optimal parameters.
There is a problem with this setup however, in taking the initial ensemble it could be that some ensemble members have initial values that are negative, or values close to 0. The first case indicates a negative resistance, which is not possible, and the second results in a division by 0 error. To circumvent this, a transformation of the parameter space is applied that the weights can only assume positive values. The initial value of all the weights is 0, and when using it in the model all parameters are multiplied by e^{weight}. Now, $e^0 = 1$ so the variable stays the same, negative parameters result in a number between 0 and 1, and positive parameters result in a positive value. In this case, the parameters are always positive, which is what we want, and a negative resistance or storage is circumvented. The equation used is still equation 2.4, but now the parameters s,

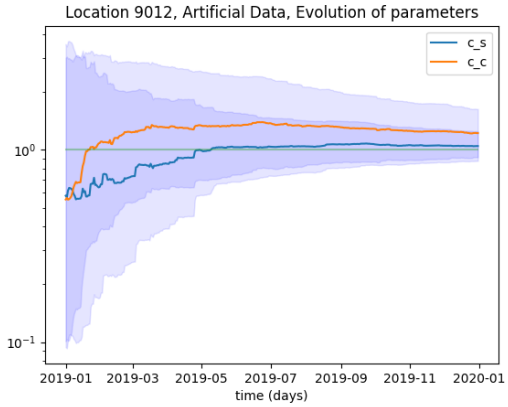
c_d and c_p used in the function are obtained by first multiplying them by their weight:

$$\begin{aligned} s &= s * exp(c_{s_{exp}}) \\ c_d &= c_d * exp(c_{cd_{exp}}) * exp(c_{c_{exp}}) \\ c_p &= c_p * exp(c_{cp_{exp}}) * exp(c_{c_{exp}}) \end{aligned} \tag{5.1}$$

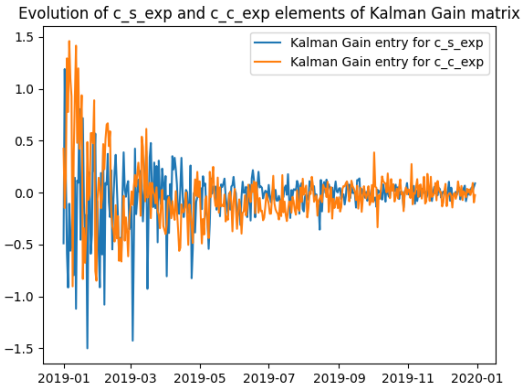
The results of parameter calibration is first computed for the weight $c_{s_{exp}}$, the storage parameter, and $c_{c_{exp}}$ a weight for all resistance terms, so for both c_p and c_d. The two weights before the resistance parameters allows for simultaneous calibration of both resistances, or each resistance separately. When the weights are not being calibrated they have a value of zero so they are equal to one when taking their exponent and they don't influence the results. When running the scheme, figures such as figure 5.10 can be plotted, this figure is obtained for calibrating the storage parameter c_s and starting with a value of 0.5.



(a) Artificial Data Parameter Estimation, the uncalibrated model has value 0.5 for both c_s and c_c, the calibrated obtains c_s of 0.990 and c_c of 1.22. The RMSE of the uncalibrated model is $0.166m$ and the RMSE of the model with calibrated constants is $0.0621m$.



(b) The parameters c_s and c_c over time, in dark blue the ensemble means, in lighter blue the ensemble standard deviation. This is the ensemble mean and spread after taking the exponent of each element. Note the logarithmic scale, since we compute the logarithm of a normally distributed value, the result is viewed as normally distributed on the logarithmic scale



(c) Kalman Gain Matrix of the parameters c_{sexp} and c_{cexp}. c_{cexp} indicates the parameter before translation.

Figure 5.10: Results of parameter calibration with starting values of 0.5 on artificial data. The model was run with the input from location 9012. The initial ensemble for the weights were samples from $N(0, 1.7)$.

From figure 5.10a it is immediately clear that the parameter calibration works, it indeed has a very positive effect on the function fit. The uncalibrated model deviates from the data quite often, whilst the calibrated model is always a lot closer to the data. There are however some points where the value of the calibrated model is a little lower than the data.

From figure 5.10b we can see that the value of c_s becomes closer to 1 than the value of c_c, but whether this is due to random fluctuations or really a difference in the parameters this result is not enough. Also, the uncertainty around c_s is smaller than the uncertainty around c_c. We can see that the uncertainty steadily decreases as time goes on, even after a year the uncertainties are still shrinking, but the parameters do become close to their final value quite early on.

Now to investigate the results of parameter estimation more rigorously, we calculate the final prediction for the parameter 10 times, and take the mean. Also, to evaluate how much better the fit becomes after calibrating a parameter, we compare the Root Mean Square Error of the uncalibrated parameter model and the artificial data with the RMSE of the model with calibrated parameter and the artificial data.

	0.5	0.8	1.25	2.0
c_s	1.002	1.042	1.032	1.035
c_c	1.018	1.226	1.212	1.250
c_s&c_c	1.002 & 1.023	1.024 & 1.216	1.008 & 1.143	1.032 & 1.311

Table 5.3: The rows indicate which parameter(s) is/are being calibrated, the columns which initial value for the parameter(s) is/are chosen. The measurements used to calibrate the model on is the model with parameters all equal to 1.

for the starting values of 1.25 and 0.8 the RMSE of the uncalibrated model has a value around $0.0632m$, whilst for a starting value of 0.5 and 2.0, the RMSE initially is around $0.141m$.

	0.5	0.8	1.25	2.0
c_s	8.58	2.01	2.32	4.67
c_c	1.32	1.19	1.21	3.51
c_s&c_c	4.15	1.47	4.02	2.50

Table 5.4: The rows indicate which parameter(s) is/are being calibrated, the columns which initial value for the parameter(s) is/are chosen. Here, the RMSE of the uncalibrated model with the data is compared with the RMSE of the calibrated model with the data by taking their ratio, $RMSE_{uncalibrated}/RMSE_{calibrated}$. For the starting values 1.25 and 0.8 the RMSE of the uncalibrated model and the data has a value around $0.0632m$, whilst for the starting values 0.5 and 2.0, the RMSE initially is around $0.141m$.

Both parameters usually become closer to one. This indicates that at the very least, the calibration is usually doing something positive. Moreover, after calibration c_s is consistently closer to 1 than c_c. This indicates that c_s has a bigger impact on the results, because the model changes from day to day more because of the storage than because of the resistance terms. A resulting value which is further away from 1 indicates that having a value closer to 1 does not yield results sufficiently better to overcome the randomness in the system. If the value in table 5.4 is greater than one, the model with calibrated parameters is closer to the measurements than the model with uncalibrated parameters. All entries in the table are greater than one, so the calibrated function for all these cases becomes better than the uncalibrated function. However, one thing is happening which is a little strange.
The value of the calibrated parameter is for every entry in the table greater than 1. This is strange since 1 is the perfect parameter, and values around this both result in good values, then it would be expected that sometimes a value a little too low, and sometimes a little to high should be found. The overshoot is most clear when considering the parameter c_c but occurs everywhere. Why this is is investigated further in the discussion in chapter 6.

5.3.2. Calibration of multiple parameters

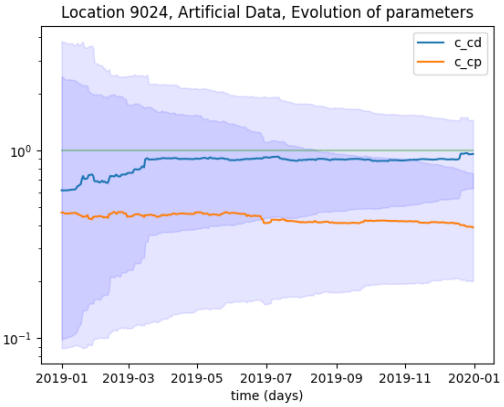
When calibrating multiple parameters the question can arise if you are calibrating to just improve the model results or if the results correspond with the physics in your location. For example: suppose there is a mathematical model $h = a + b$, for the height of a building, with a the first floor and b the second floor. We have have (accurate) measurements of the building and we use the EnKF scheme to estimate the state and the parameters a, b. The starting parameter ensembles are chosen, and due to random noise the initial ensembles have a wide range of parameters. Since the measurements are reliable, the parameters are updates each time by a large value until the prediction of the function comes close to the measured value. Now, parameters a, b are calibrated such that their sum is equal to h. However, what is important to notice here is that their are infinitely many pairs a, b that fit this description, such that your final ensemble has a very high spread for the parameters but a low spread for h. From this, you can of course gain no information about the values of the parameters a, b. Here, too many parameters to estimate are chosen.

Such a situation can be recognized by looking at the covariance of the final ensemble. In such a situation, the covariance of the state is small whilst the variance of the parameters are large.

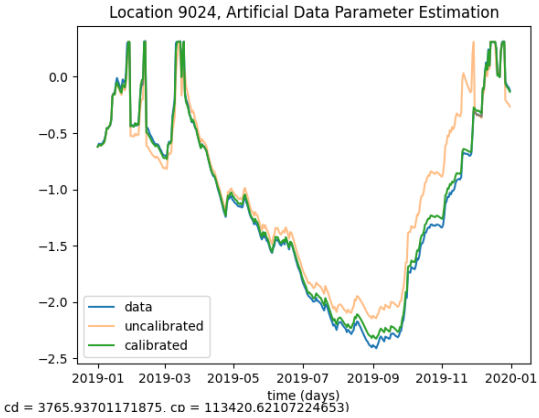
For this to be achieved it is important that the process is started by starting with a large initial spread of the parameters, if the initial spread is not large enough, the variance associated with the parameter ensemble is small and the parameters are only changed by very small increments each step. This causes the parameter to remain nearly constants. From a larger initial spread, the covariance associated with the parameter ensemble is big and if the model deviates a lot from the data the parameter is changed a lot as a result. Now, the spread becomes steadily lower and the parameters can all converge towards a single value which can be said to be the 'true' parameters. If not, and the parameter spread remains big, than the parameters are interacting or they don't impact the model sufficiently to cause the model prediction to change enough, and you can not speak of 'true' parameters. Through trial and error it was found that an initial ensemble of the weights as samples of $N(0, 1.7)$ was sufficient to calibrate the parameters.

At first, all the resistances are jointly calibrated by a single parameter c_c. It would be better to calibrate these both with a separate factor, so a factor c_{c_d} for c_d and a factor c_{c_p} for c_p. Now, we will investigate if this is possible. This could be viewed as unrealistic if you notice that they have identical terms in the groundwater evolution model. But, the difference between a summer and winter polder level could be just enough information to indeed calibrate both parameters. For the first period of time, so from the start of the year until the summer level is set, both resistances are almost constant at different values. Here, the calibration of the resistance results in that the parameters all attain values such that the sum of both resistance terms have the same model output, which is then optimal. This situation can be compared to the building example above when a, b are calibrated. Here, all of the ensemble members have elements a, b such that their sum is equal to h. In our case, all ensemble members could have values for c_{c_p}, c_{c_d} such that they result in a correct value for h. In this scenario, the variance association with both terms is expected to be very high. But then in the spring, the polder level changes, which then results in a different value for the parameter c_{c_p} which have a perfect solution. Now, these two pieces of information for the resistance terms could be enough to calibrate both c_{c_d} and c_{c_d}. Therefore, it could be possible that unique and certain parameters for both the polder and deep resistance are found.

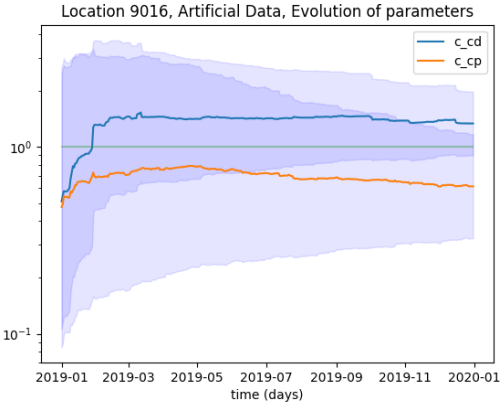
To this end, we compute the results of 3 with the artificial data, their starting values are both 0.5 and all other parameters are equal to zero. These results can be seen in figure 5.11



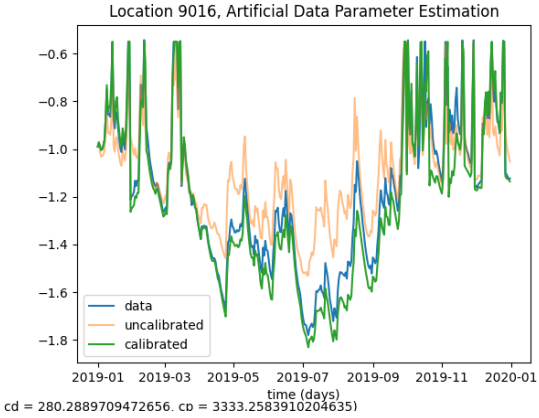
(a) Area 8907, the resistance parameters evolution with the associated uncertainty. The mean of the parameters finally become: $c_{c_d} = 0.957$ and $c_{c_p} = 0.389$.



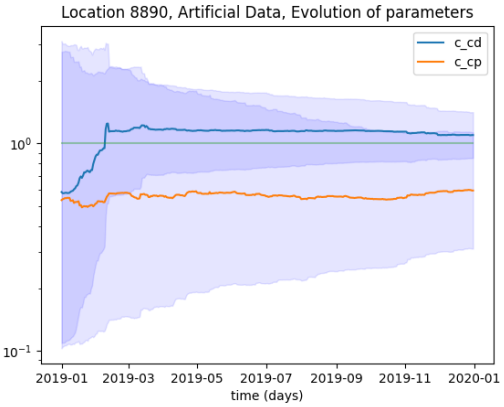
(b) Area 8907, the data with the measurement noise, the model with the shifted parameters, and the model with calibrated parameters. $RMSE_{uncalibrated}/RMSE_{calibrated} = 31.4$.



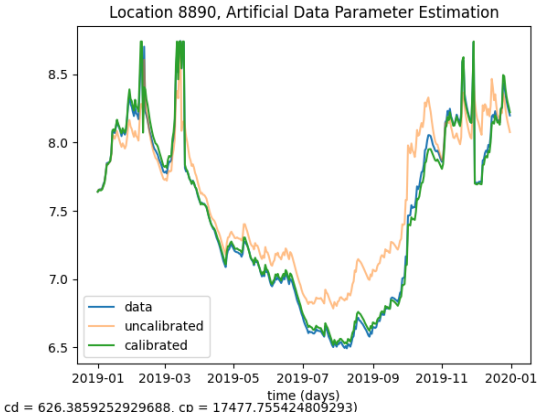
(c) Area 9016, the resistance parameters evolution with the associated uncertainty. The mean of the parameters finally become: $c_{c_d} = 1.33$ and $c_{c_p} = 0.614$.



(d) Area 9016, the data with the measurement noise, the model with the shifted parameters, and the model with calibrated parameters. $RMSE_{uncalibrated}/RMSE_{calibrated} = 3.29$.



(e) Area 8890, the resistance parameters evolution with the associated uncertainty. The mean of the parameters finally become: $c_{c_d} = 1.10$ and $c_{c_p} = 0.59$.



(f) Area 8890, the data with the measurement noise, the model with the shifted parameters, and the model with calibrated parameters. $RMSE_{uncalibrated}/RMSE_{calibrated} = 29.7$

Figure 5.11: Calibration of both resistance parameters for the year 2019 for area 9024, 9016 and 8890.

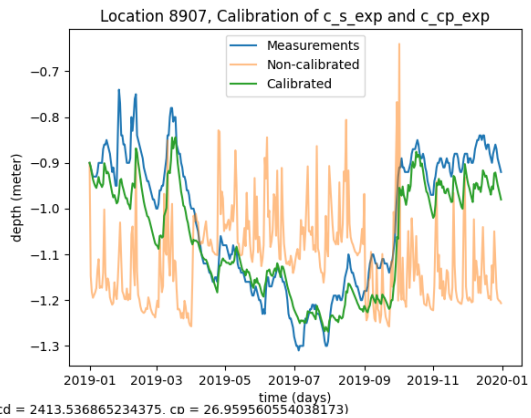
First, when you only replace the resistance terms, the calibrated model still becomes a lot better compared to the uncalibrated model. This is supported by looking at figures 5.11b, 5.11d and 5.11f, and looking at the ratio between the RMSE of the uncalibrated data and the measurements with the RMSE of the calibrated data and the measurements which is always larger than 1.

In figures 5.11a, 5.11c, 5.11e, we can see that the weight c_{c_d} becomes closer to one and ends with a smaller standard deviation that c_{c_p}. Looking at all the associated resistance terms, we note that c_d is in all these cases far lower than c_p. This has the effect that changing c_{c_p} has a greater effect on the predictions and therefore this parameter is calibrated more. Indeed, it seems to be the case that if a parameter plays a smaller role in the model, the associated weight gets calibrated less. Also, from figures 5.11c and 5.11e the hypothesis that parameters interact such that together they produce the correct is made more probable. The mean of c_{c_d} is slightly above one and the mean of c_{c_p} is lower than one such that together the resistance terms serve to give the correct prediction.

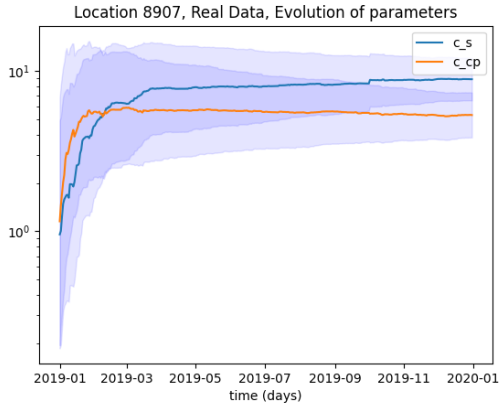
From these observations we conclude that it is not good to calibrate each resistance weight separately. Since the value of one of the two resistance parameters is a lot smaller than the other (now c_{c_d} is always smaller but in other data c_{c_p} was smaller) we think it is sufficient to simply calibrate this weight and let the other weight remain one. This way, interaction between the resistance parameters is avoided and

5.3.3. Calibration of model parameters on real data

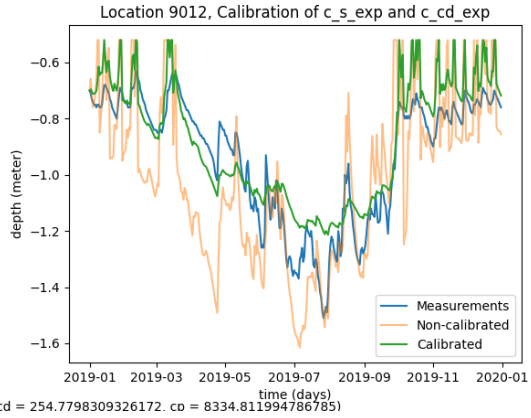
In this section, we will finally calibrate the model on the actual data. We have seen that it is realistic to expect a meaningful calibration when considering one resistance parameter (the smallest), and the storage parameter. Therefore, we will use this information to calibrate the model on the data. With all this information, we have developed the following parameter calibration scheme:
1) Investigate the magnitude of the different resistances. If one is way smaller it is reasonable to say calibration of this parameter is sufficient to act calibrate the resistance, calibration of the other would not create a much better result and introduces unnecessary freedom.
2) Calibrate the chosen resistance parameter along with the storage parameter.



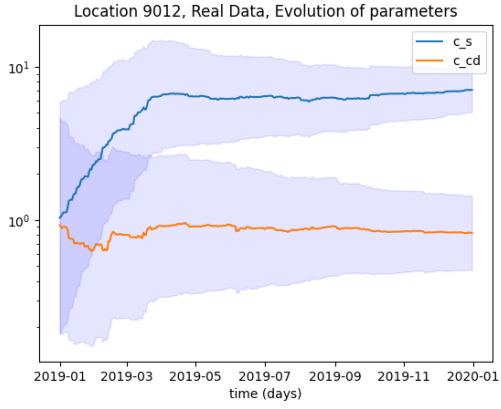
(a) The measurements plotted along with the model with uncalibrated and calibrated parameters. The parameters are calibrated over all of 2019. The location is 8907. $RMSE_{uncalibrated} = 0.217m, RMSE_{calibrated} = 0.0580m$.



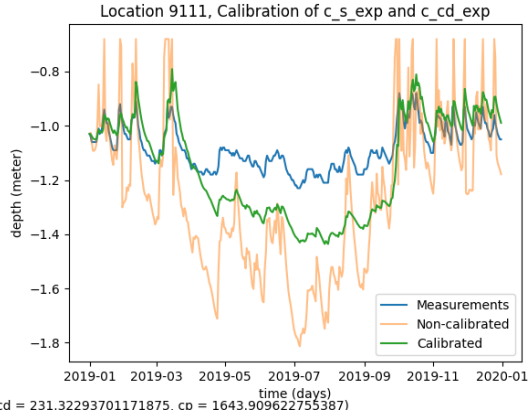
(b) The evolution of the parameters, the mean of the parameter ensembles along with the standard deviation. The location is 8907. The mean of the parameters finally become: $c_s = 8.88$ and $c_{c_p} = 5.31$.



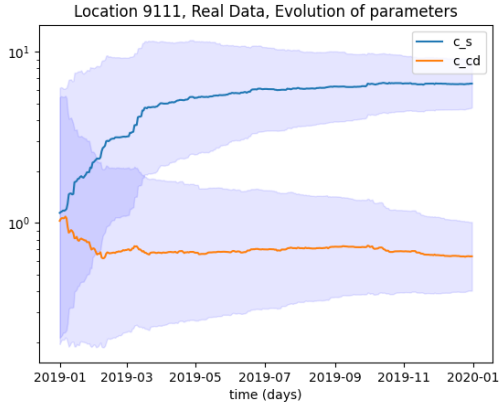
(c) The measurements plotted along with the model with uncalibrated and calibrated parameters. The parameters are calibrated over all of 2019. The location is 9012. $RMSE_{uncalibrated} = 0.196m, RMSE_{calibrated} = 0.110m$.



(d) The evolution of the parameters, the mean of the parameter ensembles along with the standard deviation. The location is 9012. The mean of the parameters finally become: $c_s = 6.37$ and $c_{c_p} = 0.768$.



(e) The measurements plotted along with the model with uncalibrated and calibrated parameters. The parameters are calibrated over all of 2019. The location is 9024. $RMSE_{uncalibrated} = 0.275m, RMSE_{calibrated} = 0.192m$.



(f) The evolution of the parameters, the mean of the parameter ensembles along with the standard deviation. The location is 9024. The mean of the parameters finally become: $c_s = 6.52$ and $c_{c_d} = 0.638$.

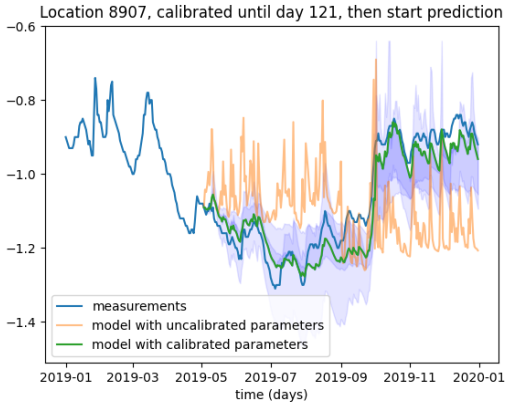
Figure 5.12: Calibration of the storage parameter and the smallest resistance parameter on real data for the period 2019 and locations 8907, 9012 and 9111.

For each location, we see that the model becomes way closer to the calibration after calibration. We do still see errors, but important is that the model behaviour starts to look more like the measurement behaviour. Locations 9012 and 8907 especially are very good. Here the model prediction is as good as can be expected from such a simple model. In the other case, there still is a large deviations throughout the year. The calibra-

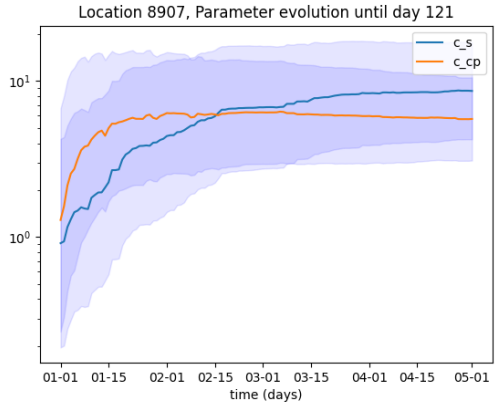
tion of the parameters can be seen on the right. Most noticeably, the parameter c_s for the storage is calibrated fast and accurately to a single value. This can be explained by noting that the storage plays a very important role in the model. The parameter for the resistance remains more uncertain, this is also to be expected since it is of less importance. What is clear from figure 5.12b is that when the value of the resistance is low, here $c_p = 27.0$, calibration of the associated weight leads to far more changes in that weight. In the other cases, the weight remained close to one but here it increases a lot to 5.31.

5.4. Prediction with the calibrated parameters

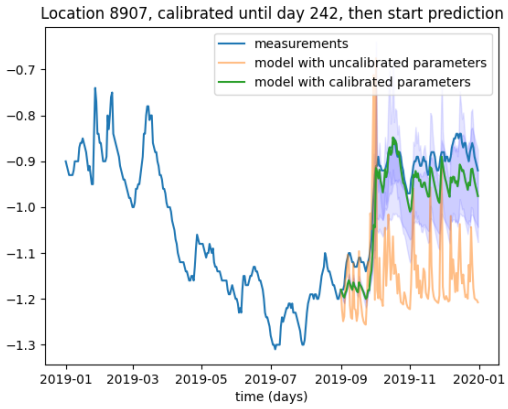
The prediction will be done by using a certain piece of the data for parameter calibration, and another piece for evaluating how good these parameters become. With this we will try to get a grip on how good the prediction of the model is, and how much data is necessary to improve the prediction, this final point can be realised by looking at the certainty associated with the model, large uncertainty of parameters indicate that perhaps not yet enough data is accumulated to gain certainty on the parameters.



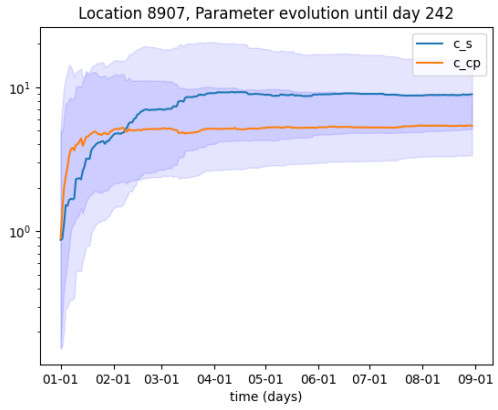
(a) The measurements plotted along with the model with uncalibrated and calibrated parameters. Here, the parameters are calibrated until day 121. The location is 8907. $RMSE_{uncalibrated} = 0.202m, RMSE_{calibrated} = 0.0528m$. In lighter blue the model with the parameters and their standard deviation is plotted.



(b) The evolution of the parameters until day 121, the mean of the parameter ensembles along with the standard deviation. The location is 8907. The mean of the parameters finally become: $c_s = 8.65$ and $c_{cp} = 5.72$.



(c) The measurements plotted along with the model with uncalibrated and calibrated parameters. Here, the parameters are calibrated until day 121. The location is 8907. $RMSE_{uncalibrated} = 0.236, RMSE_{calibrated} = 0.0560m$. In lighter blue the model with the parameters and their standard deviation is plotted.

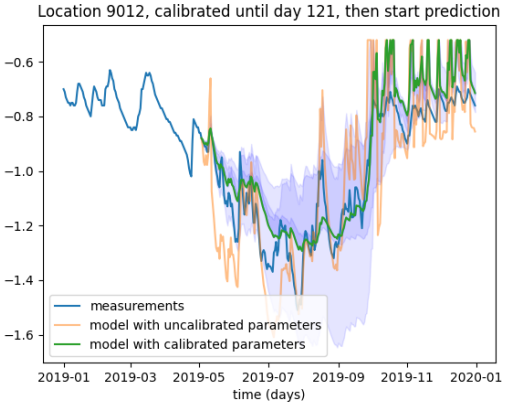


(d) The evolution of the parameters until day 121, the mean of the parameter ensembles along with the standard deviation. The location is 8907. The mean of the parameters finally become: $c_s = 8.93$ and $c_{cp} = 5.40$.

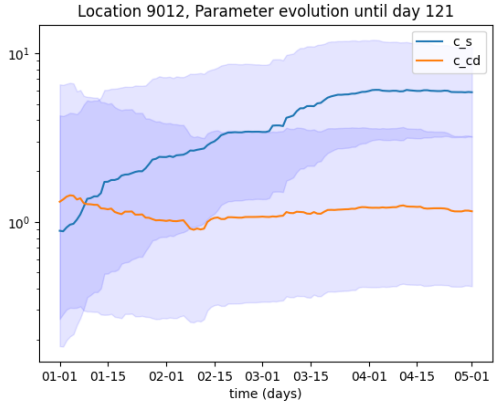
Figure 5.13: For location 8907, the calibration in certain period followed by the prediction. The prediction is given by the mean parameter, with in lighter blue the parameter with standard deviation.

Both fits perform quite well, in figure 5.13a we see is that even after the short 121 days, the weights found already perform well in the prediction. This does not improve that much compared to 5.13c, the RMSE ratio goes up but this is of course also because in the image it can be seen that the model performs really bad in

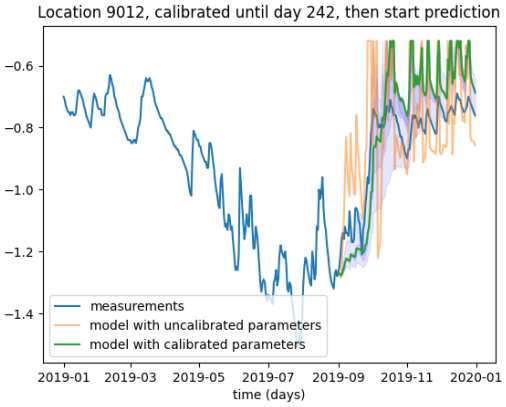
the final part of the data. In figures 5.13b and 5.13d the trajectories of both weights can be seen until 121 and 242 respectively. In both figures it is clear that the weight changes a lot in the initial period, the first month the most, the first 3 months quite a bit and then more or less stable. The standard deviation becomes smaller over time, but the mean does not change much.



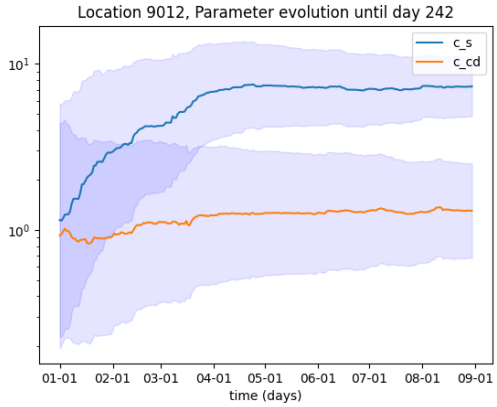
(a) The measurements plotted along with the model with uncalibrated and calibrated parameters. Here, the parameters are calibrated until day 121. The location is 9012. $RMSE_{uncalibrated} = 0.170m, RMSE_{calibrated} = 0.124m$. In lighter blue the model with the parameters and their standard deviation is plotted.



(b) The evolution of the parameters until day 121, the mean of the parameter ensembles along with the standard deviation. The location is 9012. The mean of the parameters finally become: $c_s = 5.86$ and $c_{c_d} = 1.15$.



(c) The measurements plotted along with the model with uncalibrated and calibrated parameters. Here, the parameters are calibrated until day 242. The location is 9012. $RMSE_{uncalibrated} = 0.168m, RMSE_{calibrated} = 0.0990m$. In lighter blue the model with the parameters and their standard deviation is plotted.



(d) The evolution of the parameters until day 242, the mean of the parameter ensembles along with the standard deviation. The location is 9012. The mean of the parameters finally become: $c_s = 7.31$ and $c_{c_p} = 1.31$.

Figure 5.14: For location 9012, the calibration in certain period followed by the prediction. The prediction is given by the mean parameter, with in lighter blue the parameter with standard deviation.

From this case it is more clear that the standard deviation really shrinks over time, so more measurements are useful in the sense that you gain more confidence in the parameter.

6

Conclusion and Discussion

The goal of this thesis was to calibrate the parameters in a groundwater model with the Ensemble Kalman Filter to make accurate predictions about the future groundwater level. All chapters lead up to this final goal. For each chapter, we will discuss the conclusions obtain there and their validity. We will attempt to do this chronologically, however when the results in chapter 5 says something about the chapter under discussion, we will discuss these simultaneously.

In chapter 2 the groundwater model that we used was developed. Here, precipitation, evaporation, fluxes with a deep aquifer and fluxes with the open structures in the polder were taken into account. One thing that was ignored for now is a certain portion of the ground where the water is not yet saturated but a percentage of the soil is: the soil moisture zone. Precipitation must first pass this zone and evaporation also happens mostly from this zone, so a "complete" model takes the link between and the different physical characteristics of the soil moisture zone and the groundwater zone into account.
Also, the assumption that the input stays constant over the interval needs to be validated. A few factors are easy, the resistances are expected to be constant over all time, not even only the interval. The surface level changes to times a year from a winter level to a summer level and back and is regulated closely by the water board. Also, the deep head changes little throughout the year and after dividing by the resistance so little change is left that taking constant over an interval is not unrealistic. The precipitation is not constant over time, even during a single day it can still fluctuate a lot. However, this data is obtained as the precipitation sum over the entire day, so it already given as a single value and therefore taking a constant value over the entire day is the best that can be done. But, since the groundwater table itself reacts slowly to changes, the storage also changes slowly and taking a constant would not be expected to be too bad. However as seen in chapter 5, the analytical solution does indeed contain numerical errors. It is lower than the Euler solution but still not perfect. This can be circumvented in 2 ways: You could choose to use smaller time steps also in the code, or transform try to find an exact solution for the integral with N not constant.

However, even with these potential limitations, the groundwater model resulted in a workable model of the groundwater. It is very promising that using the input, the model was able follow the same trajectories as the measurements. The model could be solved and allowed for the creation of predictions, which is what we wanted.

In chapter 3 the data the model needed was collected, along with a derivation of the uncertainties associated with each data type. The data could easily be imported from the platform provided by Acacia, but the derivations we had to derive ourselves, and they are not perfect. This is mostly due to imperfect information, and with this present more accurate estimates and a more certain bound on the results can be given. Better ways to derive uncertainties:
For the measurement uncertainty 2 relatively close data points are used, however they are not in exactly the same location so the error derived from here can also simply be a result of the distance between the points. A better way would be to calculate for each device the error by observing it's behaviour first in a laboratory where the exact values for the groundwater level are known. Also, now for both the measurement and the model uncertainty, a way is described to find the maximum deviation and take this to be the standard devi-

ation. This is a little strange, you can also find the mean of how much the 2 different series differ, this can then be more intuitively seen to be the standard deviation. For the precipitation and evaporation the KNMI could be contacted, they probably have uncertainties available. Same goes with the deep groundwater, this now comes from the MIPWA [5] model, a model contains uncertainties as we now well know, and presumably they have a way of quantifying theirs. The storage function is quite complex and in this chapter already it was stated that the function looks like a straight line when considering a reasonable groundwater range. Since the constants in the storage function are also just calibrated from data which contains errors etc. it is reasonable to suggest that you replace this entire function simply by a collection of straight lines.

In chapter 4 the tools used from Data Assimilation are described. These are the regular Kalman Filter, the Ensemble Kalman Filter and the augmented Ensemble Kalman Filter with parameters in the state vector. The choice for the Filters, which together are called sequential data assimilation methods, also needs to be justified. There are namely also Variational methods such as 4D-var that can be used for the same goals as sequential methods and are to better suited in cases when large amounts of measurements are available [19]. However, here the choice was made for sequential data assimilation since the author was interested in the uncertainty estimates. This is because methods of estimating uncertainty associated with a prediction is not used often in hydrological modelling, when discussing with Acacia is has become clear that usually the prediction is "perfect", nobody knows how certain the prediction is and it is trusted completely. This method, whilst being able to generate a prediction, has the potential to also give a measure of how certain the prediction is.

In chapter 5 first the different solutions found for the model were compared. We concluded that whilst the semi-analytical solution was better than the Euler solution, both did contain numerical errors. Since the model input was given in a single day and the numerical errors was not so big as to cause major issues, we decided to continue with the semi-analytical model with time steps of a single day.
Then, the Kalman Filter was compared with the Ensemble Kalman Filter to validate that the states remained similar. It was found that it was, for an ensemble of size 200 the means in the Ensemble Kalman Filter differ little from the estimates in the Kalman Filter. However, this is not very rigorous. Firstly, it is close for an ensemble of size 200, but this number is chosen arbitrarily, interesting would be to see which number of ensemble members is necessary to still obtain sufficiently correct predictions. Secondly, that the state prediction is stable is to be expected, always the measurements are trusted way more than the predictions, so always the model predictions are shifted to the measurements, for each ensemble member. So it is expected that this performs well for most ensemble sizes. What is more interesting is when the augmented Ensemble Kalman Filter has a large enough ensemble size. The a posteriori predictions were always far closer to the measurements than the associated standard deviation would indicate. This is a sign that the uncertainty associated with the model was too high, since lowering the largest uncertainty decreases the standard deviation the most.
Now, the Ensemble Kalman Filter with augmented state to include parameters is used to calibrate the model on artificial data where the parameters to find are known. Immediately a problem comes clear, what the spread of the parameter ensemble? When considering the state ensemble the measurement uncertainty was a logical choice, here no such choice is available. Now, playing around with the code and viewing the results indicated that an ensemble with spread $\Sigma(0, 1.7)$ was viewed to give good results, but this choice is in no way justified. Since in the parameter calibration we also note the importance of the spread of the initial ensemble, it would be better to investigate this choice more rigorously. The parameter calibration however performed well in all cases when it was used. All parameters became closer to their original value and the resulting model performed better as was expected, which is seen in the ratios of Root Mean Square Errors, these are all larger than one indicating the the calibrated model was always closer to the measurements than the uncalibrated model. This gives us confidence in the augmented Ensemble Kalman Filter and allows us to extend this to the real measurements. One thing that was noticed however is that all parameters found were a little higher than one, but we knew that one was perfect. This could be a result of the translation. Before we transition to the real data however we are interesting if the model allows for calibration of multiple resistance parameters. We found that usually one parameter was calibrated a lot whilst the other remained mostly constant to it's starting value. We concluded that this was a result of the difference in magnitude between the resistance parameters, and further in the research the weight associated with the smallest resistance parameter was calibrated, the other weight was kept at one.
Now the calibration of the storage weight and the weight of the smallest resistance on real data was per-

formed. In all cases, the model performed drastically better with calibrated parameters. The Root Mean Square Error always became lower, and usually became close $0.1m$. Interestingly, the storage always increased a lot, indicating the storage function structurally underestimates the storage. From the parameter propagation, we learn the quite fast the parameters attain correct values which change little from there on, and the rest of the data then serves to decrease the standard deviation. This could be a result of a too large initial spread leading to large uncertainty which decreases slowly. That the parameters attain values leading to a good model soon is promising, this would seem that even with 2 or 3 months of data the parameter become calibrated enough to serve in the prediction.
When testing the prediction this hypothesis was supported, with the data of $1/3$th year we could predict the future of the year quite well, not noticeably better than with data from $2/3$th year, the Root Mean Square Error did decrease with more data but not drastically. Always the prediction of the calibrated model had a lower Root Mean Square Error, and for the two locations tested their value became around $0.1m$. Therefore being able to predict the groundwater level with a standard deviation of $0.1m$ can be done with this model. This information is sufficient for farmers to implement better water management.

6.1. Further research

In this thesis the focus placed on parameter estimation and state prediction in a single location. However, the tools developed here can also be extended into a grid. The framework for this would be something like this: consider an n x n grid where there are measurements in m locations. The state vector $\boldsymbol{x}$ contains now each pixel and the model parameters which are assumed to be the same across the entire grid. In each pixel of the grid the mathematical model for the groundwater evolution holds, with the input also known. Predict the next state using the model, then transform the prediction to compare with the measurements and fuse then with the Kalman Gain. Now, we have an estimation for the state of the entire grid, and the parameters are also calibrated as before, allowing for predictions along the entire grid.

Also, as mentioned above the soil moisture zone is probably of real importance, a more accurate physical model takes this zone into account. How to model these two things simultaneously is interesting with regards to understanding the physics, and the predictions can also improve from this.

A

Fusing of Estimates

The filters used in this result are based on the theory of fusing of different estimates in an optimal way. In this section the ideas behind these mathematics are explained, along with a mathematical derivation of the equations used. The explanation is largely based on [22]. The Kalman Filter was invented by Rudolf E. Kálmán in 1960 , the original paper is still available today [15] and is cited over 38000 times.
The idea behind the Kalman Filter is straightforward: suppose there is a situation when you have measurements about a true state over a certain period, and a mathematical model describing how the state is expected to change over time. Both of these are not completely certain, the measurements contains errors and the model is an imperfect realisation of the real world. The Kalman Filter seeks a way to combine these two different sources in a way such a way that you obtain an estimate which is optimal. It will use the mathematical model to make a prediction about the new state, and combine this with a measurement to shift the prediction.
After this, a way to combine these estimates optimally is given. Since the Kalman Filter is usually stated in matrix form, the following section looks at vector estimates which are combined linearly by matrices. With all the tools explained previously, the reader will have a firm grasp of the mathematics behind combining estimates, and a general form of the Kalman Filter will be stated.

A.1. Mathematical Basics

The Kalman Filter is an advanced piece of statistics. To begin understanding it one must first have a firm grasp of certain aspects of probability and statistics. In this section this foundation is given, with additional resources for further reading.

A.1.1. Probability

A measuring device can be viewed as a continuous random variable X with probability density function (pdf) $f_X : \mathbb{R} \to [0,1]$. Since this is a pdf, this satisfies certain conditions, namely: Given y_1 and y_2 the following conditions should hold:

$$\begin{aligned} \int_{y_1}^{y_2} f_X(x)\,dx &\geq 0 \\ \lim_{y_1 \to -\infty} \lim_{y_2 \to \infty} \int_{y_1}^{y_2} f_X(x)\,dx &= 1 \end{aligned} \tag{A.1}$$

The first of this described the possibility of the event $P(y_1 < X < y_2)$, which should be positive. The second condition states that any event can have at most a probability of 1.

A.1.2. Statistics

It was assumed that the pdf associated with the measuring device X is an instance of the normal distribution, e.g. that it has the following form:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{A.2}$$

here, μ is called the mean and σ^2 the variance. To understand that measurements can be viewed as instances of a normal distribution, a small explanation about the normal distribution is given:
The normal distribution means that you have a high probability to end up close to a middle value which is the mean μ, and you have an equal probability to have a result higher or lower, it is symmetric. The further you get from μ, the lower the probability of a result becomes, how fast this probability decreases is captured in the variance σ^2, a high variance means that a result deviating a lot is more likely than if the variance was low.
Measurements can easily be seen as something that follows the pattern outlined above, the measurements are centered around some middle value, with equal probability to score lower or higher, and how certain you are about the measurement is then captured in the variance. If there is a high confidence in your measurement the variance is low and vice versa. It was also assumed that the measurement is **unbiased**; the middle value or measurement mean μ has the same value as the value of the true state x^t, however this does not have to be the case. When $x^t \neq \mu$ it is said the measurement is **biased**. In the future, we will usually assume our measurements and estimates to be unbiased. Measurements are samples from a normal distribution, but with a single measurement it is not possible to derive μ exactly.
The measurements don't affect each other, they are affected in unrelated ways. This means that knowing the value of x_1 might give us new information about x_2 (they are not independent) but the mean of x_2 wont change. This is formalized by requiring the corresponding random variables x_1 and x_2 to be **uncorrelated**. Mathematically this can be stated as $E(x_2|x_1) = E(x_2)$, which is equivalent with saying $E[(x_1-\mu_1)](x_2-\mu_2)] = 0$. Here, $E(X)$ is the expectation.

A.2. Fusing Estimates

The Kalman seeks to combine different estimates of a true state x^t into a single "optimal" estimate. A general introduction and intuition to the optimal fusing of scalar estimates is given in this section. First, what it means to combine estimates, and how the mean and variance change as a result of this. Then, it investigated how to combine the estimates optimally.

A.2.1. Scalar Estimates Properties

Suppose there are $x_1 \sim p_1(\mu_1, \sigma_1^2), \ldots, x_n \sim p_n(\mu_n, \sigma_n^2)$ pairwise uncorrelated random variables corresponding to different estimates of the true state x^t. We assume that all of these different measurements are unbiased, i.e. $\mu_1, \ldots, \mu_n = x^t$. The goal is to create a new random variable $y \sim p_y(\mu_y, \sigma_y^2)$ such that $y = \sum_{i=1}^n \alpha_i x_i$ with $\alpha_i \in \mathbb{R}$. Since y is a linear combination of the x_i, it is called a **linear estimator**.
The mean of y can be expressed in terms of the means of the x_i as follows:

$$\mu_y = E[\sum_{i=1}^{n} \alpha_i x_i] \overset{*}{=} \sum_{i=1}^{n} \alpha_i E[x_i] = \sum_{i=1}^{n} \alpha_i \mu_i \tag{A.3}$$

In $\overset{*}{=}$, the linearity of the expectation value is used. So, since x_i is an unbiased estimator for all $i, \mu_i = x^t$ and since we want the resulting estimator to be unbiased we know obtain that $\sum_{i=1}^n \alpha_i = 1$, then $\mu_y = x^t$, so the resulting estimator is an unbiased estimator for the true value.

The variance can also be derived:

$$\begin{aligned}
\sigma_y^2 = Var(Y) &= E[(Y-\mu_y)^2] \\
&= E[(\sum_{i=1}^{n} \alpha_i x_i - \sum_{i=1}^{n} \alpha_i \mu_i)^2] \\
&= E[(\sum_{i=1}^{n} \alpha_i (x_i - \mu_i))^2] \\
&= E[\sum_{i=1}^{n} \alpha_i (x_i - \mu_i) \sum_{j=1}^{n} \alpha_j (x_j - \mu_j)] \\
&= E[\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j (x_i - \mu_i)(x_j - \mu_j)] \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j E[(x_i - \mu_i)(x_j - \mu_j)] \\
&\overset{\circ}{=} \sum_{i=1}^{n} \alpha_i^2 E[(x_i - \mu_i)^2] \\
&= \sum_{i=1}^{n} \alpha_i^2 \sigma_i^2
\end{aligned} \tag{A.4}$$

Where in the step with $\overset{\circ}{=}$ it is used that the random variables are pairwise uncorrelated.
It is good to see that both the mean and variance can be simply combined linearly as well.

A.2.2. Fusing Scalar Estimates

Knowing how you can to combine estimates, one can investigate how to do this combination optimally. An "optimal combination" can be defined as a linear combination of the estimates with the α_i chosen such that the variance of the combined estimate is lowest. As stated before, a low variance can be seen as a high confidence estimate, since deviation from the mean is unlikely.
We will limit ourselves to two estimates, one corresponding to the measurement in the Kalman Filter, one corresponding to the model estimate: $x^f \sim p_{x^f}(x^t, \sigma_f^2), x^m \sim p_y(x^t, \sigma_m^2)$ where x^f is the forecast made by a mathematical model and x^m a measurement of x^t, which represents the true value. The goal now is to find a new estimator x^a which approaches x^t optimally by combining x^f and x^m. It is assumed that the estimators are unbiased and uncorrelated.
In the previous section it was explained how to combine this estimate into a new estimate, we get the following:

$$x^a = \alpha_1 x^f + \alpha_2 x^m$$

As before, here, $0 \leq \alpha_1, \alpha_2 \leq 1$ and $\alpha_1 + \alpha_2 = 1$ and the new estimator is also an unbiased estimator for x^t
The new error variance can be calculated using the result obtained in the previous section:

$$\begin{aligned}
\sigma_a^2 &= \alpha_1^2 \sigma_f^2 + \alpha_2^2 \sigma_m^2 \\
&= \alpha_1^2 \sigma_f^2 + (1-\alpha_1)^2 \sigma_m^2 \\
&= \alpha_1^2 \sigma_f^2 + \sigma_m^2 - 2\alpha\sigma_m^2 + \alpha_1^2 \sigma_m^2
\end{aligned} \tag{A.5}$$

But the interesting question is, when is this variance lowest? Or, for what value of α_1 is the variance of x^a minimal? To do this we look at the derivative with respect to α_1 and find the minimal value:

$$\begin{aligned}
\frac{d\sigma_a^2}{d\alpha_1} = -2\sigma_m^2 + 2\alpha_1(\sigma_m^2 + \sigma_f^2) &= 0 \\
2\alpha_1(\sigma_m^2 + \sigma_f^2) &= 2\sigma_m^2 \\
\alpha_1 &= \frac{\sigma_m^2}{(\sigma_m^2 + \sigma_f^2)}
\end{aligned} \tag{A.6}$$

Now, we arrive at the following optimal combination of x_1 and x_2.

$$\begin{aligned} x^a &= \alpha_1 x^f + \alpha_2 x^m \\ &= \alpha_1 x^f + (1-\alpha_1) x^m \\ &= x^m + \alpha_1 (x^f - x^m) \\ &= x^m + \frac{\sigma_m^2}{(\sigma_m^2 + \sigma_f^2)} (x^m - x^f) \end{aligned} \tag{A.7}$$

Here, $\frac{\sigma_m^2}{(\sigma_m^2+\sigma_f^2)}$ is called the **Kalman Gain**
With this, the error variance becomes:

$$\begin{aligned} \sigma_a^2 &= \sigma_f^2 - 2\frac{\sigma_f^2}{(\sigma_m^2+\sigma_f^2)}\sigma_f^2 + (\frac{\sigma_f^2}{(\sigma_m^2+\sigma_f^2)})^2(\sigma_m^2+\sigma_f^2) \\ &= \sigma_f^2 - 2\frac{(\sigma_f^2)^2}{(\sigma_m^2+\sigma_f^2)} + \frac{(\sigma_f^2)^2}{(\sigma_m^2+\sigma_f^2)} \\ &= \sigma_f^2(1 - \frac{\sigma_f^2}{(\sigma_m^2+\sigma_f^2)}) \\ &= \sigma_f^2\sigma_m^2(\frac{1}{\sigma_m^2+\sigma_f^2}) \end{aligned} \tag{A.8}$$

Here, we also see that $\sigma_a^2 < \sigma_m^2, \sigma_a^2 < \sigma_f^2$, the variance of our new estimator is lower than the variance of both of the original estimators, in other words, it can immediately be seen from the expression of the error variance that our new estimator is better then either estimator independently.

A.2.3. Vector estimates

Now we know how to fuse scalar estimates optimally. However, the Kalman Filter is described for vector estimates. For this, we need to generalize the earlier results to vectors.
Suppose there are $\boldsymbol{x}_1 \sim p_1(\boldsymbol{\mu}_1, \Sigma_1), \ldots, \boldsymbol{x}_n \sim p_n(\boldsymbol{\mu}_n, \Sigma_n)$ pairwise uncorrelated random variables corresponding to different estimates of the true state $\boldsymbol{x}^t$, all with dimension m. Now, the variance of the scalar estimates is replaced with a **Covariance Matrix** Σ_i. This is the matrix $E[(\boldsymbol{x_i} - \boldsymbol{\mu}_i)(\boldsymbol{x_i} - \boldsymbol{\mu}_i)^T]$. Now, entry (k,l) of the matrix is the covariance between the k and l components of $\boldsymbol{x}_i$, with the diagonal containing the variance of each element.
We again assume that all of these different measurements are unbiased, i.e. $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_n = \boldsymbol{x^t}$. To fuse an estimate linearly we create a new random variable $\boldsymbol{y} \sim p_y(\boldsymbol{\mu}_y, \Sigma_y)$ such that $\boldsymbol{y} = \sum_{i=1}^n A_i \boldsymbol{x}_i$ with A_i $m \times m$ matrices. Again, the mean of $\boldsymbol{y}$ can be expressed in terms of the means of the $\boldsymbol{x}_i$ as follows:

$$\boldsymbol{\mu}_y = E(\sum_{i=1}^n A_i \boldsymbol{x}_i) = \sum_{i=1}^n A_i E(\boldsymbol{x}_i) = \sum_{i=1}^n A_i \boldsymbol{\mu}_i \tag{A.9}$$

Since for all $i, \boldsymbol{\mu}_i = \boldsymbol{x}^t$ and we again want the resulting estimate to be an unbiased estimator of the state, we require $\sum_{i=1}^n A_i = 1$, then we obtain that $\boldsymbol{\mu}_y = \boldsymbol{x}^t$.

The variance can also be derived:

$$\begin{aligned}
\Sigma_y &= E[(\boldsymbol{y}-\boldsymbol{\mu}_y)(\boldsymbol{y}-\boldsymbol{\mu}_y)^T] \\
&= E[(\sum_{i=1}^{n} A_i x_i - \sum_{i=1}^{n} A_i\mu_i)(\sum_{i=1}^{n} A_i x_i - \sum_{i=1}^{n} A_i\mu_i)^T] \\
&= E[\sum_{i=1}^{n} A_i(x_i-\mu_i)(\sum_{i=1}^{n} A_i(x_i-\mu_i))^T] \\
&= E[\sum_{i=1}^{n} A_i(x_i-\mu_i)\sum_{i=1}^{n}(x_i-\mu_i)^T A_i^T] \\
&= E[\sum_{i=1}^{n}\sum_{j=1}^{n} A_i(x_i-\mu_i)(x_j-\mu_j)A_j^T] \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} A_i E[(x_i-\mu_i)(x_j-\mu_j)]A_j^T \\
&\stackrel{\circ}{=} \sum_{i=1}^{n} A_i E[(x_i-\mu_i)^2]A_i^T \\
&= \sum_{i=1}^{n} A_i\Sigma_i A_i^T
\end{aligned} \tag{A.10}$$

Again in step $\stackrel{\circ}{=}$ the pairwise uncorrelation is used.

A.2.4. Fusing Vector Estimates

The final step to take is how to fuse vector estimates. Let now $\boldsymbol{x}^f, \boldsymbol{x}^m, \boldsymbol{x}^t$ again the forecast, measurement and the true value as before, but now vectors. The measurements are unbiased and the covariance matrix of the forecast and the measurement are known and denoted by Σ_f, Σ_m. Again we create a new random variable $\boldsymbol{x}^a = A_1\boldsymbol{x}^f + A_2\boldsymbol{x}^m$, for which we want to minimize the new covariance matrix Σ_a.

$$\Sigma_a = A_1\Sigma_f A_1^T + A_2\Sigma_m A_2^T \tag{A.11}$$

But now, how to find the minimal value of the A_i? It is evident that this has now become more difficult than in the scalar case. The first question that arises naturally is what does it mean to minimize a matrix? There are multiply answers to this question, here it is chosen that we would like to minimize the sum of the variances, the sum of the diagonal entries of Σ_a. So, want to find a function that has this has output and can be minimized for the A_i. To this end it is useful to recognize that if matrix $A = bb^T$, a neat expression for the sum of it's diagonals is $b^T b$. Now we can rewrite the minimization in the form as stated below

$$f(A_1,A_2) = E[(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T A_1^T A_1(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T A_2^T A_2(\boldsymbol{x}_2-\boldsymbol{\mu}_2)] + <\Lambda,(A_1+A_2-I)> \tag{A.12}$$

Minimizing this function for the A_i results in the minimal A_i's we are looking for. Here Λ is introduced to function as a matrix of Lagrange multipliers. This is used often in constrained optimization, for further reading access this book [3]. Using the differentiation rules as stated in [23], we can now differentiate this function with respect to an A_i. Here, we chose to show the derivative for A_1, but note that for A_2 the result is the same except that name of some variables change.

$$\begin{aligned}
\frac{df(A_1,A_2)}{dA_1} &= \frac{d}{dA_1}E[(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T A_1^T A_1(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + (\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T A_2^T A_2(\boldsymbol{x}_2-\boldsymbol{\mu}_2)] + \frac{d}{dA_1}Tr(\Lambda^T(A_1+A_2-I)) \\
&= E[\frac{d}{dA_1}(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T A_1^T A_1(\boldsymbol{x}_1-\boldsymbol{\mu}_1) + \frac{d}{dA_1}(\boldsymbol{x}_2-\boldsymbol{\mu}_2)^T A_2^T A_2(\boldsymbol{x}_2-\boldsymbol{\mu}_2)] + \frac{d}{dA_1}(Tr(\Lambda^T A_1) + Tr(\Lambda^T A_2) - Tr(\Lambda^T I)) \\
&= E[2A_1((\boldsymbol{x}_1-\boldsymbol{\mu}_1)(\boldsymbol{x}_1-\boldsymbol{\mu}_1)^T] + \Lambda
\end{aligned} \tag{A.13}$$

Setting this to 0 for A_1 and A_2 we get

$$A_1\Sigma_f = A_2\Sigma_m = -\frac{1}{2}\Lambda \tag{A.14}$$

From this equality, we know A_1 and A_2 must have the following form:

$$\begin{aligned} A_1 &= B\Sigma_f^{-1} \\ A_2 &= B\Sigma_m^{-1} \end{aligned} \tag{A.15}$$

We can calculate the value of B with the fact that $A_1 + A_2 = I$:

$$\begin{aligned} B\Sigma_f^{-1} + B\Sigma_m^{-1} &= I \\ B(\Sigma_f^{-1} + \Sigma_m^{-1}) &= I \\ B &= (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1} \end{aligned} \tag{A.16}$$

So:

$$\begin{aligned} A_1 &= (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_f^{-1} \\ A_2 &= (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_m^{-1} \end{aligned} \tag{A.17}$$

The final thing to do is compute the covariance matrix of the new variable $\boldsymbol{y}$. Using the result from the previous section we can derive this as follows, here it is important to note that a covariance matrix is symmetric, it is it's own transpose. Also, the matrix rule $(A^{-1} + B^{-1})^{-1} = A(A+B)^{-1}B$ is used a lot.

$$\begin{aligned} \Sigma_y &= (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_f^{-1}\Sigma_f((\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_f^{-1})^T + (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_m^{-1}\Sigma_m((\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_m^{-1})^T \\ &= (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_f^{-1}(\Sigma_f^{-1} + \Sigma_m^{-1})^{-1} + (\Sigma_f^{-1} + \Sigma_m^{-1})^{-1}\Sigma_m^{-1}(\Sigma_f^{-1} + \Sigma_m^{-1})^{-1} \\ &= \Sigma_f^{-1}(\Sigma_f + \Sigma_m)^{-1}\Sigma_m(\Sigma_f + \Sigma_m)^{-1}\Sigma_m + \Sigma_f(\Sigma_f + \Sigma_m)^{-1}\Sigma_f(\Sigma_f + \Sigma_m)^{-1}\Sigma_m \\ &= \Sigma_f(\Sigma_f + \Sigma_m)^{-1}(\Sigma_m + \Sigma_f)(\Sigma_f + \Sigma_m)^{-1}\Sigma_m \\ &= \Sigma_f(\Sigma_f + \Sigma_m)^{-1}\Sigma_m \end{aligned} \tag{A.18}$$

Now, the Kalman Gain matrix is the value of A_1 that minimizes the variance of $\boldsymbol{x}^a$, this can be stated in various form, we choose the following:

$$K = \Sigma_f(\Sigma_f + \Sigma_m)^{-1} \tag{A.19}$$

B

Code

B.1. Load input

```
import csv
from datetime import datetime
import numpy as np

data = []
with open("data/timeseries/9012.csv") as csvfile:
    reader = csv.reader(csvfile)
    for row in reader:
        data.append(row)

data = data[1:]
# data has shape: [[time, prec, deep head, polder head, meas, BOFEK, surface level], [...], [...], ...
times, prec, h_d, h_p, measurements, constants = (
    np.array([datetime.fromisoformat(row[0]) for row in data]),
    np.array([float(row[1]) for row in data]),
    np.array([float(row[2]) for row in data]),
    np.array([float(row[3]) for row in data]),
    np.array([float(row[4]) for row in data]),
    {
        "groundtype": float(data[0][5]),
        "sl": float(data[0][6]),
        "cd": float(data[0][7]),
        "c1": float(data[0][8]),
        "c2": float(data[0][9]),
        "Sigma_y": 0.0049,
        "Sigma_m": 0.0841,
        "c_s": 1,
        "c_c": 1,
    },
)

BOFEK = {
    # code: (a, b, c, d)
    101: (0.000303691, 1285.538567, 0.31755023, 1.293560239),
    102: (-0.00171752, 1568.493141, 1.028670946, 1.028078548),
    103: (-0.00144013, 1008.565908, 0.537106661, 1.12531779),
    104: (0.005371247, 40313.54772, 0.103202519, 2.549597467),
    105: (0.001939729, 1276.551716, 0.518408741, 1.172037814),
```

```
    106: (-0.002824644, 1063.402536, 0.444133433, 1.184539211),
    107: (0.00049918, 739.4457865, 0.373017707, 1.204034812),
    108: (-0.002829721, 800.5077888, 0.6564414, 1.03068723),
    109: (-0.002225621, 876.5315566, 1.180106953, 0.907188721),
    110: (0.001230167, 1197.579027, 0.608512682, 1.159873983),
    ...
}

if __name__ == "__main__":
    import matplotlib.pyplot as plt

    plt.plot(times, h_d, label="deep␣head")
    plt.plot(times, h_p, label="polder␣head")
    plt.plot(times, [constants["sl"]] * len(times), label="surface␣level")
    plt.title("Input␣data")
    plt.figtext(
        0, 0, f"cd␣=␣{constants['cd']},␣c1␣=␣{constants['c1']}),␣c2␣=␣{constants['c2']}"
    )
    plt.legend()
    plt.show()
```

B.2. Models

```
import sys

sys.path.append(r"C:\Users\moos.castelijn\Documents\Scriptie")

import numpy as np
from numpy.random import normal

from data.importInput import BOFEK


def groundwater_euler(h_opt, p, h_d, h_p, constants, dt=1):
    """
    Iterative model for the groundwater


            Parameters:
                    level (float): Current groundwater level
                    q (float): Groundwater entering the cell from bellow (recharge)
                    p (float): Groundwater entering the cell from above (precipitation)

            Returns:
                    predict (float): prediction of the groundwater according to the model
    Euler solution is not correct.. Now it is!! Model validated check.

    """
    groundtype = constants["groundtype"]
    sl = constants["sl"]
    cd = constants["cd"]
    c1 = constants["c1"]
    c2 = constants["c2"]

    s = storage(h_opt, groundtype, sl)
```

```
    F = 1 - (dt / s) * (1 / c1 + 1 / c2 + 1 / cd)
    B = dt / s * np.array([1 / cd, 1, (c1 + c2) / (c1 * c2)])
    u = np.array([[h_d], [p], [h_p]])

    h_pred = min((F * h_opt + float(np.matmul(B, u))), sl)
    return h_pred, F, B, u


def groundwater_exponential(h_opt, p, h_d, h_p, constants, dt=1):
    # Retrieving Constants
    groundtype = constants["groundtype"]
    sl = constants["sl"]
    c_c = constants["c_c"]
    cd = constants["cd"] * c_c
    c1 = constants["c1"] * c_c
    c2 = constants["c2"] * c_c
    c_s = constants["c_s"]

    # Calculate Storage for this h
    s = c_s * storage(h_opt, groundtype, sl)

    # defining matrices and vectors for Model Calculations
    c = 1 / cd + 1 / c1 + 1 / c2
    d = np.exp(-dt * c / s)
    F = d
    B = 1 / c * (1 - d) * np.array([1 / cd, 1, (c1 + c2) / (c1 * c2)])
    u = np.array([[h_d], [p], [h_p]]) + [[normal(0, 0)], [0], [0]]

    # Model Calculations, removed: + normal(0, Sigma_m)
    h_pred = min((F * h_opt + float(np.matmul(B, u))), sl)
    return h_pred, F, B, u


def storage(h, groundtype, sl):
    if h >= sl:
        f = -5
    else:
        a, b, c, d = BOFEK[groundtype]
        num1 = a * b
        num2 = c * ((sl - h) * 100) ** d
        den = 1 / (b + ((sl - h) * 100) ** d)
        f = (num1 + num2) * den
    return np.clip(f, 0.01, 0.9)
```

B.3. Model Comparison

```
from data.importInput import times, prec, measurements, h_d, h_p, constants
from models.groundwater_model import (
    groundwater_exponential,
    groundwater_euler,
)
from tools import MSE

import matplotlib.pyplot as plt
```

```
# use 1/2 year of data for calibration, 2nd half for testing

t = 365


h_euler_lst = [measurements[0]]
h_analytical_lst = [measurements[0]]

for i in range(t - 1):
    h_euler_lst.append(
        groundwater_euler(
            h_euler_lst[-1],
            prec[i],
            h_d[i],
            h_p[i],
            constants,
        )[0]
    )
    h_analytical_lst.append(
        groundwater_exponential(
            h_analytical_lst[-1],
            prec[i],
            h_d[i],
            h_p[i],
            constants,
        )[0]
    )

MSE_euler = MSE(measurements, h_euler_lst)
MSE_exponential = MSE(measurements, h_analytical_lst)
print(MSE_euler, MSE_exponential)

plt.plot(times[:t], h_euler_lst, label="Prediction␣from␣Euler␣model")
plt.plot(times[:t], measurements[:t], label=f"Measurements")
plt.title("Euler␣solution")
plt.legend()
plt.show()

plt.plot(times[:t], h_analytical_lst, label="Predicitons␣from␣analytical␣solution")
plt.plot(times[:t], measurements[:t], label=f"Measurements")
plt.title("Anaylical␣solution")
plt.legend()
plt.show()
```

B.4. Kalman Filter

```
from models.groundwater_model import groundwater_exponential
from data.importInput import times, prec, h_d, h_p, measurements, constants

import numpy as np
from numpy.random import normal


def KMFilter1D():
    h_opt = measurements[0]
    Sigma_m = constants["Sigma_m"]
```

```
    Sigma_y = constants["Sigma_y"]
    S_opt = Sigma_y
    h_predictions, h_optimums = [h_opt], [h_opt]
    S_predictions, S_optimums = [S_opt], [S_opt]
    for i, measurement in enumerate(measurements[1:]):
        # prediction
        h_pred, F, _, _ = groundwater_exponential(
            h_opt, prec[i], h_d[i], h_p[i], constants
        )
        S_pred = F**2 * S_opt + Sigma_m
        # measurement fusion
        meas_pred = measurement
        K_t = S_pred / (S_pred + Sigma_y)
        h_opt = h_pred + K_t * (meas_pred - h_pred)
        S_opt = (1 - K_t) * S_pred

        h_predictions.append(h_pred), h_optimums.append(h_opt)
        S_predictions.append(S_pred), S_optimums.append(S_opt)
    return (
        np.array(h_predictions),
        np.array(h_optimums),
        np.array(S_predictions),
        np.array(S_optimums),
    )
```

B.5. Kalman Filter Results

```
from models.KalmanFilter import KMFilter1D
from data.importInput import times, measurements, constants
from tools import MSE, run_model

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# Dit moet nog in verslag!
# 9024 was best mooi, 8890 ook, 8907 ook

h_predictions, h_optimums, S_predictions, S_optimums = KMFilter1D()
model_solution = run_model()

title = plt.title("Kalman␣Filter␣a␣priori␣state␣estimates,␣1␣year")
plt.plot(times, measurements, label="Measurements")
plt.plot(times, h_predictions, label="A␣priori␣state␣estimates")
plt.plot(times, model_solution, alpha=0.4, label="Model␣prediction")
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_predictions[:31], label="A␣priori␣state␣estimates")
```

```
plt.plot(times[:31], model_solution[:31], alpha=0.4, label="Model␣prediction")
title = plt.title("Kalman␣Filter␣a␣priori␣state␣estimates,␣1␣month")
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title("Kalman␣Filter␣a␣priori␣state␣error␣s.␣d.␣evolution,␣1␣month")
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_predictions[:31], label="A␣priori␣state␣estimates")
ax.fill_between(
    times[:31],
    (h_predictions[:31] - np.sqrt(S_predictions[:31])),
    (h_predictions[:31] + np.sqrt(S_predictions[:31])),
    color="r",
    alpha=0.3,
)
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.ylim(-1.1, -0.2)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title("Kalman␣Filter␣a␣posteriori␣state␣estimates,␣1␣year")
plt.plot(times, measurements, label="Measurements")
plt.plot(times, h_optimums, label="A␣posteriori␣state␣estimates")
plt.plot(times, model_solution, alpha=0.4, label="Model␣prediction")
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title("Kalman␣Filter␣a␣posteriori␣state␣estimates,␣1␣month")
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_optimums[:31], label="A␣posteriori␣state␣estimates")
plt.plot(times[:31], model_solution[:31], alpha=0.4, label="Model␣prediction")
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
```

```
title = plt.title("Kalman␣Filter␣a␣posteriori␣state␣error␣s.␣d.␣evolution,␣1␣month")
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_optimums[:31], label="A␣posteriori␣state␣estimates")
ax.fill_between(
    times[:31],
    (h_optimums[:31] - np.sqrt(S_optimums[:31])),
    (h_optimums[:31] + np.sqrt(S_optimums[:31])),
    color="r",
    alpha=0.3,
)
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.ylim(-1.1, -0.2)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

print("MSE␣predictions:␣" + str(MSE(measurements, h_predictions)))
print("MSE␣optimums:␣" + str(MSE(measurements, h_optimums)))

title = plt.title("Kalman␣Filter,␣Propagation␣of␣error␣standard␣deviation")
plt.plot(times, np.sqrt(S_predictions), label="A␣priori␣state␣error␣standard␣deviation")
plt.plot(
    times, np.sqrt(S_optimums), label="A␣posteriori␣state␣error␣standard␣deviation"
)
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

Kalman_Gain = S_predictions / (S_predictions + constants["Sigma_y"])
title = plt.title("Kalman␣Filter,␣Kalman␣Gain␣matrix")
plt.plot(times, Kalman_Gain)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

print(Kalman_Gain.mean())
```

B.6. Ensemble Kalman Filter

```
from models.groundwater_model import groundwater_exponential
from data.importInput import prec, h_d, h_p, measurements, constants

import numpy as np
from numpy.random import normal


def EnKF_state(n=200):
    Sigma_m = constants["Sigma_m"]
    Sigma_y = constants["Sigma_y"]

    h_init = [normal(measurements[0], Sigma_y) for i in range(n)]
    h_init = np.array([min(h, constants["sl"]) for h in h_init])
```

```
    h_pred = h_init.copy()
    h_opt = h_init.copy()
    noisy_measurements, h_predictions, h_optimums = (
        [h_init.copy()],
        [h_pred.copy()],
        [h_opt.copy()],
    )
    for j, measurement in enumerate(measurements[1:]):
        # Predictions for all previous optima
        for i, optima in enumerate(h_opt):
            h_pred[i], _, _, _ = groundwater_exponential(
                optima, prec[j], h_d[j], h_p[j], constants
            )
            h_pred[i] += normal(0, Sigma_m)
            h_pred[i] = min(h_pred[i], constants["sl"])
        # Usually translation of model to measured variables

        # Define Covariance Matrices
        pred_mean = h_pred.mean()
        Sigma_x = 1 / (n - 1) * sum((h_pred - pred_mean) ** 2)
        Sigma_xy = Sigma_x

        # Add noise to Measurement
        meas_t = [measurement + normal(0, Sigma_y) for m in range(n)]

        # Fuse Prediction with Measurement
        K = Sigma_xy * ((Sigma_y + Sigma_x) ** (-1))
        for j, pred in enumerate(h_pred):
            h_opt[j] = pred + K * (meas_t[j] - pred)

        noisy_measurements.append(meas_t)
        h_predictions.append(h_pred.copy()), h_optimums.append(h_opt.copy())
    return h_predictions, h_optimums, noisy_measurements
```

B.7. Ensemble Kalman Filter Results

```
from models.EnKF_state import EnKF_state
from data.importInput import times, measurements, constants
from tools import MSE, run_model

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import numpy as np

# Create Results
h_predictions, h_optimums, S_predictions, S_optimums = EnKF_state()
model_solution = run_model()


title = plt.title("Ensemble␣Kalman␣Filter␣a␣priori␣state␣estimates,␣1␣year")
plt.plot(times, measurements, label="Measurements")
plt.plot(times, h_predictions, label="A␣priori␣state␣estimates")
plt.plot(times, model_solution, alpha=0.4, label="Model␣prediction")
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
```

```
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_predictions[:31], label="A priori state estimates")
plt.plot(times[:31], model_solution[:31], alpha=0.4, label="Model prediction")
title = plt.title("Ensemble Kalman Filter a priori state estimates, 1 month")
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time (days)")
plt.ylabel("depth (meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title(
    "Ensemble Kalman Filter a priori state error s. d. evolution, 1 month"
)
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_predictions[:31], label="A priori state estimates")
ax.fill_between(
    times[:31],
    (h_predictions[:31] - np.sqrt(S_predictions[:31])),
    (h_predictions[:31] + np.sqrt(S_predictions[:31])),
    color="r",
    alpha=0.3,
)
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time (days)")
plt.ylabel("depth (meter)")
plt.ylim(-1.1, -0.2)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title("Ensemble Kalman Filter a posteriori state estimates, 1 year")
plt.plot(times, measurements, label="Measurements")
plt.plot(times, h_optimums, label="A posteriori state estimates")
plt.plot(times, model_solution, alpha=0.4, label="Model prediction")
plt.xlabel("time (days)")
plt.ylabel("depth (meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title("Ensemble Kalman Filter a posteriori state estimates, 1 month")
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_optimums[:31], label="A posteriori state estimates")
```

```
plt.plot(times[:31], model_solution[:31], alpha=0.4, label="Model␣prediction")
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.ylim(-1.1, -0.2)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
title = plt.title(
    "Ensemble␣Kalman␣Filter␣a␣posteriori␣state␣error␣s.␣d.␣evolution,␣1␣month"
)
plt.plot(times[:31], measurements[:31], label="Measurements")
plt.plot(times[:31], h_optimums[:31], label="A␣posteriori␣state␣estimates")
ax.fill_between(
    times[:31],
    (h_optimums[:31] - np.sqrt(S_optimums[:31])),
    (h_optimums[:31] + np.sqrt(S_optimums[:31])),
    color="r",
    alpha=0.3,
)
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.ylim(-1.1, -0.2)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

print(MSE(measurements, h_optimums))
print(MSE(measurements[:31], h_optimums[:31]))

title = plt.title("Ensemble␣Kalman␣Filter,␣Propagation␣of␣error␣standard␣deviation")
plt.plot(times, np.sqrt(S_predictions), label="A␣priori␣state␣error␣standard␣deviation")
plt.plot(
    times, np.sqrt(S_optimums), label="A␣posteriori␣state␣error␣standard␣deviation"
)
plt.xlabel("time␣(days)")
plt.ylabel("depth␣(meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

Kalman_Gain = S_predictions / (S_predictions + constants["Sigma_y"])
title = plt.title("Ensemble␣Kalman␣Filter,␣Kalman␣Gain␣matrix")
plt.plot(times, Kalman_Gain)
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()
```

B.8. Ensemble Kalman Filter with regular Kalman Filter comparison

```
from models.EnKF_state import EnKF_state
```

```
from models.KalmanFilter import KMFilter1D
from tools import MSE

import numpy as np

MSE_priori_state = []
MSE_posteriori_state = []
MSE_priori_sd = []
MSE_posteriori_sd = []
for i in range(50):
    (
        h_predictions_EnKF,
        h_optimums_EnKF,
        S_predictions_EnKF,
        S_optimums_EnKF,
    ) = EnKF_state()
    h_predictions_KF, h_optimums_KF, S_predictions_KF, S_optimums_KF = KMFilter1D()
    MSE_priori_state.append(MSE(h_predictions_KF, h_predictions_EnKF))
    MSE_posteriori_state.append(MSE(h_optimums_KF, h_optimums_EnKF))
    MSE_priori_sd.append(MSE(np.sqrt(S_predictions_KF), np.sqrt(S_predictions_EnKF)))
    MSE_posteriori_sd.append(MSE(np.sqrt(S_optimums_KF), np.sqrt(S_optimums_EnKF)))

print(np.array(MSE_priori_state).mean())
print(np.array(MSE_posteriori_state).mean())
print(np.array(MSE_priori_sd).mean())
print(np.array(MSE_posteriori_sd).mean())
```

B.9. Ensemble Kalman Filter, State and Parameter

```
from models.groundwater_model import groundwater_exponential
from data.importInput import prec, h_d, h_p, measurements, constants
from copy import deepcopy

import numpy as np
from numpy.random import normal


def EnKF_state_param(
    n=100, t=182, vary=["c_s"], constants=constants, measurements=measurements
):
    # initialize
    ensemble_init, ensemble_constants = initialization(n, vary, constants=constants)

    ensemble_pred = deepcopy(ensemble_init)
    ensemble_opt = deepcopy(ensemble_init)
    ensemble_predictions, ensemble_optimums = (
        [ensemble_init.copy()],
        [ensemble_init.copy()],
    )
    for day, measurement in enumerate(measurements[1 : t + 1]):
        ensemble_pred = deepcopy(ensemble_opt)
        ensemble_pred, ensemble_opt, ensemble_constants = EnKF_sp_step(
            day, measurement, ensemble_pred, ensemble_opt, ensemble_constants, n, vary
        )

        ensemble_predictions.append(deepcopy(ensemble_pred))
```

```
        ensemble_optimums.append(deepcopy(ensemble_opt))

    return ensemble_predictions, ensemble_optimums


def initialization(n, vary, constants=constants):
    Sigma_m = constants["Sigma_m"]
    Sigma_y = constants["Sigma_y"]

    constantvalue_lst = []
    for param in vary:
        constantvalue_lst.append(constants[param])

    ensemble_init = []
    for i in range(n):
        ensemble_init.append(
            np.array(
                [
                    min(
                        normal(measurements[0], Sigma_m),
                        constants["sl"],
                    ),
                ]
                + [
                    max(param_value + normal(0, param_value / 2.5), 0.01)
                    for param_value in constantvalue_lst
                ]
            )
        )

    ensemble_constants = [constants.copy() for i in range(n)]
    for i in range(n):
        for j, param in enumerate(vary):
            ensemble_constants[i][param] = ensemble_init[i][j + 1]
    return ensemble_init, ensemble_constants


def EnKF_sp_step(
    day, measurement, ensemble_pred, ensemble_opt, ensemble_constants, n, vary
):
    Sigma_m = constants["Sigma_m"]
    Sigma_y = constants["Sigma_y"]
    # Predictions for all previous optima
    for i, ensemble_member_opt in enumerate(ensemble_opt):
        ensemble_pred[i][0], _, _, _ = groundwater_exponential(
            ensemble_member_opt[0], prec[day], h_d[day], h_p[day], ensemble_constants[i]
        )
        ensemble_pred[i][0] += normal(0, Sigma_m)
        ensemble_pred[i][0] = min(ensemble_pred[i][0], constants["sl"])

    # Usually translation of model to measured variables
    h_pred = np.array([ensemble_pred[i][0] for i in range(n)])

    # Define Covariance Matrices
    pred_mean = h_pred.mean()
    Sigma_yy = 1 / (n - 1) * sum((h_pred - pred_mean) ** 2)
```

```
    res = 0
    ensemble_mean = np.array(1 / n * sum(np.array(ensemble_pred)))
    for i, prediction in enumerate(ensemble_pred):
        res += (prediction - ensemble_mean) * (h_pred[i] - pred_mean)
    Sigma_xy = 1 / (n - 1) * res

    # Add noise to Measurement
    meas_t = [measurement + normal(0, Sigma_y) for m in range(n)]

    # Fuse Prediction with Measurement
    alpha = 1
    K = Sigma_xy * ((Sigma_yy + Sigma_y) ** (-1))
    for j, pred in enumerate(h_pred):
        ensemble_opt[j] = ensemble_pred[j] + alpha * K * (meas_t[j] - pred)
    alpha = 0
    # Clip Ensemble Constants,
    for i, ensemble in enumerate(ensemble_opt):
        for j, param in enumerate(vary):
            ensemble_opt[i][j + 1] = np.clip(
                ensemble[j + 1],
                0.1 * constants[param],
                10 * constants[param],  # ensemble pread cap
            )
            ensemble_constants[i][param] = ensemble_opt[i][j + 1]

    return ensemble_pred, ensemble_opt, ensemble_constants
```

B.10. Artificial data parameter calibration

```
from models.EnKF_state_param import EnKF_state_param
from data.importInput import times, measurements, constants, location
from tools import EnKF_result_dictionary, MSE, run_model


import numpy as np
from matplotlib import pyplot as plt
from copy import deepcopy

# geen effect: c1, c2
# effect: cd
# c_cp_exp grote onzekerheid, logisch wand waarde cp = 8333, in verg tot cd = 250
start = 0
stop = 365
amt = 1

mock_data = run_model(start=start, stop=stop, variance=constants["Sigma_y"])
measurements = mock_data

init_constants = constants

for param_value in [0.5]:
    for vary in [["c_cd_exp", "c_cp_exp"]]:
        constants = deepcopy(init_constants)
        for i, param in enumerate(vary):
```

```
            constants[param] = np.log(param_value)

        uncalibrated = run_model(start=start, stop=stop, constants=constants)

        param_lsts = {}
        for param in vary:
            param_lsts[param] = []

        MSEs_cal = []
        for i in range(amt):
            (
                ensemble_predictions,
                ensemble_optimums,
                Sigma_xy_lst,
                Sigma_yy_lst,
                Kalman_lst,
            ) = EnKF_state_param(
                n=200,
                start=start,
                stop=stop,
                vary=vary,
                constants=constants,
                measurements=measurements,
            )
            results = EnKF_result_dictionary(
                ensemble_predictions, ensemble_optimums, vary
            )
            for param in vary:
                param_lsts[param].append((results["constants"][param]))

            constants = results["constants"]
            calibrated = run_model(start=start, stop=stop, constants=constants)
            MSEs_cal.append(MSE(calibrated, measurements[start:stop]))
        for param in vary:
            print(
                f"{param} from {param_value} becomes equal to: {(sum(np.exp(param_lsts[param])) 
            )

        MSE_uncal = MSE(uncalibrated, measurements[start:stop])
        MSE_cal = sum(MSEs_cal) / amt

        print(
            f"the MSE with of {vary, param_value} uncalibrated constants: {MSE_uncal}"
        )
        print(f"the MSE with of {vary, param_value} calibrated constants: {MSE_cal}")

        print(f"{vary, param_value} now has ratio: {MSE_uncal / MSE_cal}")

    plt.plot(times[start:stop], measurements[start:stop], label="data")
    plt.plot(times[start:stop], uncalibrated, label="uncalibrated", alpha=0.5)
    plt.plot(times[start:stop], calibrated, label="calibrated")
    title = plt.title(f"Location {location}, Artificial Data Parameter Estimation")
    plt.figtext(0, 0, f"cd = {constants['cd']}, cp = {constants['cp']}")
    plt.xlabel("time (days)")
    plt.legend()
    plt.savefig(f"images/{title._text}.png")
```

```
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
for param in vary:
    plt.plot(
        times[start:stop],
        np.exp(results[param + "_means"]),
        label=param.replace("_exp", ""),
    )
    ax.fill_between(
        times[start:stop],
        np.exp(results[param + "_means"] - np.sqrt(results[param + "_variances"])),
        np.exp(results[param + "_means"] + np.sqrt(results[param + "_variances"])),
        color="b",
        alpha=0.1,
    )
plt.plot(times, [1] * len(times), alpha=0.4)
title = plt.title(f"Location␣{location},␣Artificial␣Data,␣Evolution␣of␣parameters")
plt.figtext(
    0,
    0,
    f"cd␣=␣{np.exp(constants['c_cd_exp'])},␣cp␣=␣{np.exp(constants['c_cp_exp'])}",
)
plt.xlabel("time␣(days)")
plt.yscale("log")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

Kalman_state = [K[0] for K in Kalman_lst]
for i, param in enumerate(vary):
    results[param + "_Kalman"] = [K[i + 1] for K in Kalman_lst]

title = plt.title("Evolution␣of␣c_s_exp␣and␣c_c_exp␣elements␣of␣Kalman␣Gain␣matrix")
for param in vary:
    plt.plot(
        times[start : stop - 1],
        results[param + "_Kalman"],
        label="Kalman␣Gain␣entry␣for␣" + param,
    )
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()
```

B.11. Real data parameter calibration

```
from models.EnKF_state_param import EnKF_state_param
from data.importInput import times, measurements, constants, location
from tools import EnKF_result_dictionary, MSE, param_uncertainty_fixed_time, run_model

import matplotlib.pyplot as plt
from math import sqrt
import numpy as np
```

```
print(constants["cd"], constants["cp"])
if constants["cp"] > constants["cd"]:
    vary = ["c_s_exp", "c_cd_exp"]
else:
    vary = ["c_s_exp", "c_cp_exp"]

# mandetory

start = 0
stop = 365

# initial results
print(constants)
uncalibrated = run_model(start=start, stop=stop, constants=constants)

# Create Results
(
    ensemble_predictions,
    ensemble_optimums,
    Sigma_xy_lst,
    Sigma_yy_lst,
    Kalman_lst,
) = EnKF_state_param(vary=vary, start=start, stop=stop, n=200)
results = EnKF_result_dictionary(ensemble_predictions, ensemble_optimums, vary)

# What happens to the parameters?
fig = plt.figure()
ax = fig.add_subplot(111)
for param in vary:
    plt.plot(
        times[start:stop],
        np.exp(results[param + "_means"]) * 1,
        label=param.replace("_exp", ""),
    )
    ax.fill_between(
        times[start:stop],
        (np.exp(results[param + "_means"] - np.sqrt(results[param + "_variances"])))
        * 1,
        (np.exp(results[param + "_means"] + np.sqrt(results[param + "_variances"])))
        * 1,
        color="b",
        alpha=0.1,
    )
title = plt.title(f"Location␣{location},␣Real␣Data,␣Evolution␣of␣parameters")
plt.xlabel("time␣(days)")
plt.yscale("log")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

# propagate the model again and see
constants = results["constants"]
print(constants)
c_s = np.exp(constants["c_s_exp"])
c_cp = np.exp(constants["c_cp_exp"])
c_cd = np.exp(constants["c_cd_exp"])
```

```
print(f"c_s = {c_s}, c_cp = {c_cp}, c_cd = {c_cd}")

calibrated = run_model(start=start, stop=stop, constants=constants)

# Plot meas and optima

title = plt.title(f"Location {location}, Calibration of {vary[0]} and {vary[1]}")
plt.plot(times[start:stop], measurements[start:stop], label="Measurements")
plt.plot(times[start:stop], uncalibrated, label="Non-calibrated", alpha=0.5)
plt.plot(times[start:stop], calibrated, label="Calibrated")
plt.figtext(0, 0, f"cd = {constants['cd']}, cp = {constants['cp']})")
plt.xlabel("time (days)")
plt.ylabel("depth (meter)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

RMSE_uncal = sqrt(MSE(uncalibrated, measurements[start:stop]))
RMSE_cal = sqrt(MSE(calibrated, measurements[start:stop]))

print(f"the RMSE with the calibrated constants: {RMSE_cal}")
print(f"the RMSE with the uncalibrated constants: {RMSE_uncal}")


print(RMSE_uncal / RMSE_cal)
```

B.12. Predictions

```
import matplotlib.pyplot as plt
from copy import deepcopy
from math import sqrt
import numpy as np
import matplotlib.dates as mdates


from models.EnKF_state_param import EnKF_state_param
from tools import run_model, EnKF_result_dictionary, MSE
from data.importInput import constants, times, measurements, location


# set inportant values
start_calibration = 0
stop_calibration = 242
start_prediction = stop_calibration + 1
stop_prediction = 365
if constants["cp"] > constants["cd"]:
    vary = ["c_s_exp", "c_cd_exp"]
else:
    vary = ["c_s_exp", "c_cp_exp"]

# obtain calibrated parameters
ensemble_predictions, ensemble_optimums, _, _, _ = EnKF_state_param(
    vary=vary, start=start_calibration, stop=stop_calibration, n=200
)
results = EnKF_result_dictionary(ensemble_predictions, ensemble_optimums, vary)
```

```
constants = deepcopy(results["constants"])

# get the 2 predictions
uncalibrated = run_model(start=start_prediction, stop=stop_prediction)
calibrated = run_model(
    start=start_prediction, stop=stop_prediction, constants=constants
)

# compare results

fig = plt.figure()
ax = fig.add_subplot(111)
plt.plot(times, measurements, label="measurements")
plt.plot(
    times[start_prediction:],
    uncalibrated,
    label="model␣with␣uncalibrated␣parameters",
    alpha=0.5,
)
plt.plot(times[start_prediction:], calibrated, label="model␣with␣calibrated␣parameters")
for i in [0, 1]:
    for j in [0, 1]:
        constants[vary[0]] = results["constants"][vary[0]] + (-1) ** i * np.sqrt(
            results[vary[0] + "_variances"][-1]
        )
        constants[vary[1]] = results["constants"][vary[1]] + (-1) ** j * np.sqrt(
            results[vary[1] + "_variances"][-1]
        )
        newoutput = run_model(
            start=start_prediction, stop=stop_prediction, constants=constants
        )
        ax.fill_between(
            times[start_prediction:],
            calibrated,
            newoutput,
            color="b",
            alpha=0.1,
        )
title = plt.title(
    f"Location␣{location},␣calibrated␣until␣day␣{stop_calibration},␣then␣start␣prediction"
)
plt.xlabel("time␣(days)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

fig = plt.figure()
ax = fig.add_subplot(111)
for param in vary:
    plt.plot(
        times[start_calibration:stop_calibration],
        np.exp(results[param + "_means"]) * 1,
        label=param.replace("_exp", ""),
    )
    ax.fill_between(
        times[start_calibration:stop_calibration],
```

```
        np.exp(results[param + "_means"] - np.sqrt(results[param + "_variances"])) * 1,
        np.exp(results[param + "_means"] + np.sqrt(results[param + "_variances"])) * 1,
        color="b",
        alpha=0.1,
    )
# ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d"))
plt.yscale("log")
title = plt.title(
    f"Location␣{location},␣Parameter␣evolution␣until␣day␣{stop_calibration}"
)
plt.xlabel("time␣(days)")
plt.legend()
plt.savefig(f"images/{title._text}.png")
plt.show()

RMSE_uncal = sqrt(MSE(uncalibrated, measurements[start_prediction:]))
RMSE_cal = sqrt(MSE(calibrated, measurements[start_prediction:]))

print(f"the␣RMSE␣with␣the␣calibrated␣constants:␣{RMSE_cal}")
print(f"the␣RMSE␣with␣the␣uncalibrated␣constants:␣{RMSE_uncal}")

print(RMSE_uncal / RMSE_cal)

constants = deepcopy(results["constants"])
c_s = np.exp(constants["c_s_exp"])
c_cp = np.exp(constants["c_cp_exp"])
c_cd = np.exp(constants["c_cd_exp"])
print(f"c_s␣=␣{c_s},␣c_cp␣=␣{c_cp},␣c_cd␣=␣{c_cd}")
```

B.13. Tools

```
from copy import deepcopy
import matplotlib.pyplot as plt
import numpy as np
from numpy.random import normal

from data.importInput import times, constants, measurements, prec, h_d, h_p
from models.groundwater_model import groundwater_exponential, groundwater_euler


def plot(measurements, h_opt, T=365):
    plt.plot(times[:T], measurements[:T], label="Measurements")
    plt.plot(times[:T], h_opt[:T], label="Optimal␣predictions")
    plt.legend()
    plt.show()


def MSE(list1, list2):
    return ((np.array(list1) - np.array(list2)) ** 2).mean()


def EnKF_result_dictionary(ensemble_predictions, ensemble_optimums, vary):
    """
    Function to compute the result from EnKF with state & parameter estimation
```

```
    returns:
        Dictionary filled with:
            - "h_predictions": list, for each day an ensemble of the predictions
            - "h_optima": list, for each day an ensemble of the optima
            - "param": list, for each param, for each day, an ensemble of the parameters
            - "h_pred_means": list, for each day the mean of the prediction ensemble
            - "h_opt_means": list, for each day the mean of the optima ensemble
            - "param_means": list, for each day the mean of the param ensemble
            - "param_variances": list, for each param, for each day, the sample variance of that
            - "constants": dict, constants at final timestep.


    """
    results = {
        "h_predictions": [],
        "h_optima": [],
    } | {param: [] for param in vary}

    for t, day in enumerate(ensemble_optimums):
        for key in results.keys():
            results[key].append([])
        for i, ensemble in enumerate(day):
            results["h_predictions"][t].append(ensemble_predictions[t][i][0])
            results["h_optima"][t].append(ensemble[0])
            for i, param in enumerate(vary):
                results[param][t].append(ensemble[i + 1])
    results["h_opt_means"] = [np.array(day).mean() for day in results["h_optima"]]
    results["h_pred_means"] = [np.array(day).mean() for day in results["h_predictions"]]
    constantscopy = deepcopy(constants)
    for param in vary:
        results[param + "_means"] = np.array(
            [np.array(day).mean() for day in results[param]]
        )
        results[param + "_variances"] = np.array(
            [np.array(day).var() for day in results[param]]
        )
        constantscopy[param] = results[param + "_means"][-1]
    results["constants"] = constantscopy
    return results


def param_uncertainty_fixed_time(results, t=182, p1="c_s_exp", p2="c_c"):
    p1_values = results[p1][t]
    p2_values = results[p2][t]
    plt.plot(p1_values, p2_values, ".")
    plt.xlim(left=0, right=15)
    plt.ylim(bottom=0, top=15)


def run_model(
    measurements=measurements,
    constants=constants,
    start=0,
    stop=365,
    model="exponential",
```

```
    variance=0,
    returnN=False,
    returns=False,
    error=False,
    points_day=1,
):
    if model == "exponential":
        model = groundwater_exponential
    elif model == "euler":
        model = groundwater_euler
    else:
        return "Incorrect model name"
    results = [measurements[start]]
    N_lst = []
    s_lst = []
    error_lst = []
    for i in range(start, stop - 1):
        for j in range(points_day):
            result, _, s, N = model(
                results[-1], prec[i], h_d[i], h_p[i], constants, dt=1 / points_day
            )
            results.append(result + normal(0, variance))
            N_lst.append(N)
            s_lst.append(s)
    N_lst.append(N_lst[-1])
    s_lst.append(s_lst[-1])
    if error:
        return results, error_lst
    if returnN & returns:
        return results, N_lst, s_lst
    elif returnN:
        return N_lst
    elif returns:
        return s_lst
    else:
        return results
```

Bibliography

[1] Nikos Alexandratos and Jelle Bruinsma. World agriculture towards 2030/2050: the 2012 revision. AgEcon Search, 2012.

[2] Richard P Allan, Ed Hawkins, Nicolas Bellouin, and Bill Collins. Ipcc, 2021: Summary for policymakers. Intergovernmental Panel on Climate Change, 2021.

[3] Dimitri P Bertsekas. Constrained optimization and Lagrange multiplier methods. Academic press, 2014.

[4] NHI Deltares. Bodemkaart, 2022. URL `https://data.nhi.nu/geonetwork/srv/api/records/2ad30d79-1784-4895-8a71-b4e39a7207e5`.

[5] Wageningen Deltares. Home, Jan 2022. URL `https://www.mipwa.nl/`.

[6] Geir Evensen. Using the extended kalman filter with a multilayer quasi-geostrophic ocean model. Journal of Geophysical Research: Oceans, 97(C11):17905–17924, 1992.

[7] Geir Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using monte carlo methods to forecast error statistics. Journal of Geophysical Research: Oceans, 99(C5):10143–10162, 1994.

[8] Geir Evensen. The ensemble kalman filter for combined state and parameter estimation. IEEE Control Systems Magazine, 29(3):83–104, 2009.

[9] Steven Gillijns, O Barrero Mendoza, Jaganath Chandrasekar, BLR De Moor, DS Bernstein, and A Ridley. What is the ensemble kalman filter and how well does it work? In 2006 American control conference, pages 6–pp. IEEE, 2006.

[10] Mohinder S Grewal and Angus P Andrews. Kalman filtering: Theory and Practice with MATLAB. John Wiley & Sons, 2014.

[11] Harrie-Jan Hendricks Franssen and W Kinzelbach. Real-time groundwater flow modeling with the ensemble kalman filter: Joint estimation of states and parameters and the filter inbreeding problem. Water Resources Research, 44(9), 2008.

[12] Paul Hiemstra and Raymond Sluiter. Interpolation of makkink evaporation in the netherlands. 2011.

[13] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. Acta Numerica, 19:209–286, 2010.

[14] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In Signal processing, sensor fusion, and target recognition VI, volume 3068, pages 182–193. International Society for Optics and Photonics, 1997.

[15] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. Transactions of the ASME–Journal of Basic Engineering, 1960.

[16] KNMI. Evaporation - gridded daily makkink evaporation for the netherlands. - knmi data platform, 2022. URL `https://dataplatform.knmi.nl/nl/dataset/ev24-2`.

[17] KNMI. Precipitation - radar/gauge 24 hour accumulations over the netherlands - knmi data platform, 2022. URL `https://dataplatform.knmi.nl/nl/dataset/radar-corr-accum-24h-1-0`.

[18] Yuqiong Liu and Hoshin V Gupta. Uncertainty in hydrologic modeling: Toward an integrated data assimilation framework. Water resources research, 43(7), 2007.

[19] Andrew Lorenc. Relative merits of 4d-var and ensemble kalman filter. Technical report, NWP Internal Report, 2003.

[20] Dennis McLaughlin. Recent developments in hydrologic data assimilation. Reviews of Geophysics, 33 (S2):977–984, 1995.

[21] Hamid Moradkhani, Soroosh Sorooshian, Hoshin V Gupta, and Paul R Houser. Dual state–parameter estimation of hydrological models using ensemble kalman filter. Advances in water resources, 28(2): 135–147, 2005.

[22] Yan Pei, Swarnendu Biswas, Donald S Fussell, and Keshav Pingali. An elementary introduction to kalman filtering. Communications of the ACM, 62(11):122–133, 2019.

[23] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. Technical University of Denmark, 7(15):510, 2008.

[24] JJ Schroder, GJ Hilhorst, J Oenema, J Verloop, and Wim van den Berg. Bofek2020-bodemfysische schematisatie van nederland: update bodemfysische eenhedenkaart. Technical report, Wageningen Environmental Research, 2021.

[25] Soroosh Sorooshian, Kuo-lin Hsu, Erika Coppola, Barbara Tomassetti, Marco Verdecchia, and Guido Visconti. Hydrological modelling and the water cycle: coupling the atmospheric and hydrological models, volume 63. Springer Science & Business Media, 2008.

[26] Elom Foli Stefan Korenberg, Arjen Roelandse. Dynamische gt kaart wdod. Acacia, 2022.

[27] Department of Economic United Nations and Population Division Social Affairs. World Population Prospects 2019: Highlights. United Nations, 2019.

[28] Remco van de Beek. Rainfall variability in the Netherlands from radars, rain gauges, and disdrometers. Wageningen University and Research, 2013.

[29] J.W.J. van der Gaast, H.T.L. Massop, and H.R.J. Vroon. Effecten van klimaatverandering op de watervraag in de Nederlandse groene ruimte : analyse van de waterbeschikbaarheid rekeninghoudend met de freatische grondwaterstand. Number 1791 in Alterra-rapport. Alterra, 2009.

[30] WDODelta Waterschap. Peilgebieden, May 2022. URL `https://open-data-portaal-2-wdodelta.hub.arcgis.com/datasets/WDODelta::peilgebieden/about`.