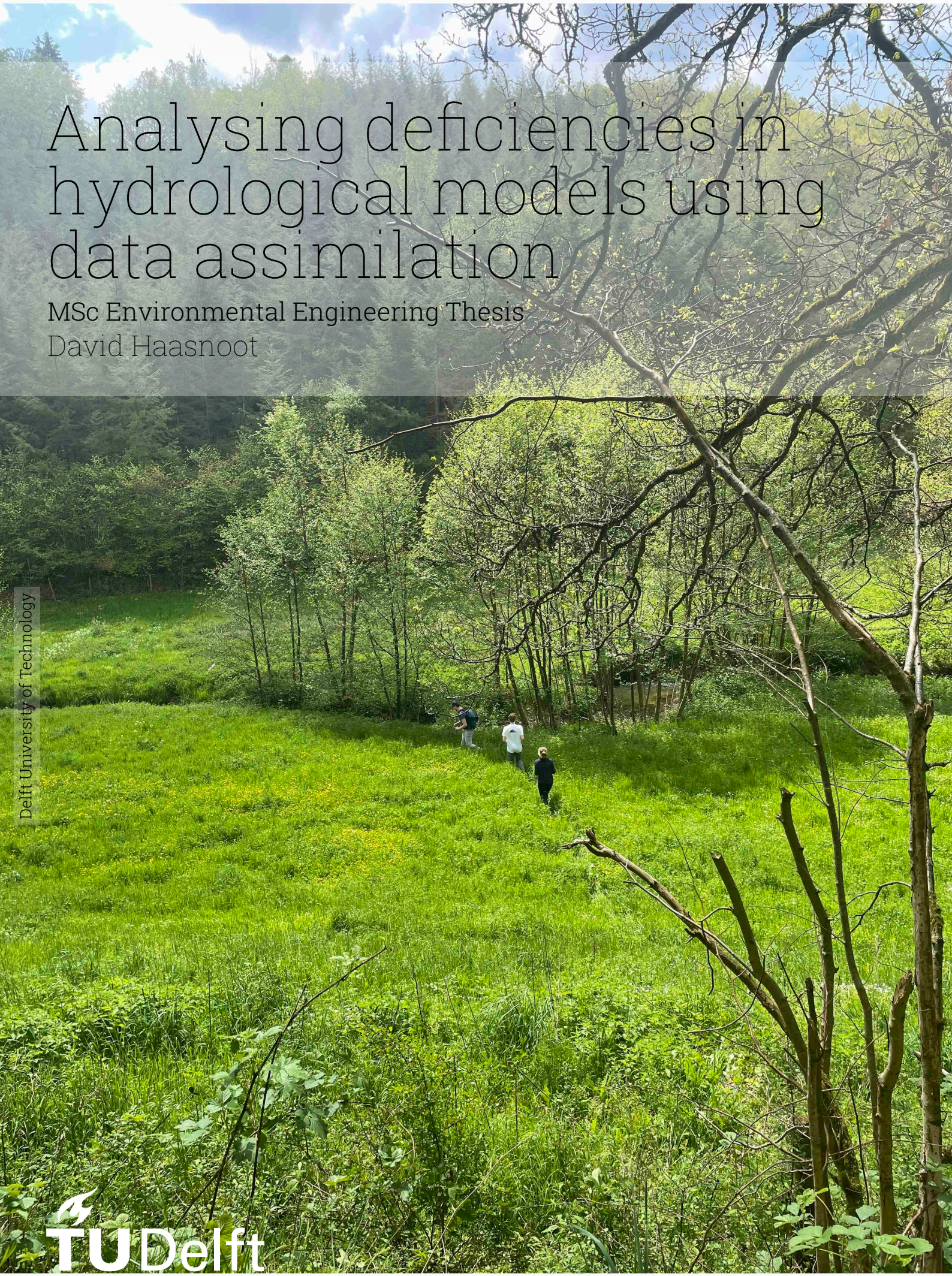
# Analysing deficiencies in hydrological models using data assimilation

MSc Environmental Engineering Thesis

David Haasnoot



Delft University of Technology

TU Delft

# Analysing deficiencies in hydrological models using data assimilation

by

David Haasnoot

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday 11$^{th}$ July, 2024 at 14:30.

| | | |
|---|---|---|
| Student number: | 4897900 | |
| Project duration: | February 5$^{th}$, 2024 – July 11$^{th}$, 2024 | |
| Thesis committee: | Dr. ir. R. W. Hut, | TU Delft |
| | Dr. ir. F. C. Vossepoel, | TU Delft |

Cover: Field trip to Luxembourg, Mëchelbach catchment

An electronic version of this thesis is available at repository.tudelft.nl.

TU Delft

# Contents

# Summary

Hydrological modeling is used to estimate the states and fluxes in a system given inputs. This allows us to gain an understanding of the water system and make predictions such as flooding or drought. Models use parameters to describe a general process such as infiltration. These parameters can be measured, inferred or calibrated. Parameters represent local aspects such as infiltration into the soil and can differ for a catchment in rural Ohio versus mountainous Colorado. Even with the best local parameters, models will always fall short of reality since they contain simplifications. Data assimilation allows the uncertainty in the prediction of stream flow to be reduced or quantified. This thesis aims to answer the research question 'What do the adjustments in parameters and states imposed by data assimilation say about the deficiencies in the hydrological models of observed streamflow?'.

In this thesis, a data assimilation framework was developed for the eWaterCycle platform. This framework takes care of the intricacies of data assimilation for the user. Any model implemented with the Basic Model Interface can make use of this, models within eWaterCycle are focused on FAIR hydrology. The framework was implemented in such a way that any data assimilation scheme can be used, following the FAIR principles. The framework was verified to work with synthetic observations for both the HBV (Hydrologiska Byråns Vattenbalansavdelning) and the Lorenz model. The framework also makes applying data assimilation within eWaterCycle easier and scalable.

To answer the research question, the five-year hydrological response of 671 catchments in the United States of America was modeled. These catchments are from the CAMELS dataset, which is a collection of hydrological data with characteristics for catchments across the United States of America. The data assimilation scheme particle filtering was applied to the conceptual hydrological model HBV using eWaterCycle. The parameters and states of each experiment were stored and analysed. The data assimilation scheme requires hyperparameters. These hyperparameters were optimised using the first 26 catchments in the dataset. This single combination of hyperparameters was used across the 671 catchments.

Taking the best prediction of stream flow at every time step, the data assimilation experiment outperformed the calibrated model in 461 of these catchments. Part of this is due to the bias introduced through the use of one set of hyperparameters. Analysis shows that catchments of the same characteristics perform similarly. Catchments with a low mean stream flow, have a high spread of predicted observations causing very good predictions when selecting the best. Due to this bias, no real correlations can be drawn about the relation of background characteristics and model deficiency. Analysing three catchments on a catchment scale showed that step changes in parameters often occur around flood peaks. The data assimilation scheme adjusts the working of the model to better capture the observed streamflow.

To further incorporate the framework in eWaterCycle more testing should be done on 2D models. Deficiencies in the HBV model can be further analysed by optimising hyperparameters per catchment type or flow regime.

# Glossary

## Terms

| Term | Definition |
|---|---|
| Ensemble | Distribution of model simulations |
| CAMELS | A dataset containing Catchment Attributes and Meteorology for Large-sample Studies |
| FAIR | Findable Accessible Interoperable Reproducible |
| Fluxes | Amount of water moving between two storage units |
| HBV | Hydrologiska Byråns Vattenbalansavdelning - a conceptual hydrological model |
| Jitter | Small variation added to the state vector after resampling |
| Model | A mathematical representation of reality. |
| Model state | Quantities which fully describe a model at a given time |
| NSE | Nash-Sutchcliff Efficiency - measure of goodness of fit |
| State vector | Uncertain part of the model state |

## Symbols

| Symbol | Definition | Unit |
|---|---|---|
| $C_e$ | HBV model parameter used to describe what factor actually evaporates from the ground | - |
| β | A factor controlling the split between fast and slow flow in the HBV model | - |
| $\mathbf{d}$ | Vector containing observations/data | Various |
| $\mathbf{f(d\|z)}$ | Likelihood of observations given the state vector | - |
| $\mathbf{F}$ | Vector of forcing values (inputs) | Various |
| $FM$ | Amount of snow melt per day per degree in the HBV model | mm/d/°C |
| $f_n$ | Fraction of ensemble members which remain without resampling taking place | - |
| $I_{max}$ | Maximum amount of interception in the HBV model | mm |
| $k$ | Size of the assimilation window | days |
| $K_f$ | Fraction of reservoir storage flowing out of the fast flow in the HBV model | - |
| $K_s$ | Fraction of reservoir storage flowing out of the slow flow in the HBV model | - |
| $\mathbf{M}$ | Model | NA |
| $N$ | Number of ensemble members | - |
| $\theta$ | Vector of parameter values | Various |
| $P_{max}$ | Maximum amount of percolation which can occur from the ground to the deeper ground water flow in the HBV model | mm |
| $S_f$ | Fast flow (run-off) storage in the HBV model | mm |
| $S_i$ | Interception storage in the HBV model | mm |
| $S_p$ | Snow Pack storage in the HBV model | mm |
| $S_s$ | Slow flow (groundwater) storage in the HBV model | mm |
| $S_u$ | Unsaturated root zone (upper soil) storage in the HBV model | mm |

| Symbol | Definition | Unit |
|---|---|---|
| $Su_{max}$ | Maximum size of the reservoir of the unsaturated root zone | mm |
| $\sigma_p$ | Standard deviation of the noise distribution from which the perturbation is sampled | - |
| $\sigma_w$ | Standard deviation of the weight distribution used to generate a likelihood distribution | - |
| $T_{lag}$ | Lag time between water falling and it reaching the river in the HBV model | d |
| $w_j$ | Likelihood weights | - |
| $\mathbf{x}$ | Model state | Various |
| f($\mathbf{y}$) | Distribution of arbitrary vector $\mathbf{y}$ | Various |
| $\mathbf{z}$ | State vector | Various |
| f($\mathbf{z}$\|$\mathbf{d}$) | Posterior distribution | - |

# Preface

This thesis was written to finalize my environmental engineering master program at Delft University of Technology. In our ever advancing world, more FAIR tools can help ensure hydrological modeling continues to fully capture the changing hydrological systems. The data assimilation framework presented in this thesis is just one such example.

An introduction to the concepts of hydrology and data assimilation are given in chapter 1 and should be accessible to all readers. These concepts are further explored in chapter 2, which is aimed at readers more interested in the technicalities. The results and conclusions are explored in chapter 3 and 4 respectively. The appendices contain further technical explanations about the methods used, including links to the code developed as part of this thesis.

I would like to thank my supervisors Dr. ir. R.W. Hut and Dr. ir. F.C. Vossepoel for their advice, guidance and interesting discussion throughout this thesis. Thanks to the eWaterCycle team, in particular Dr. ir. B Schilperoort, for their patience, advice and for letting me join workshops to improve my coding skills. Thank you to all those who took time to proof read this thesis for spelling and grammar errors. And finally, a thank you to all the friends and family who have supported me throughout my past few years of studying here at Delft University of Technology.

No generative artificial intelligence tools were used in writing or coding of this thesis.

*David Haasnoot*
*Delft, July 2024*

# 1 Introduction

Hydrological modelling is a field of science focused on understanding and quantifying the states and fluxes of a hydrological system given inputs (precipitation, evaporation, temperature). This allows us to gain an understanding of the processes contributing to the movement of water through a hydrological system. Another use is making predictions about the implications this has on society. These implications range from flooding and drought to food production and climate change (Wood et al., 2011). When the scale of hydrological models increases to the size of a region or country, directly measuring parameters such as permeability of the soil or rainfall interception becomes more difficult. Instead, 'effective' parameters can be used to represent the processes in a given area in the model. The represented area can be a whole catchment in more conceptual models or a grid square in so-called distributed models. These parameters are traditionally inferred based on remote sensing data, derived from other sources such as land use maps or they can be calibrated based on streamflow observations (Kirchner, 2006). The Leaf Area Index, soil type/moisture and snow cover area are examples of physical quantities which can be sensed remotely and incorporated into hydrological models (Ali et al., 2023).

No model is perfect and every model will miss some aspect of the true hydrological response of a system. Through the use of data assimilation, the uncertainty in the predicted streamflow can be quantified and/or reduced. Data assimilation uses observations to adjust the parameters and states of a model for the current time step. By running a distribution of model simulations, a collection of model results is obtained. The collection of models is used, together with observations, through a data assimilation scheme to obtain the probability distribution of the quantities of interest and update the parameters and states through time. The uncertainty of the combination of model and data is typically reduced compared to the uncertainty of the model only. Data assimilation has already been used for different applications in the hydrology field with the focus mainly on improving forecasting (Liu et al., 2012). Assessing model deficiencies using data assimilation in hydrology is not explored much in the existing literature.

The goal of this thesis is to use data assimilation in a hydrological model to see what the adjustments in parameters and states say about the deficiencies in the hydrological model itself when compared to observed streamflow. To achieve the research goal, many model simulations are used across different catchments. The catchments used are part of the CAMELS (Catchment Attributes and Meteorology for Large-sample Studies) dataset, a collection with prepared hydrological modelling data for catchments across the contiguous United States of America. Using the streamflow simulations, parameters and states will be observed over time. If a parameter is constant, this implies the optimal parameter is found and thus the model can accurately describe that part of the system. If the data assimilation scheme adjusts parameters this can show model deficiencies. The variations in parameters and states are compared to a calibrated model simulation which does not utilise data assimilation. To apply data assimilation to a wide range of catchments in an easy-to-use and scalable way, a data assimilation framework will be developed using the eWaterCycle platform.

The eWaterCycle platform enables hydrological modelling to be Findable, Accessible, Interoperable and Repeatable (FAIR) (Hut et al., 2022). The eWaterCycle platform is hosted on GitHub, making it findable and accessible. The different models and data sources are compatible, making different components of the platform interoperable. The data sources and code structure of the platform make the modelling carried out repeatable. This developed data assimilation framework has been built around the existing eWaterCycle platform, adhering to these FAIR principles. This means all of the code developed as a part of this thesis is also accessible on GitHub. An added benefit of this approach is that future users will also be able to easily apply data assimilation to their models or use cases of choice

using the framework developed. It also allows for a repetition of the methods presented in this thesis with another model or set of models to further analyse more deficiencies.

## 1.1. Data assimilation

Data assimilation is the combination of prior information from model simulations with observations to estimate the posterior distribution of the model state and parameters (Evensen et al., 2022). The field of ensemble data assimilation focuses on a collection of prior model simulations, known as an ensemble. The ensemble contains a set of ensemble members. We assume these ensemble members can each be fully explained by the states of the model, the parameters and the inputs (or forcing) applied. This can mathematically be expressed by equation 1.1, where $\mathbf{x_k}$ is the state of the model ($\mathbf{M}$) after $k$ time steps, $\theta$ is a vector of the model parameters and $\mathbf{F}$ the forcing on the model.

$$\mathbf{x}_k = \mathbf{M}\{\mathbf{x}_{k-1}, \theta, \mathbf{F}\} \tag{1.1}$$

Together, the states, parameters and forcing provide all the information needed to reproduce a model at a given time. The states and parameters of the prior model simulations can be collected in the state vector. Only the uncertain quantities are of interest, known quantities are not collected. Mathematically this means $\mathbf{z} = \{\mathbf{x}, \theta\}$. An explanation of the state vector used for the experiments can be found in section 2.4. The prediction is the part of the state vector which can also be observed. This is either part of the state vector or a derivative of it. This can be mathematically expressed as $\mathbf{d} = \mathbf{H}(\mathbf{x}) + \varepsilon$. Where $\mathbf{H}$ is known as the measurement operator and $\varepsilon$ is the measurement error. To move from the prior to the posterior distribution, data assimilation is based on Bayes' Theorem given in equation 1.2. The state vector $\mathbf{z}$ is the uncertain part of the model state. $\mathbf{d}$ is the vector of observations (or data). $f(\mathbf{z}|\mathbf{d})$ is the posterior distribution. $f(\mathbf{z})$ the prior distribution and $f(\mathbf{d}|\mathbf{z})$ is the likelihood of the observations given known state vector (Evensen et al., 2022).

$$f(\mathbf{z}|\mathbf{d}) = \frac{f(\mathbf{d}|\mathbf{z})f(\mathbf{z})}{f(\mathbf{d})} \tag{1.2}$$

## 1.2. Hydrological modelling

The eWaterCycle platform aims to make modelling easier for hydrologists by removing technical challenges. Using the Basic modelling Interface (Hutton et al., 2020), different hydrological models can all be accessed through eWaterCycle. This allows users to easily use a variety of models and forcing. Forcing is the data that the model takes as input. Forcing often includes, but is not limited to: precipitation, temperature and evaporation. Models and forcing sources vary across different use cases. The implementation of eWaterCycle makes models and forcing interchangeable. Interchangeability allows users to contribute their own work, or apply existing forcing and models to their own work. This is achieved using grpc4bmi, a software wrapper that allows communication between models written in different programming languages which have been implemented using the BMI standard (van den Oord et al., 2023). This thesis will use the eWaterCycle platform as a starting point, on top of which a data assimilation framework has been developed. This allows other hydrologists to use data assimilation in the future.

The conceptual model used in this thesis is a version of the HBV (Hydrologiska Byråns Vattenbalansavdelning) bucket model. A collection of HBV models was first developed in Sweden in the 1970s (Seibert and Bergström, 2022). Conceptual models, also known as bucket models, simplify the catchment to just a handful of states and parameters. This original collection contained four versions, the first of which was very simple and each increased in complexity to the more complex the fourth version. The implementation used in this thesis most resembles the HBV-3 original model by Bergström (1976), with some adjustments made to better suit the use case. With just five buckets modelling snow, interception, the root zone, overland flow and groundwater, the HBV model performs quite well considering its simplicity. The fluxes or flows between these buckets are governed by nine parameters. The details of HBV can be found in section 2.4. The HBV-96 model as described by Lindström et al. is a more modern version but adds more input data and contains 12 parameters (1997). To reduce the complexity, the number of parameters and states is intentionally kept at 9 in this research, whilst still reflecting the processes in the HBV(96) model.

## 1.3. Examples of data assimilation in hydrology from literature

The following examples explain what has already been done when combining data assimilation and hydrology and show how the chosen approach for this thesis is different. Data assimilation can be used in several types of models as explored by Liu et al., such as groundwater, sediment transport, biogeochemical and hydrological models (2012). Within hydrological models, data assimilation can be applied to different parts of the model structure. One of the model aspects that can be assimilated is the states (e.g. soil moisture storage), these can be updated based on the observation at a given time step (Kim et al., 2021). The assimilation of soil moisture for example can be used to improve flood forecasting (Aubert et al., 2003, Seo et al., 2009). Data assimilation can also be used to improve the internal fluxes of a model. Rakovec et al. (2012) uses EnKF for assimilating the interior gauges to improve final outflow predictions. Whilst examples of assimilating the model fluxes or states can be found in literature, parameters are often left unchanged. They are often optimised using least-square-based techniques (Kirchner, 2006, Bárdossy, 2007). Vrugt et al. (2005, 2013) combine optimization with data assimilation as a tool for parameter estimation. Vrugt et al. (2005, 2013) only explore the effect of parameter changes, but does not make the link to what the changes in parameters and states say about the deficiencies in the model itself. There are, to my knowledge, no examples of changes in parameters and states being used to analyse model deficiency within the field of hydrology in other literature.

## 1.4. This thesis

Although the FAIR principles are being recognised within hydrological modelling (Cudennec et al., 2020, Hall et al., 2022), there appear to be no examples of these principles being implemented in data assimilation in hydrology. To address the research questions, I have first built a framework on top of the eWaterCycle platform which allows data assimilation to be applied within hydrology. The framework was designed in such a way to work with any model, data assimilation scheme or use case. There are limitations to this, which are explored in section 2.1. As a first scheme, particle filtering is implemented and extensively tested. Adding more data assimilation schemes to the framework can be done easily. During the design process, the requirements of the most commonly used schemes have been taken into account. Chapter 2 presents the methodology and consists of two major parts. Sections 2.1 - 2.3 detail the development of the data assimilation framework, applying it using particle filters and testing this framework with the Lorenz-96 model (Lorenz, 1995). Sections 2.4 - 2.5 explore the experiments carried out. These experiments apply data assimilation to the conceptual hydrological model HBV and compare it to a baseline model which is calibrated using a classical approach. This allows for the analysis of deficiencies in the model. Chapter 3 outlines the results from the tests and experiments. It then explores what these experiments tell us about the deficiencies in the hydrological model. Finally, Chapter 4 presents the conclusions we can draw from the results and discusses them. In appendix A a more detailed flowchart containing the sub-goals of the research is shown. Appendix B presents more detailed diagrams outlining the design process of the data assimilation framework. Appendix C contains more results which can be interesting to consider for readers looking into the relation of data assimilation to catchment characteristics. The figures also further substantiate the general analysis described in the chapter. Appendix D contains a set of 5 notebooks showcasing some of the work done in this thesis, including links to where the rest of the coding done in this thesis can be found.

# 2
# Methodology

This chapter describes the methods used to answer the research question 'What do the adjustments in parameters and states imposed by data assimilation say about the deficiencies in the hydrological models of observed streamflow?'. Firstly, in sections 2.1 - 2.3 the implementation of the data assimilation framework is outlined. Secondly, in sections 2.4 - 2.5 the experiments themselves are presented: running the data assimilation for different catchments.

## 2.1. Implementation of data analysis framework

The purpose of the implementation of a data assimilation framework in eWaterCycle is twofold. Firstly, making the implementation of data assimilation easier for the user. Secondly, making it faster to compute an ensemble run in eWaterCycle. Currently data assimilation can be implemented by an experienced user of the eWaterCycle platform. All pieces exist, however, they are not easily connected and require technical knowledge of both data assimilation and eWaterCycle. This thesis describes a framework which I have built that all users can apply within eWaterCycle to more easily set up data assimilation for hydrological models.

#### Ease of use

A future user of the data assimilation framework is assumed to have the knowledge to already interact with a normal eWaterCycle model. For them to add data assimilation, they will follow a similar set of steps as a model run without data assimilation. The data assimilation scheme takes care of the intricacies of data assimilation in the background. Through the standardisation and integration with eWaterCycle, changes to simulations are easily made, allowing for rapid scalability across models, catchments or use cases. This makes it easier to run data assimilation for many catchments and models. At the same time, use cases can vary strongly, making flexibility and adjustability key. The framework was implemented in such a way that other types of data assimilation schemes can be implemented by other developers in the future, adhering to the FAIR principles.

#### Computational speed

As described, a key part of implementing ensemble data assimilation is many model runs. For larger models, longer time periods and/or large ensemble size, this increases computational times drastically. The different models can be run in parallel rather than sequentially, speeding up overall computation times. The framework was implemented in such a way that it can be scaled to different machines. This would mean every ensemble member runs on its own machine or node. As an instance of eWaterCycle is already deployed by Delft University of Technology and the eScienceCenter in the SURF research cloud (SURF, 2023), scaling up to their cloud infrastructure is attainable. The framework was designed to ensure this is possible in the future, but the implementation of this is left for future work.

### 2.1.1. Requirements from data assimilation schemes

To develop a framework in which different types of data assimilation schemes can be run, their varying requirements are important to take into account in the design process. Evensen et al. (2022) give an overall view of the different schemes. There are many different types of data assimilation schemes. Implementing all these algorithms was not the main goal, but ensuring they can theoretically all be implemented by a future user without a redesign of the system was. The commonly used ensemble based data assimilation algorithms included in the considerations are:

- Ensemble Kalman Filter (EnKF)
- Ensemble Smoother (ES)

- Particle Filter (PF)
- Ensemble Smoother Multiple Data Assimilation (ES-MDA)
- Iterative Ensemble Smoother (IES)

This includes both sequential methods (filters) and smoothers as they have different use cases. There are some exceptions, in particular the '4DVar' scheme, which is not included in the design process. 4DVar is a variational method that requires the use of an adjoint ("backward") model. This adjoint moves backwards in time, which greatly increases the complexity and is highly model-specific.

The first two of the mentioned algorithms (EnKF & ES) make use of the error covariance of the ensemble to efficiently compute the posterior distribution by minimizing a cost function, requiring a smaller ensemble size. This is based on the Gaussian approximation, which is not suitable for highly nonlinear systems. In case of high non-linearity particle filtering is used, which is explained further in 2.2. All three (EnKF, ES & PF) require the prior state vectors of the ensemble at the given timestep to return a set of posterior state vectors. The latter two algorithms (ES-MDA, IES) use an extra iteration step. ES-MDA essentially reruns (*Multiple* DA) a data assimilation scheme and combines this with a smoother function to improve the performance of the model. IES also minimises a cost function, like ES, however, does this using a Gauss-Newton iterations scheme (Evensen et al., 2019). Due to the iterative nature, these two algorithms (ES-MDA, IES) require all the state vectors of the ensemble over time. Although the algorithms are not explained in depth here, the main outcome is the different requirements they each have. Flow charts have been developed outlining the processes for each algorithm, which can be found in appendix B.

### 2.1.2. User requirements

User requirements are also taken into account when designing the framework. These requirements follow from the eWaterCycle design philosophy and from discussions with maintainers of the platform. These requirements can be summarised as follows:

- Intuitive: the interface should be predictable and understandable. A bachelor's student with some background in earth sciences should be able to apply the framework.
- Improvable: users should be able to easily add their own data assimilation scheme. This should be such that this is possible with bachelor-level Python programming skills.
- Flexible: users should be able to use the scheme for various use cases. The parameter estimation use case presented in this thesis is just one example.
- Compatible: existing hydrological models which are compatible with eWaterCycle should be easily incorporated with minimal changes.
- Scalable: parts of the scheme should be able to be hosted elsewhere to reduce computation times.

The following assumption was made to more clearly define the scope of the project:

- The state vector is assumed to be small. Small is taken to mean less than 100 items.

For more complex models this assumption does not hold, but the focus remains on simpler models. To deal with larger models, only a part of the state vector should be sent back and forth. Operations on vectors larger than 100 items start to increase in computational complexity. Implementing a decentralised option where only part of the state vector is exchanged makes the framework less flexible and more complex.

### 2.1.3. Design of data assimilation framework

Using the requirements outlined above, two designs were created. These are differentiated by a centralised and a decentralised approach. The centralised approach is given in figure 2.1. The decentralised approach can be found in the appendix B.2.

The centralised approach is more suited for algorithms, such as ES-MDA and IES, which involve an iteration step (see section 2.1.1). When iterating, a requirement is that all the state vectors over time need to be accessible to the data assimilation scheme. When updating the state vector sequentially

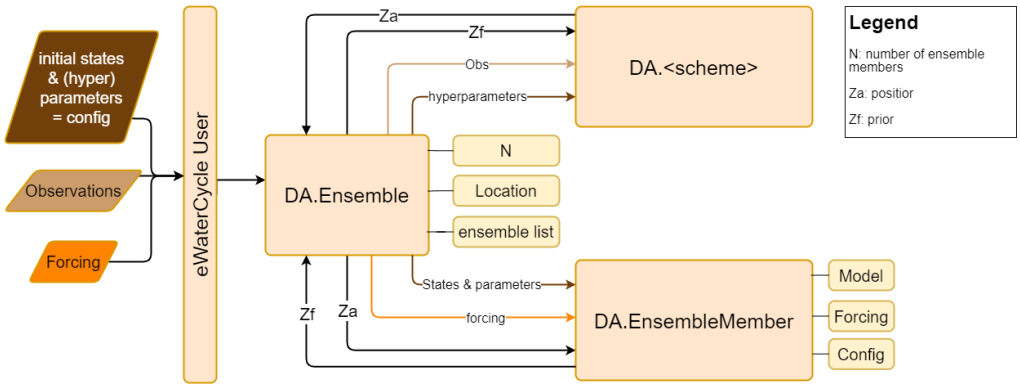



**Figure 2.1:** Flow chart showing the centralised design for data assimilation framework. Arrows show the flow between different components of the framework.

to the next state (PF, EnKF), only the previous state is required to be accessible. If only the previous step is needed, a more decentralised approach can be sufficient. The main benefit of the decentralised option is that the state vector is exchanged less, requiring a lower computational cost. In more complex or distributed models this state vector can be very large. The centralised design does require the state vector to be exchanged, however increases flexibility.

The function handling the assimilation part of the code is implemented as a separate class in its own file. Without changing any of the main code running the ensemble computation, a user can add their own implementation of a data assimilation scheme. The only requirement is that the state vector is returned to the ensemble in the correct format for sequential schemes. Smoothers can use this functionality to execute the smoothing step. In the decentralised approach, the scheme is much more integrated with the running of the ensemble itself. This adds performance benefits as the state vector can be more optimally managed for the given scheme, but adds complexity for the user and requires more knowledge of the system.

For the current use case of a data assimilation framework within eWaterCycle, the centralised approach is more fitted due to it being applicable to a wider set of use cases and it is easier to add more data assimilation schemes.

### 2.1.4. Coding the framework

The flowchart as shown in figure 2.1 is still at a fairly high level. The framework was coded as an independent Python package which is dependent on eWaterCycle. This choice was mainly to make development faster and because it is only dependent on eWaterCycle, but requires no real changes to the eWaterCycle platform itself. In future, it could be easily added as an (optional) utility. Translating this framework into Python code yields a more detailed plan as shown in figure 2.2. The flowchart shows a linear approach, which is how the framework is used for the experiments in this thesis. The experiment is thought out before running the framework. All the settings for data assimilation need to be supplied to the framework at the start. These settings include which data assimilation scheme is used, accompanying hyperparameters, observations and the quantities of the model state which are included in the state vector.

In another use case, the settings for data assimilation can also be supplied just before assimilating. This 'on the fly' implementation allows the model to be run normally up to a point, then data assimilation can be added in. This allows for more flexibility in how the framework can be used. One example use case would be if extra data is gathered before or during a period of flooding. Historic data can be used to run the model up to the flooding event without assimilating, the newly gathered observation can then be supplied to the algorithm on the fly to improve the flood predictions. A flowchart showing the steps when using the framework 'on the fly' is shown in figure 2.3.

Another added benefit is that the framework can be used as a tool to run model ensembles without using the data assimilation scheme. The assimilation step is skipped and the model runs as it would normally, but utilising the parallel data structure and the convenient methods of retrieving and setting

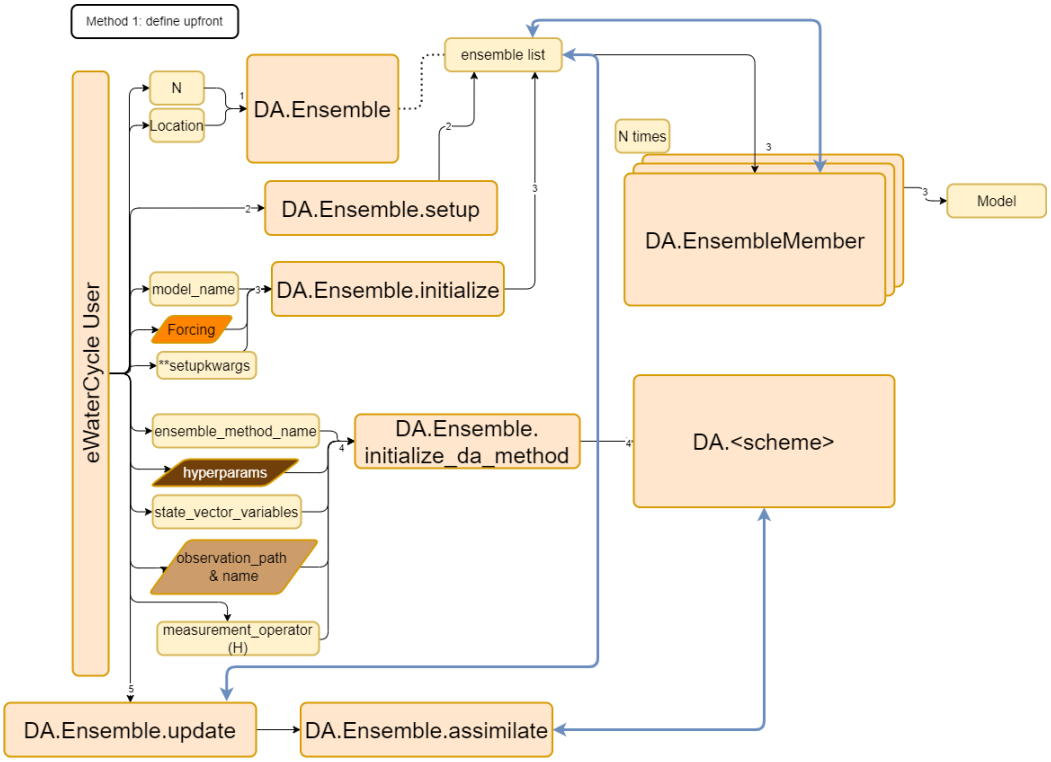



**Figure 2.2:** Flowchart showing a linear use of the framework, here all the data assimilation method and hyperparameters are defined before running the model. The small numbers next to the arrows show the order in which the functions are called.

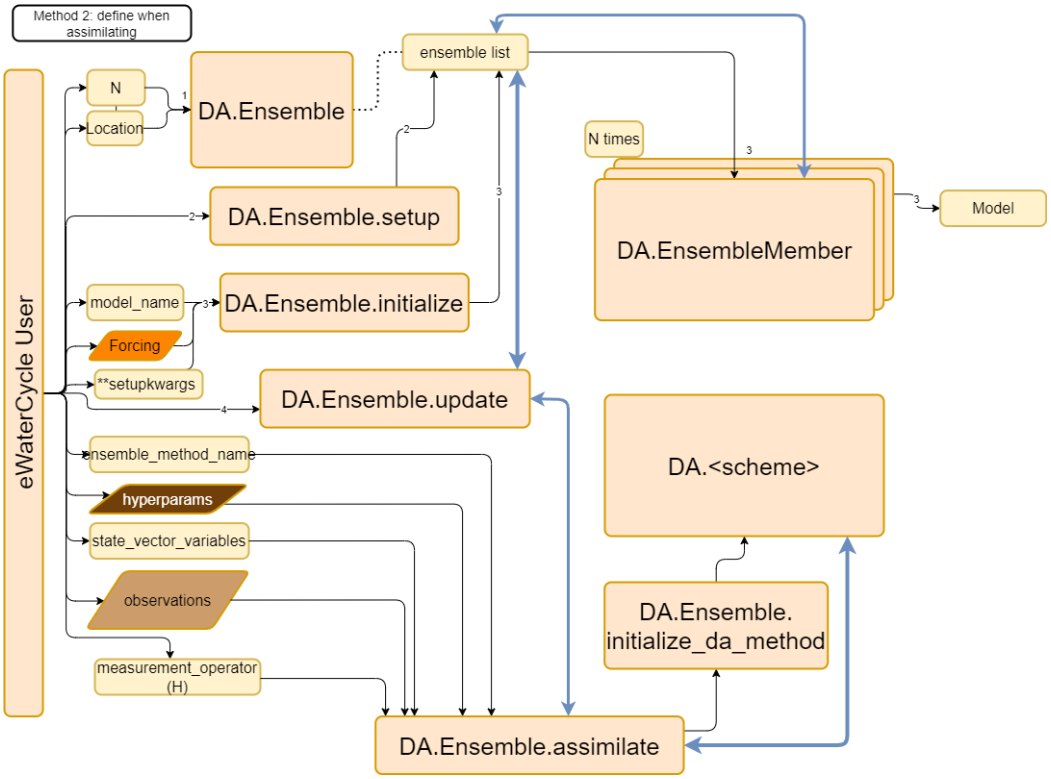



**Figure 2.3:** Flowchart showing an 'on the fly' use of the framework, here the data assimilation hyperparameters and method can be defined at any point during the model run. The small numbers next to the arrows show the order in which the functions are called.

parameters and states for a large number of models. Example use cases of 'on the fly' data assimilation include running hydrological models for different climate scenarios, sensitivity analysis and more classical calibration of model parameters. In this thesis, this is also used to calibrate the reference model run.

The appendix contains three Jupyter notebooks showcasing the framework being applied in different scenarios. Firstly, defining the data assimilation parameters up front in section D.1.1; secondly defining the parameters on the fly in section D.1.2; and lastly an example of a traditional calibration run in section D.1.3.

## 2.2. Application of the framework using particle filters

Particle filtering is used to demonstrate the functionality of the framework. Using the same formulation as in 1.1, equation 2.1 shows the workings of the particle filter. The data assimilation scheme is represented as $\mathbf{M}_{\text{DA}}$.

$$(\mathbf{z}_k, \theta_k) = \mathbf{M}_{\text{DA}}\{\mathbf{z}_{k-1}, \theta_{k-1}, \mathbf{F}\} \tag{2.1}$$

Particle filtering is the only data assimilation scheme currently implemented, but the framework was designed to work with any data assimilation scheme. Particle filtering, specifically Sequential Importance Resampling, was implemented according to the following steps. First, the model is run $N$ times for slightly different parameters and states. The results of the ensemble predictions are compared to the observable ($\mathbf{d}$) at the end of an assimilation window. For particle filters, the assumption is made that the prior probability density function ($f(\mathbf{z})$) can be sampled by a finite number of samples as given in equation 2.2a, where $\delta$ is the Dirac Delta function. Substituting this into Bayes' Theorem (equation 1.2), gives equation 2.2b for the posterior probability density function. (Evensen et al., 2022).

$$f(\mathbf{z}) \approx \sum_{j=1}^{N} \frac{1}{N} \delta(\mathbf{z} - \mathbf{z}_j) \quad \text{(a)} \qquad f(z|d) \approx \sum_{j=1}^{N} w_j \delta(z - z_j) \quad \text{(b)} \tag{2.2}$$

From substitution into Bayes' theorem, the likelihood weights are given by equation 2.3.

$$w_j = \frac{f(\mathbf{d}|\mathbf{z}_j)}{\sum_{i=0}^{N} f(\mathbf{d}|\mathbf{z}_j)} \tag{2.3}$$

The likelihood weights are computed for each ensemble member, where a particle with a modelled value closer to the observation receives a higher weight than one further away. In the implementation of particle filters with importance resampling the number of particles is kept constant. Depending on the normalised likelihood weights ($w_i$) a choice is made to resample or not. If the ensemble as a whole is too far away from the observed value, only a few particles contribute and a resampling step is needed. Whether to resample can be based on the criterion of effective ensemble size ($N_{eff}$) as described by Kuptametee and Aunsri (2022), given in equation 2.4. If $N_{eff}$ is lower than $f_n * N$, resampling takes place. $f_n$ is a hyperparameter value, which can be adjusted depending on the use case. A value of $f_n$ between 0.5 and 1 is often used depending on the use case (Varsi et al., 2020).

$$N_{eff} = \frac{1}{\Sigma_{i=0}^{N} w_i^2} \tag{2.4}$$

To resample, the state vectors of particles with low weights are replaced. This new set of state vectors is sampled from the prior ensemble. Particles with a high weight are more likely to be resampled. This is illustrated in figure 2.4 by Van Leeuwen et al. (2015). To generate the weights using equation 2.3, a zero-mean normal distribution is used to determine how the weights are distributed. The standard deviation of this distribution is a hyperparameter ($\sigma_w$) supplied by the user. In the hydrological use cases presented, a log-normal distribution is used because river discharge can physically never be negative. To ensure the ensemble is varied enough, the new state vector is perturbed slightly. This

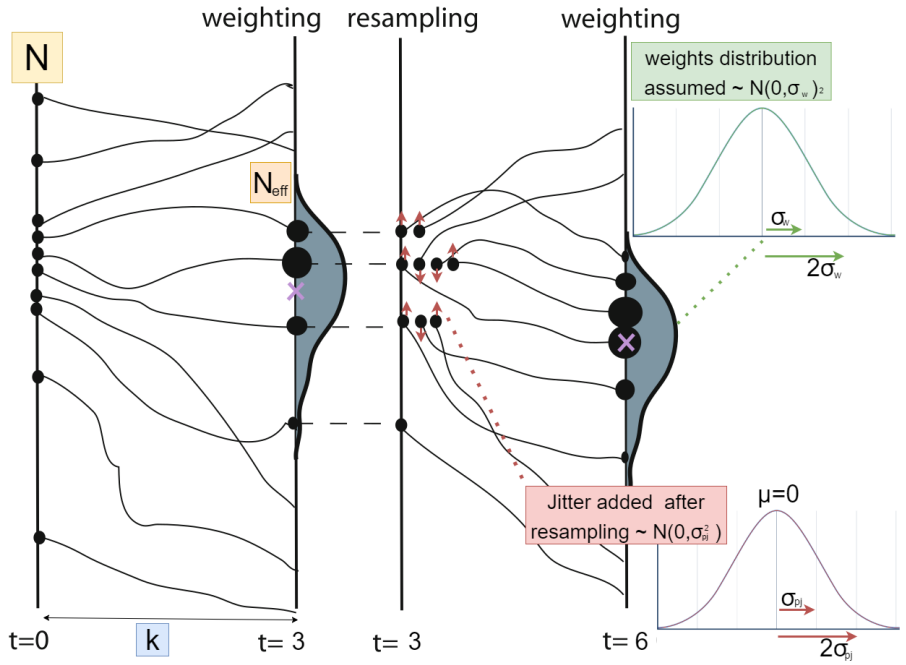



**Figure 2.4:** Explanation of the function of hyperparameters, figure from Van Leeuwen et al.(2015), adapted by adding annotations on $N$,$N_{eff}$, $k$, $\sigma_{p_j}$ and $\sigma_w$. With the observation shown as a purple cross, the jitter as the red arrows which slightly perturb the particles sampled from a zero-mean normal distribution and the zero-mean normal likelihood distribution in green used to calculate the weights for each particle.

is also known as jitter. This variation in the particles ensures that the ensemble can always capture the true model well. The variance of the jitter is a hyperparameter ($\sigma_p$). Assimilation, in particular the calculation of weights, occurs every few time steps, known as an assimilation window. The size of this assimilation window is a parameter varying per use case. Whether resampling takes place after an assimilation window depends on the mentioned criterion.

### 2.2.1. Particle filter hyperparameters

Hyperparameters are needed to tune the behaviour of the data assimilation scheme. Ideally, one universal set of hyperparameters would exist, but due to varying use cases, these are often adjusted in order to ensure the data assimilation scheme functions as desired. All data assimilation schemes use Bayes' theorem as a foundation (Evensen et al., 2022). Assumptions are then applied to allow for the use of the theorem for different use cases. For particle filtering the assumption is that likelihood and noise follow a given distribution which has a mean of zero. The scale or variance of this zero-mean normal distribution is a hyperparameter. In the implementation of particle filters used, there are five hyperparameters:

- The size of the ensemble (N).
- The scale of the distribution used to estimate the particle weights/likelihood ($\sigma_w$).
- The scale of the zero-mean Gaussian distribution used to add noise to the state vector ($\sigma_p$).
- The number of steps $k$ in an assimilation window.
- The factor $f_n$ controlling the resampling threshold: resample if $N_{eff}$ is lower than $f_n * N$.

These five hyperparameters are also shown in figure 2.4. The figure shows a general case, where a standard normal distribution is used. In hydrology, the distribution is skewed as discharge can not be negative. To account for the skew, a log-normal distribution was used. The principles shown in figure 2.4 remain the same. How the likelihood estimation and the perturbation are implemented is explained in section 2.2. The hyperparameters N, k and $\sigma_w$ are, by definition, single values. Ideally, the size of the ensemble (N) is very large to fully capture the distribution of the particles. This is however limited by computation power and time available. The width of the assimilation window, $k$, should be chosen

realistically. If $k$ is one timestep, the weight function is computed every time step, which is too often and too expensive. If the value of $k$ is too large, the model spreads out too far and the assimilation can lose its meaning. $f_n$ was initially set to 0.8, as suggested by Evensen et al. (2022). The value of $\sigma_w^2$ should match the variance of the noise in the observations. In the synthetic runs, normal noise with a normal distribution is added and thus $\sigma_w$ can easily be matched to the chosen value. In real observations, the variance of the noise is unknown, thus choosing $\sigma_w$ is difficult. To establish a good initial estimate, the scheme is run several times, each run using a different value of $\sigma_w$. With trial and error runs these can then be tuned.

The amount of jitter added to the state vector (controlled by $\sigma_p$) can be the same for all values of the state vector, or it can vary. Varying the amount of jitter added can also add bias to a given quantity if it is varied more than the rest. If the quantities are physically different, an absolute amount should be used to ensure that the smaller quantities are not perturbed more than the larger quantities. Further explanation of the method used to apply jitter can be found in section 2.4.4.

## 2.3. Testing the framework on a Lorenz-96 model

Hydrological models can be difficult to troubleshoot and test that the behaviour is as expected. In the field of data assimilation, the Lorenz-96 model is often used to test implementations of data assimilation, for example by Vrugt et al. (2005). The model is known for simulating chaotic behaviour, small perturbations in the model lead to large changes in the results later in time. This Lorenz-96 model was implemented as one of the available models in eWaterCycle to test the performance of the framework. It is given by equation 2.5, for $i...N$, where $N$ is the number of dimensions of the model. This parameter is chosen by the user. The state vector of the system is given as $x_i$. To compute $x(t + \Delta t)$, a fourth-order Runga-Kutta scheme is used.

$$\frac{dx_i}{dt} = (x_{i+1} - x_{i-2})\, x_{i-1} - x_i + F \tag{2.5}$$

In the test case of this section, 40 dimensions were used and the $5^{th}$ dimension was observed and assimilated. $F$, the force constant is known to create chaotic behaviour when a value of 8 is used (Lorenz, 1995). On the $19^{th}$ dimension, the initial state of each ensemble member is perturbed by a zero-mean normal distribution with a variance of 0.01. If left to run without data assimilation, the result is chaotic. The $5^{th}$ dimension is shown in figure 2.5.

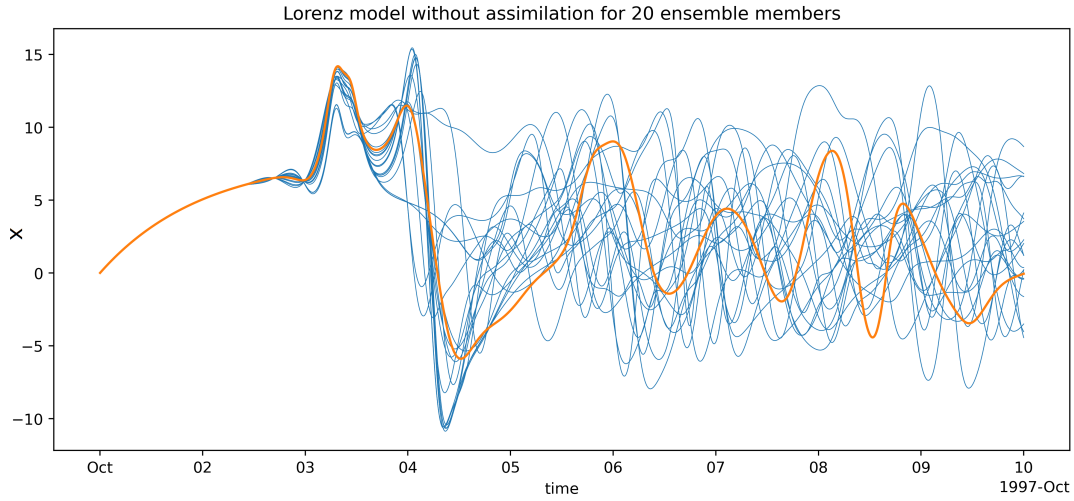



**Figure 2.5:** Lorenz model with every state is perturbed slightly at the start, leading to chaotic behaviour over time. Synthetic observation known as truth run shown in orange and the other states in blue. The state (x) of the $5^{th}$ dimension of the ensemble is shown against time.

To generate synthetic observations, one model run is selected and noise is added to its state. The noise follows a zero-mean normal distribution with a variance of 0.01, similar to that of the perturbations. This run is shown in orange in figure 2.5.

## 2.4. Experiments with a hydrological model

The experiments are the first full-scale test of the data assimilation framework. An overview of how the framework will be used can be found in figure 2.6. This shows the data flow into the framework and demonstrates how the framework was used in the experiments. The Catchment Attributes and Meteorology for Large-sample Studies (CAMELS) dataset is used to implement data assimilation across many catchments in the USA, the model runs can be analysed to show where deficiencies in conceptual models lie.

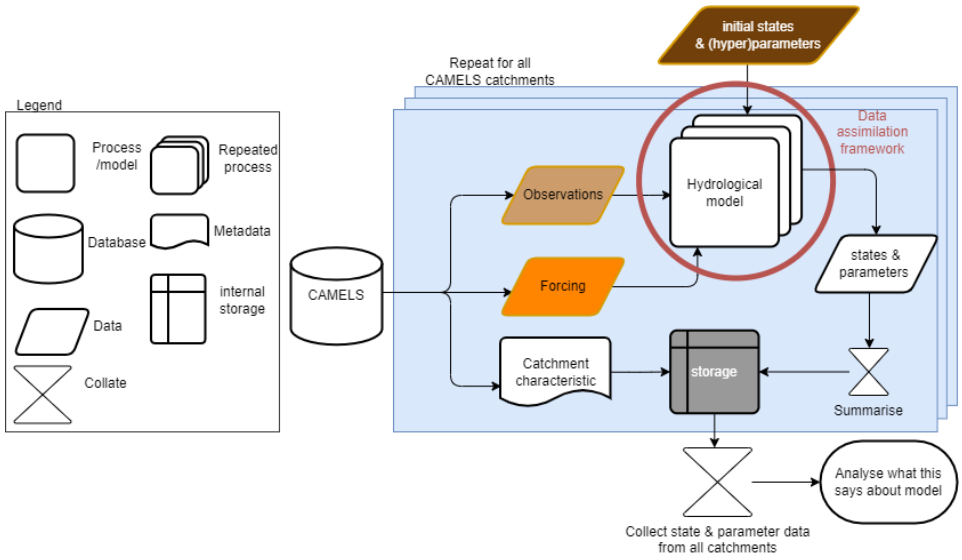



**Figure 2.6:** Flow chart showing the proposed data assimilation structure to use with the experiments. In blue the section repeated for many catchments. The red circle shows where the data assimilation framework operates relative to the other experimental components.

### 2.4.1. HBV model

A schematic representation of the HBV model is given in figure 2.7.

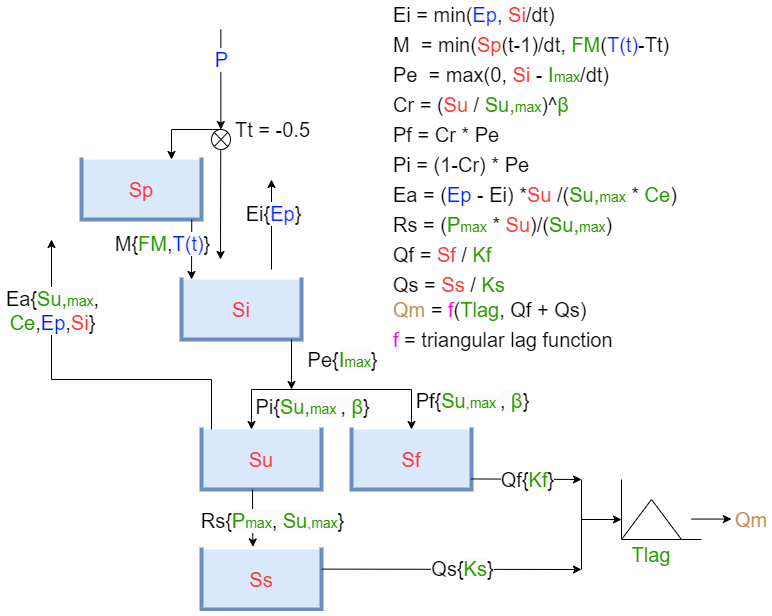



**Figure 2.7:** Schematic representation of the hydrological model HBV and underlying equations. Forcing is given in blue, states in red and parameters in green.

The five states representing the snowpack, interception, unsaturated rootzone, fast reservoir (runoff) and slow reservoir (groundwater) are given in red. The nine parameters are given in green. These combined with the modelled discharge (Qm) form the fifteen quantities included in the state vector. These nine parameters are fixed inputs to the model in normal use cases. The three forcings are precipitation, potential evaporation and temperature. These forcings are given in blue.

A Monte Carlo calibration method is used as a baseline to which a model that uses data assimilation can be compared. Using 500 Monte Carlo simulations, different combinations for the nine parameters are tested. Each parameter is chosen randomly within a range of parameters. The distribution of these parameters is unknown, thus the very rough assumption is made that these are uniformly distributed between a maximum and a minimum value. This assumption is a common approach in initial calibration methods used in hydrology (Seibert, 1997). The parameter range values used are given in table 2.1.

**Table 2.1:** Range of parameter values used in calibration

| | Min | Max |
|---|---|---|
| $I_{max}$ | 0 | 8 |
| $C_e$ | 0.2 | 1 |
| $Su_{max}$ | 40 | 800 |
| β | 0.5 | 4 |
| $P_{max}$ | 0.001 | 0.3 |
| $T_{lag}$ | 1 | 10 |
| $K_f$ | 0.01 | 0.1 |
| $K_s$ | 0.0001 | 0.01 |
| FM | 0.1 | 6 |

These are similar to those used by Seibert (1997), but with a widened range to better suit a variety of catchments. The approach described for calibrating a model is very basic, ideally a more state-of-the-art technique is used, but given the scope of this thesis, this approach will suffice. The calibrated model runs are not directly used to analyse deficiency, but they are compared to the data assimilation method. Ideally, a validation period is also used following this calibration, as taking the results from only the calibration period itself can be seen as cherry-picking. In this thesis, only the calibration period was used, as further explored in section 2.5.

The time period used was the same as in experiments using data assimilation, which is explained in section 2.4.3. Each model simulation is compared to the observations using the Nash-Sutcliffe Efficiency (NSE) and the log NSE (Nash and Sutcliffe, 1970). The NSE and log NSE are given in equation 2.6, with $Q_o$ being the observed discharge, $\bar{Q}_o$ its mean, and $Q_m$ the modelled discharge.

$$NSE = 1 - \frac{\sum(Q_o - Q_m)^2}{\sum(Q_o - \overline{Q_o})} \quad \text{and} \quad NSE_{log} = 1 - \frac{\sum(ln(Q_o) - ln(Q_m))^2}{\sum(ln(Q_o) - ln(\overline{Q_o}))} \tag{2.6}$$

The log NSE follows the same approach but with the natural logarithm of the values. This focuses on the timing of the peaks rather than the magnitude which is more suited in regions with high flood peaks (Oudin et al., 2006). A (log)NSE value of 1 is a perfect fit, whilst 0 indicates that the mean of observations would be a better measure. The highest NSE and log NSE simulations are stored to be used for comparison later. Although not always the best goodness of fit indicator, it was chosen as it is common practice in hydrology making it comparable to other research. Bárdossy (2007), Rakovec et al. (2012) and Kim et al. (2021) to name a few who use the Nash-Sutcliffe Efficiency as a goodness of fit indicator.

### 2.4.2. Experiments using synthetic observations

Testing the framework with synthetic observations can also be done for the HBV model. The model is run once with a randomly chosen set of parameters known as a truth run. The original forcing is kept the same. To the output of the truth run, noise is added to simulate the noise in real observations. This truth run with noise is used as observations. The added noise is sampled from a zero-mean normal distribution with a standard deviation of 0.05$mm/d$. Several combinations of parameters and noise are

investigated to see what the effect is on the behaviour of the ensemble and how well it captures the generated observations. This serves as an extra check that the framework functions as expected.

### 2.4.3. Experiments using real observations

For the experiments using the HBV model, the Catchment Attributes and Meteorology for Large-sample Studies dataset (CAMELS) was used (Newman, Andrew, 2014). The CAMELS dataset has precipitation, temperature, evaporation and discharge observation data for 671 catchments spread across the contiguous United States. The catchment locations can be seen in figure 2.8, representing a broad range of catchment types. Using this existing dataset removes the need for data collection, and guarantees a level of quality as the forcing data has already been tested on a set of hydrological models.

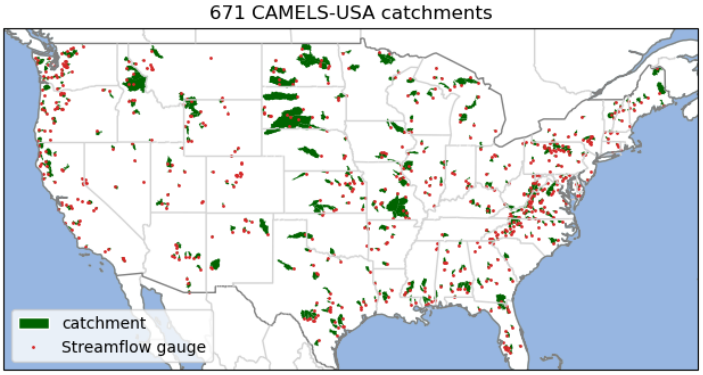



**Figure 2.8:** Map showing the locations of the catchments in the CAMELS-USA data set and the gauges

The streamflow observations are obtained from the United States Geological Survey (USGS). The dataset contains data between 1980 and 2010 for three different forcing sources: Daymet, Maurer and NLDAS. These are gridded meteorological data products. Daymet has the finest resolution at 1x1km, whilst the other two sources have resolutions of 1/8th a degree. Daymet (2024) aims to reproduce the weather conditions in the whole of the USA. NLDAS (2024) is more focused on soil moisture stores and energy. Both Daymet and NLDAS are NASA products. The dataset by Maurer et al. (2002) is a baseline for climate predictions. For the purpose of this research, Daymet was chosen. The precipitation data has already been converted to daily lumped data for each catchment by Newman, Andrew (2014). The potential evaporation $E_p$ needs to be calculated and is given by equation 2.7. The factor $\alpha$ is a correction factor that compensates for the lack of data normally used to calculate the aerodynamic effect in the Penman-Monteith equation.

$$Ep = \frac{\alpha}{\lambda}\frac{s(R_n - G)}{s+\gamma} \quad \text{where} \quad s = \frac{4098\left[0.6108\exp\left(\frac{17.27\ \mathrm{T}}{\mathrm{T}+237.3}\right)\right]}{(\mathrm{T}+237.3)^2} \tag{2.7}$$

Where:

- $\lambda$ latent heat of vaporization = 2.45 $\times 10^6$(J/kg)
- $R_n$ net radiation estimate in J/m$^2$/d
- $G$ is soil heat flux = 0 in a day
- $s$ is the slope of the saturation vapour pressure relationship (kPa/°C)
- $\gamma$ is psychrometric constant 0.066 (kPa/°C)
- $\alpha$ is the unitless P-T parameter provided by the CAMELS dataset.

The data set also contains metadata for each catchment, including topographic, climatic, land cover, soil and geological characteristics for each catchment. These metadata allow for a comparison between the characteristics of the catchments and the resulting variations in parameters and states (state vector) from the model runs, explored later on in 2.5.

For the experiment in this thesis, a time period of 5 years was selected between 1997 and 2002. This limited time frame was mainly due to the constraints on computational times. 5 years should be long enough to capture the behaviour of the model in the system.

### 2.4.4. Data assimilation hyperparameter analysis for hydrology

Using a variety of structured trial and error techniques, a set of suitable values can be found for the three hyperparameters $N$, $\sigma_w$ and $\epsilon_p$. $N$ was chosen by varying between 200 and 800 for different catchments and values of $\sigma_w$ and $\epsilon_p$. In the case shown in figure 2.9, below $N = 500$ there is a visibly smaller distribution width, whilst above 500 there is no real visible change to the distribution. To keep computation times acceptable, $N = 500$ was chosen. For $k$, the length of the assimilation window, a value of 3 was chosen. The effect seemed less significant than the others and thus was not investigated. After testing, $f_n$ was adjusted from the initial value of 0.8 to 0.975 to ensure resampling occurred often enough.

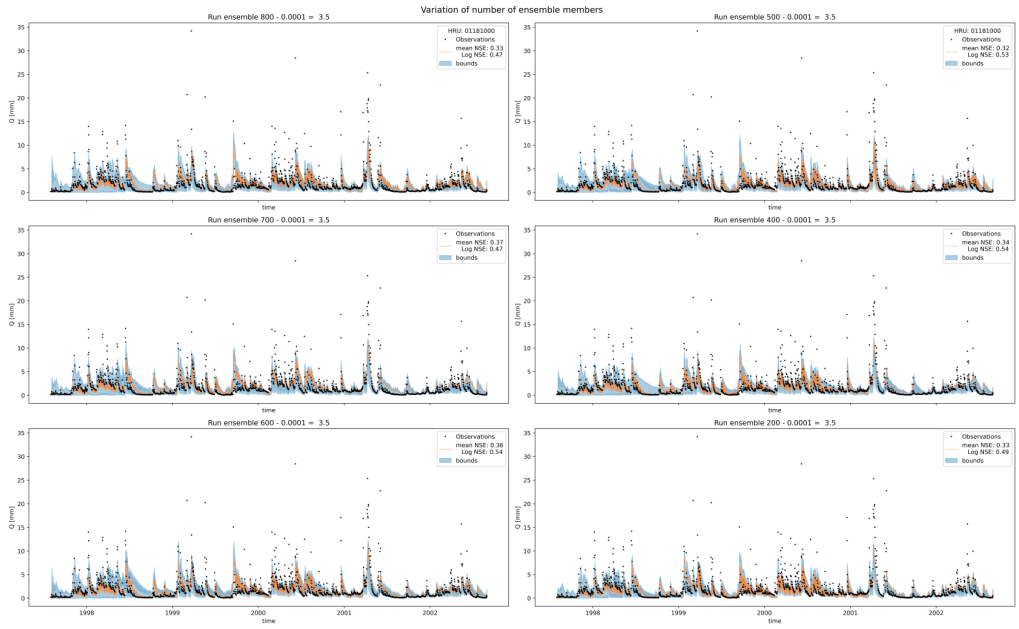



**Figure 2.9:** Varying ensemble size

Choosing a value for $\sigma_p$ requires more attention due to the fact that in the HBV model, all parameters have different units. In the Lorenz model shown, the states all had the same units. $\sigma_p$ controls the variance of the distributions from which the jitter is sampled which is added after resampling. When this distribution is the same for all parameters and states, some smaller values will comparatively be perturbed more than large ones. Parameters are generally smaller, thus when testing with the same distribution for jitter, the states that are controlled by the parameters could not adjust appropriately causing them to behave unrealistically. To combat this, an initial run is first done for two years of data. Here an initial guess for the hyperparameters is used, known to work for a catchment from previous testing. The data assimilation scheme takes two to three months to adjust the states, this is known as a spin-up period. This serves to gain a catchment-specific estimate of the size of the parameters and states. The distribution of the state vector quantities is unknown and difficult to estimate. Similar to the Monte Carlo calibration, the distribution is assumed to be uniform. This is a very rough estimation, but it is done more often in hydrology (Seibert, 1997). The mean of the maximum ($\mu_{Max_j}$) and minimum ($\mu_{Min_j}$) of each quantity on the state vector is computed (see equation 2.8), excluding the initial spin-up period of three months. Using this maximum and minimum, an estimate of the standard deviation ($\sigma_{z_j}$) can be computed for each quantity. The vector $\sigma_{\mathbf{z}}$ serves as input for the actual run, to ensure that the amount of jitter is consistent with the size of parameters and states in different catchments. The value of $\sigma_{p_j}$ corresponding to $z_j$ is given by equation 2.8.

$$\sigma_{p_j} = \epsilon_p \sigma_{z_j} \quad \text{where} \quad \sigma_{z_j} = \frac{(\mu_{Max_j} - \mu_{Min_j})^2}{12} \tag{2.8}$$

Where $z_j$ is a quantity on the state vector $\mathbf{Z}$. Adding noise with a standard deviation which is a fraction of a measurable standard deviation is a common approach used in literature (Moradkhani et al., 2005,

Piazzi et al., 2021, Weerts and El Serafy, 2006). The actual hyperparameter controlling the jitter added is therefore $\epsilon_p$. The actual state vector must always be greater than or equal to zero in the HBV model used, thus this constraint is imposed after resampling. To determine a suitable value for $\epsilon_p$, trial and error was used. If $\sigma_w$ and $\epsilon_p$ are chosen too large, the spread of the ensemble becomes too large and the results are no longer meaningful. If they are chosen too small, the danger with particle filters is a collapse of the ensemble where only one particle is deemed suitable by the algorithm when resampling. The process outlined here seems complex but ensures that particle filtering can be applied to a variety of catchments without tuning hyperparameters for every catchment.

To determine a feasible value for $\sigma_w$ and $\epsilon_p$, the first 26 catchments listed in the CAMELS dataset were used to test different combinations, shown in 2.10. The downside of using the first set was that they were not varied in geographical location and thus did not form a representative selection for all the catchments.

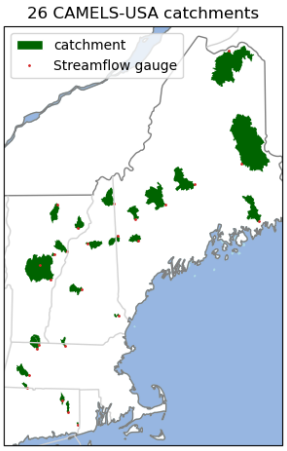



**Figure 2.10:** Maps showing catchments used to estimate hyperparameters

With a constant value of $N = 500$, $\sigma_w$ was varied between 2 and 9 and $\epsilon_p$ between $10^{-2}$ and $10^{-7}$ for 21 different combinations. These values were chosen based on earlier tests with just one catchment where $\epsilon_p = 1e-4$ and $\sigma_w = 3.5$ were the best choices. Values of $\epsilon_p$ larger than $10^{-1}$ lead to an unrealistically large spread in the ensemble, while values of $\sigma_w$ smaller than 1 lead to a collapse of the ensemble to just a few particles.

To determine if a combination of hyperparameters is a good fit, an initial visual inspection is useful to see if the ensemble captures the observation well, as done in figure 2.9. Analysing the 546 different model runs resulting from testing 21 different parameter sets on 26 catchments is visually unfeasible. To summarise the goodness of fit for a larger set of catchments, the mean of the ensemble at each time step was compared to the observations using the Nash-Sutcliffe efficiency (NSE) and the log NSE (Nash and Sutcliffe, 1970). The NSE is a measure that compares the difference in model output to the observations, normalised by the difference between the observations and their mean, assessing how well a model can predict the flows in a system. Using the NSE of the mean of the ensemble is not in every case the best measure for how effective data assimilation is. Looking at whether the observations lie in the ensemble could also be a measure. If the observation lies on the edge of the ensemble, the ensemble does capture the observation well, but this is not reflected in the goodness of fit. For this reason, the ensemble member with the highest weight, or the best ensemble member, is also used later on. As the weight, and thus the spread of the ensemble, is fairly low, using the mean is a fair simplification for hyperparameters estimation. This is further explored in section 2.5.

The mean NSE across the 26 catchments can be compared to the different hyperparameters to choose one best-fit combination of hyperparameters as shown in figure 2.11. A value of $\epsilon_p = 1e-3$ and $\sigma_w = 2$ is a good fit for these catchments, as can be seen in figure 2.11. The feasible parameter space can be estimated from this distribution. Any combination in this region will produce usable results. More testing was done closer to $\epsilon_p = 1e-3$ and $\sigma_w = 2$ to verify this, thus we see $\epsilon_p = 5e-4$ would be a better choice. Due to time constraints and long computation times, the hyperparameter selection process was

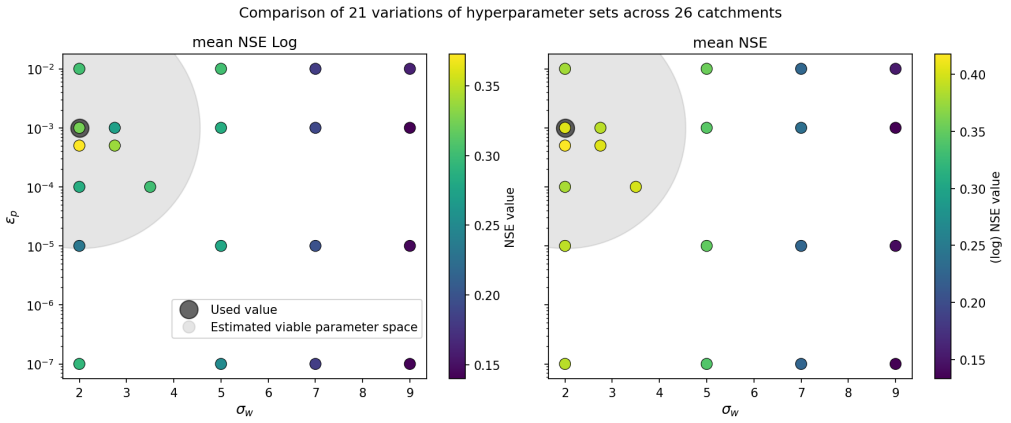



**Figure 2.11:** Comparison of hyperparameters to the mean (log)NSE across 26 catchments

highly non-linear. The parameters set was already chosen and applied to the whole CAMELS dataset experiment when the final round of hyperparameter optimisations finished. Restarting the analysis of the CAMELS dataset with the slightly better parameter set was not done due to time constraints. For the purpose of this thesis, the hyperparameters used are $N = 500$, $\epsilon_p = 1e - 3$ and $\sigma_w = 2$.

## 2.5. Analysis of deficiency

To analyse what deficiencies models have, the underlying assumption is that in a 'perfect' model parameters are constant in time. The data assimilation scheme is applied to a five-year period for all the CAMELS catchments, storing a summary of the parameters and states. The constraint in time of 5 years is due to the computational power it takes to run a large number of catchments. Running the model for a longer time period could give more insight into the long-term behaviour, but this comes at a cost. An initial spin-up run of 2 years is used to improve the performance of hyperparameters, as described in section 2.2.1. The experiment is carried out with the hyperparameters which resulted in one of the highest mean of the NSE in the calibration catchments. Some catchments will for this reason not be captured well by the scheme. Adjusting the hyperparameters for each catchment is possible and would lead to better results. Catchments where both the NSE and log NSE values are below 0 are not used in the analysis of deficiency.

During the experiment, only a summary of values was stored to optimize the speed and storage usage of the experiment, since storing the whole state vector becomes computationally expensive. The summary of the parameters and states stored was the maximum, minimum, mean and the best. The best is defined as the particle that has the highest weight and thus predicts the observations best for the given time step. Choosing the most suitable metric to summarise the ensemble can be tricky. Using the mean describes the behaviour of the ensemble as a whole best and is mostly used in the field of data assimilation. Because of this, the mean of the ensemble is also used for the calibration of the hyperparameters. Choosing the highest weight can lead to cherry picking as a very widely spread ensemble can always generate 'the best' results. In the baseline calibration of the HBV model used, the best of 500 model runs is used. If the mean of the Monte Carlo simulations is used, this would not produce as good of a model fit as explored. As we are interested in deficiencies in the model and not deficiencies in the calibration, we take the best result of the calibration. To compare to this best calibration run we also use the best which the ensemble can predict. At each timestep, the ensemble member closest to the observation is stored, this is 'the best' result. The changes in parameters and states which occur in order to model the streamflow in this way are analysed. Throughout the analysis, the bias introduced when choosing the best ensemble member every time step will be kept in mind.

Three sets of analyses will be done to investigate parameters and states and how this links to model deficiency. Initially, a visual representation of the parameters and states over time is useful to analyse patterns for one catchment. Analysing a large number of catchments visually is unfeasible. For this reason, the second analysis will focus on summarising the results. The experiments with data assimilation will be compared to the result of the Monte Carlo calibration run. The calibration is not perfect but can

be used to highlight where the HBV model shows deficiencies. The improvement in NSE when using data assimilation compared to the calibration will be used. The improvement in NSE is compared to catchment characteristics to give insight into which types of catchments perform better when using data assimilation. An improvement when using data assimilation in certain catchment types can highlight a deficiency in the model.

Using the results from analysing a large number of catchments, two catchments are chosen to be further analysed. Analysing nine parameters and five states can be difficult. For this reason, it is important to consider the three model phases: the initial phase, the fast phase and the slow phase. Initially, snow and interception are modelled. Interception includes everything that stops water from entering the soil or running off but is mainly vegetation. The snow routine works by storing precipitation as snow if the temperature is less than $Tt$ $(0.5°)$, and allowing it to melt if the temperature is above this using a melt factor parameter $(FM)$. The water resulting from interception is then split between the fast phase and the slow phase. The fast phase models all the water moving overland to the river. The slow phase models the underground-based processes including infiltration into the soil and percolation to the groundwater. These latter two phases also form two separate paths through which the water flux reaches the modelled discharge. This is shown in figure 2.12.

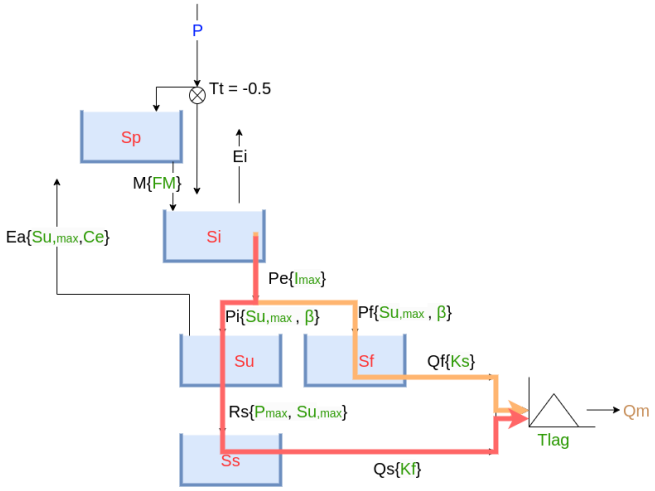



**Figure 2.12:** Schematic representation of the hydrological model with two flow paths

A five-year period is still a long period to analyse in detail. For this reason, the sum of the squared errors is computed as $SSE = \sum(Q_{obs} - Q_m)^2$. Comparing the sum squared errors for the data assimilated experiment to the calibrated model shows when the data assimilation adjusts to perform better than the baseline HBV model. Studying the moments in time when these adjustments occur and what changes to the parameters and states in order to better capture the modelled streamflow can give insight into deficiencies in the hydrological model.

### Expected results

In the experiments, data assimilation is applied to a variety of catchments. The HBV model at its core is a simple conceptual model which can not fully capture the observations given the forcing and nine parameters. Data assimilation can vary the parameters and states to find the best possible combination of parameters and states at every timestep. The comparison between the experiments utilising data assimilation and the calibrated model can show differences between the calibrated and the assimilated parameters and states. Analysing these differences can give an idea of where these deficiencies lie.

If a particular part of the model is deficient, I would expect that for given parameters we see a strong correlation in improvement in the data assimilated experiment when compared to the baseline model run. The catchment characteristics can also show a correlation, if the model is deficient in one particular type of catchment we would also expect to see a relation here. Analysing the changes on a catchment level I expect to observe changes in the fast and slow flow paths outlined. The two flow paths allow the data assimilation scheme to adjust either the fast or slow flow path in order to capture the observed streamflow better.

# 3 Results

To answer the research question ”What do the adjustments in parameters and states imposed by data assimilation say about the deficiencies in the hydrological models of observed streamflow”, a series of experiments are conducted. An experiment is defined as the application of the data assimilation framework with a model for a given area. The results of these experiments are presented and used to analyse deficiencies in the model.

The results in this thesis are twofold. Firstly, the data assimilation framework itself is a delivered result. The framework allows users to use data assimilation in their modelling on the eWaterCycle platform. Some results from tests with synthetic data are shown in section 3.1. Secondly, the data assimilation framework was applied to a large number of catchments. This demonstrates why a framework was necessary and is used to analyse deficiencies in the HBV model on a large scale. The catchments used are those in the CAMELS dataset, a collection of hydrological data for 671 catchments across the contiguous USA. These experiments can be analysed to investigate deficiencies in the HBV model.

## 3.1. Experiments with synthetic data

The data assimilation scheme can struggle to capture real observations due to noise or a mismatch of hyperparameters. The framework can be tested using synthetic runs to ensure the data assimilation itself works as expected. Synthetic runs use a model run with noise added to the output to act as observations.

### 3.1.1. Lorenz model - synthetic experiment

The Lorenz model is often used as a benchmark for data assimilation in literature. Here synthetic observations shown in orange are used, generated from a previous model run. Without data assimilation the system shows chaotic behaviour, however, the particle filtering scheme ensures the observations are still captured well by the ensemble as seen in figure 3.1. Each blue line represents the path of one particle through time with the observations the data assimilation scheme aims to model shown in orange.

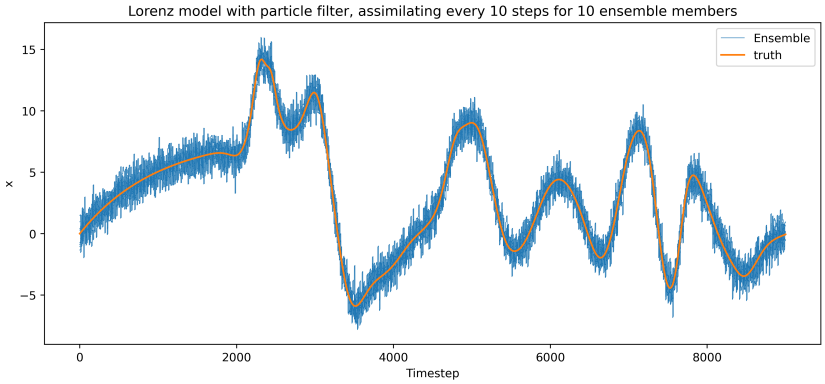



**Figure 3.1:** Lorenz model with 20 particles, assimilating after every 10 steps

Figure 3.2 shows the behaviour with a longer assimilation window. Here, after only 50 time steps resampling takes place. In the zoomed-in section the updating of the state after 50 time steps can clearly be seen. This demonstrates that, for this use case, the framework functions as expected.

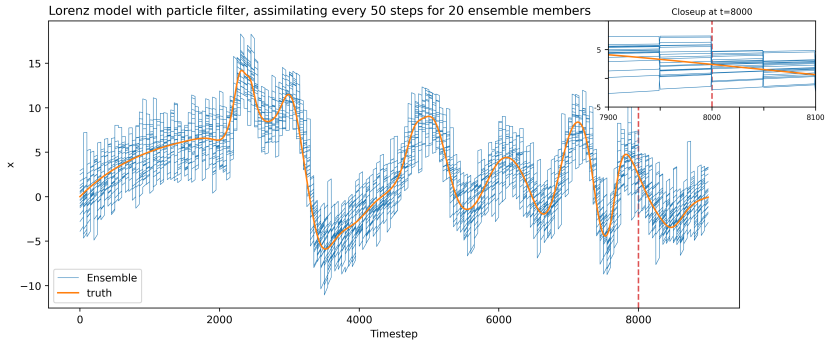



**Figure 3.2:** Lorenz-96 model with 10 particles, assimilating after every 50 steps. The blue lines show the state (x) of the ensemble over time, the orange shows the truth run or synthetic observations used to assimilate.

### 3.1.2. HBV model - synthetic experiment

In figure 3.3 the results from the synthetic experiment are shown, with $\sigma_w$ set to 0.005. The value of $\sigma_p$ matches the noise added and is set to 0.05. The hyperparameter $\sigma_w$ affects how spread out the ensemble is; too small of a value resulted in collapses of the ensemble. A value of one-tenth $\sigma_p$ seemed reasonable for this use case. When the scale of the noise distribution and the value of $\sigma_p$ were different, the scheme did not capture the synthetic observations well. This experiment shows the importance of finding a suitable $\sigma_p$. From the synthetic experiment we see that in a favourable scenario, the data assimilation scheme is capable of capturing the model well. The blue area in figure 3.3 shows the bounds of the ensemble, as only a summary of the ensemble was stored at each timestep. With 25 particles for 2 years, storing and plotting the path of each ensemble member is still feasible. Moving to larger ensemble sizes and longer time periods, this becomes computationally very expensive. For this reason only the bounds are only shown on all plots from here onwards.

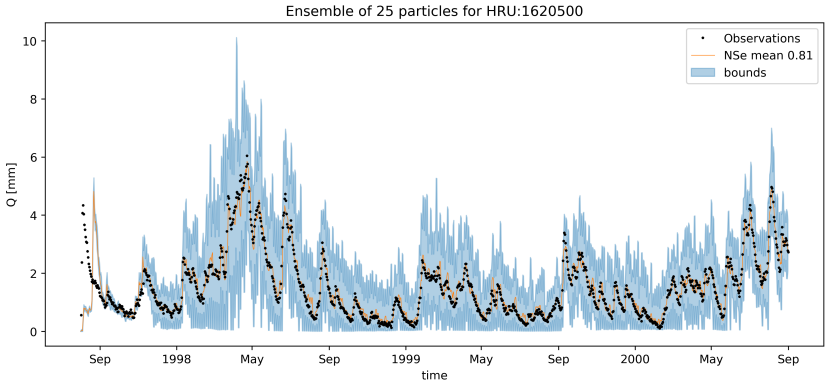



**Figure 3.3:** Assimilation using Particle Filter of synthetically generated discharge (mm) observations (black) for HBV model. The blue area shows the bounds of the ensemble, and the orange line the ensemble member with the highest NSE over the whole time period.

## 3.2. Experiments with real data

Applying data assimilation with synthetic observations shows that the data assimilation scheme and framework function. Synthetic observations do not test the model and system itself. For this, real data was used. First, one catchment is shown with the parameters and states. Then this is repeated for a large number of catchments. Lastly, a few of these catchments are further analysed. Throughout the analysis, the ensemble member with the highest weight is chosen, unless specified otherwise.

### 3.2.1. HBV model for single catchments

To demonstrate the results the data assimilation produces, one catchment was first chosen to show the results of applying data assimilation to a hydrological model. This catchment is not part of the 26 catchments used for the calibration of the hyperparameters. It was selected as it shows how data

assimilation improves the predicted streamflow compared to the baseline-calibrated HBV model.

### Bush Kill at Shoemakers, Pennsylvania

Figure 3.4 shows the hydrograph of the experiment with data assimilation on top and the calibrated HBV model is shown on the bottom. We see that the ensemble is better at capturing the peaks and the recession curves than the calibrated HBV model, although still struggles to fully capture all the peaks as it has an NSE of 0.91.

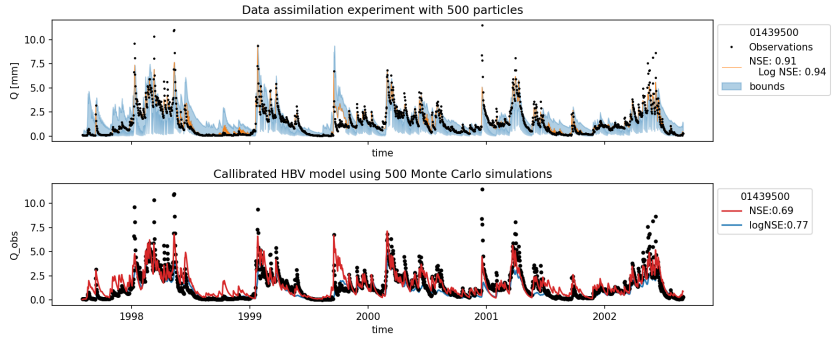



**Figure 3.4:** Experiment with data assimilation compared to baseline HBV model *TODO: add indicator when resampled*

The parameters and states corresponding to the hydrographs shown can be seen in figure 3.5. The black line shows the parameter or state in the baseline case, the dots show the parameter or state of the particle best capturing the streamflow at that time and the solid coloured line shows the parameter or state of the mean particle. The states generally follow the pattern of the baseline, with the exception of groundwater storage. The groundwater storage ($Ss$) can be seen to build up over time, the factor $K_s$ which controls the outflow is by comparison much lower. The dots show that there are periods where the highest weight particle is up at the maximum bound, mostly during the winter. This is likely because the simple conceptual model can not fully explain these peak flooding events well and the data assimilation scheme adjusts to try and explain these.

For almost all parameters we see a step change in the fall of 1999, where a large flood peak occurs much earlier than the normal spring floods. The mean of the parameters $I_{max}$, $T_{lag}$, $K_f$ and $\beta$ all go up during this event, whilst the parameters $Ce$, $K_s$ and $P_{max}$ decrease at that event. Using the mean ensemble parameters the effect is more visible, overall the best ensemble member also shows this effect. The precipitation peak (100+mm) is also much higher than the discharge peak (5mm). A precipitation peak of this magnitude either points to errors in the data or a very large storm event. Either way, the stormwater needs to be stored somewhere as it does not immediately flow out and cannot be stored as snow as it is too warm. We see an increase in $Su_{max}$, which causes the soil ($S_u$) to hold more water. The calibrated model (black line) in the $S_u$ plot also increases here, but we see that the data assimilation increases more.

In December of 2000, we see another step change, but different to the one in 1999. The mean of the parameters $Ce$and $P_{max}$ increase, whilst the $Su_{max}$, $T_{lag}$, $K_f$ decrease. Here $K_s$ stays unchanged and mean $S_s$ increases slightly. Here the temperatures are lower meaning some of it is stored as snow, but there is a decrease followed by a sharp increase in the $K_f$ parameter and a peak in the $S_f$ storage. The length of lag applied ($T_{lag}$) reduces to one. This suggests that the timing of flood peaks in the model is deficient, which the data assimilation compensates for.

In the past two paragraphs, the focus was on the mean of the ensemble, however, the best prediction of streamflow at a time step can give us more insight into deficiencies. For some parameters, this best parameter changes rapidly, for others less so. Looking closer at the $T_{lag}$ parameter of the best ensemble member, we see it decreasing during high discharge flows and increasing during periods of low flow. In the relatively dry summer of 1998 this is clearly visible, but also repeats in the other years. During this time the $Ce$ parameter decreases and the $\beta$ parameter. This could show the model operating well in a drier period as the values that the best parameter takes are repeated in the following dry periods. Relatively little percolates down to groundwater seen by low values of $P_{max}$ and $S_s$ during

this period. Most of the water runs off almost directly to the river after being held by interception storage ($S_i$) and the fast flow storage ($S_f$), as we see peaks in the fast flow storage when there is precipitation.

The higher storage and generally lower outflow factors, as described for groundwater previously, could indicate that the conceptual model considers too short of a time scale. Water can often stay in catchments for a long time, something the models do not take into account. The data assimilation scheme seems to compensate for this by increasing the size of the states and adjusting the outflow parameters when needed. We also see a combined adjustment of parameters and states across different parts of the model, suggesting that the timing of floods is controlled by the data assimilation scheme, resulting in higher (log)NSE values. We see that step changes are often linked to the occurrence of flood peaks. During these floods the nature of the system will often change, meaning a calibrated model can not capture these changes. The data assimilation experiment gives insight into the changes required to model the system and thus points to deficiencies in the model.

### 3.2.2. HBV model for all catchments in CAMELS

In the previous section, the results for one catchment were explored. A deficiency can be seen as a part of the model which underperforms causing the model to wrongly represent what we observe in reality. To analyse deficiencies in a model, more than one catchment needs to be considered. For this reason, the same experiments were conducted in 671 catchments in the CAMELS dataset. These contribute to the results presented in this chapter. One of the 671 catchments had a section of missing observations in the data during the time period and was thus omitted. The used hyperparameters produce good results in many catchments, but not all. Catchments with an NSE below 0 were disregarded for the purpose of identifying deficiencies with these particular hyperparameters.

#### Comparison of data assimilation with Monte Carlo calibration

To answer the research question, the aim is to highlight the deficiencies in the model. These deficiencies can be analysed when the data assimilation scheme performs better than an HBV model without the use of data assimilation. As a baseline, the HBV model was calibrated using 500 Monte Carlo simulations, each with a different combination of parameters. The NSE of this baseline model run can be compared to the NSE of the highest weighting particle of the data assimilation experiment. The results can be seen in figure 3.6. The dashed line shows where the calibration and data assimilation experiments are the same. When taking the highest weight particle at every time step, most lie above this line. In 574 of the 671 catchments, the NSE is higher than zero. The main reason for the 97 catchments having a poor fit is the use of a single combination of hyperparameters for all experiments. The selected combination of hyperparameter values works well for most but not all catchments.

The CAMELS dataset provides catchment characteristics as well as geographical information. The catchment characteristics can be used to compare experiment results of the different types of catchments to each other. Figure 3.7a shows the geographical information used to show the $\Delta$NSE values per catchment on the map. The grey catchments have values of $\Delta$NSE less than zero. If we compare the map of the $\Delta$NSEs to the map of dominant vegetation cover shown in figure 3.7b, we can see a correlation between types of similar dominant land cover and resulting $\Delta$NSE values.

The trend between the dominant land cover and $\Delta$NSE can be investigated further. Similar plots to figure C.1 can be made but including catchment characteristics. The relationships between the catchment characteristics and experiment results are explored for two characteristics: dominant vegetation type and aridity.

Firstly, the correlation between dominant vegetation type and $\Delta$NSE seen in the map in figure 3.7 can be explored further by investigating the fraction of forest in the catchment. Figure 3.8a shows the fraction of the catchment considered a forest. Here we see a slight correlation between catchments with fewer forests and their performance. If we only consider the catchments with a high forest fraction we see a stronger trend with a slight positive correlation for $Su_{max}$ and a slight negative correlation for $Ce$. The other parameters show no particular trend, although some clustering can be identified in areas. Such as around higher $K_f$ values of 0.1. Other vegetation characteristics were also accessed. The leaf area index and green vegetation fraction both showed similar trends as described here. More plots can be found in appendix C.3b.

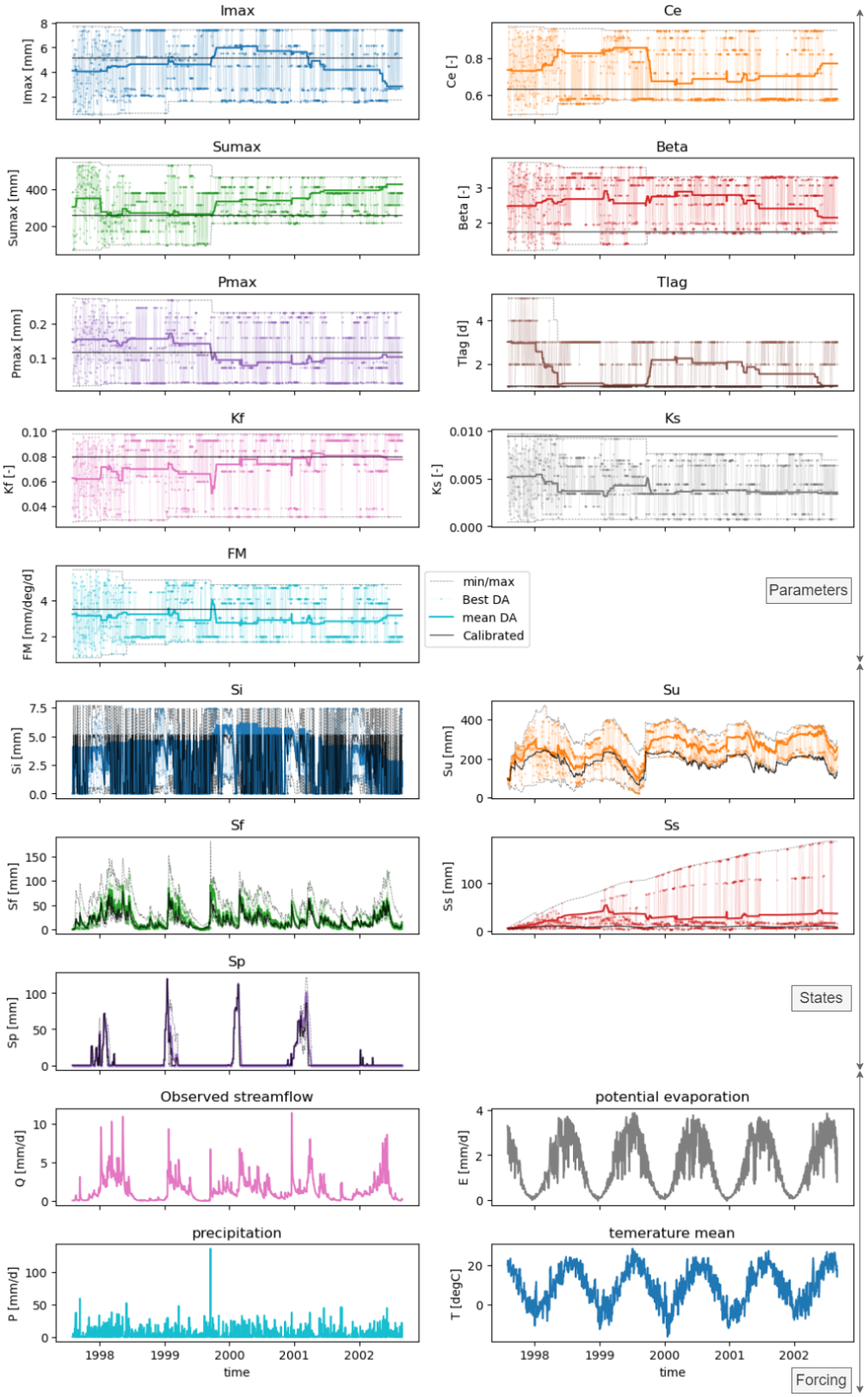



**Figure 3.5:** Nine parameters, five states, three forcing and observed streamflow for Bush Kill catchment. This catchment was selected due to a high NSE. Dotted grey lines show the min and max, the grey line shows the reference calibrated model, the solid coloured line shows the mean of the ensemble and the points show the ensemble member best predicting the observed streamflow.

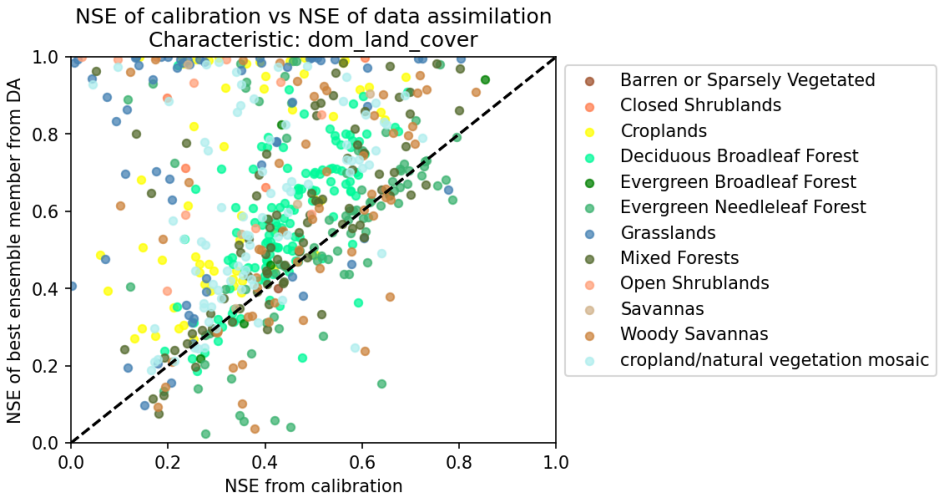
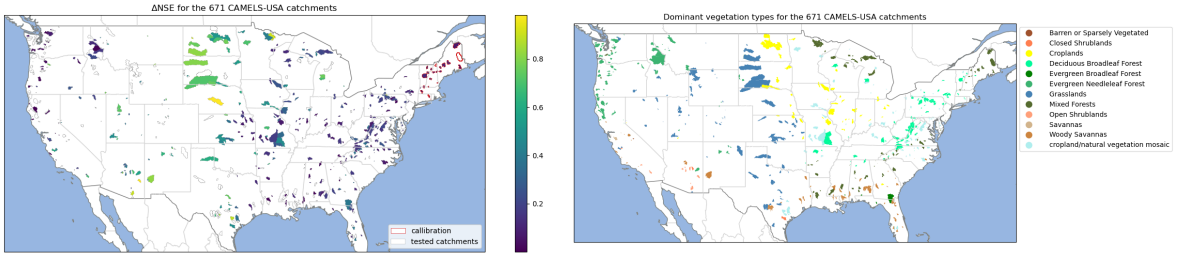



**Figure 3.6:** NSE of the calibration experiment compared to the NSE of the data assimilation experiment with catchments categorised by dominant land cover. Each dot represents a catchment.




**(a)** ΔNSE values for different catchments

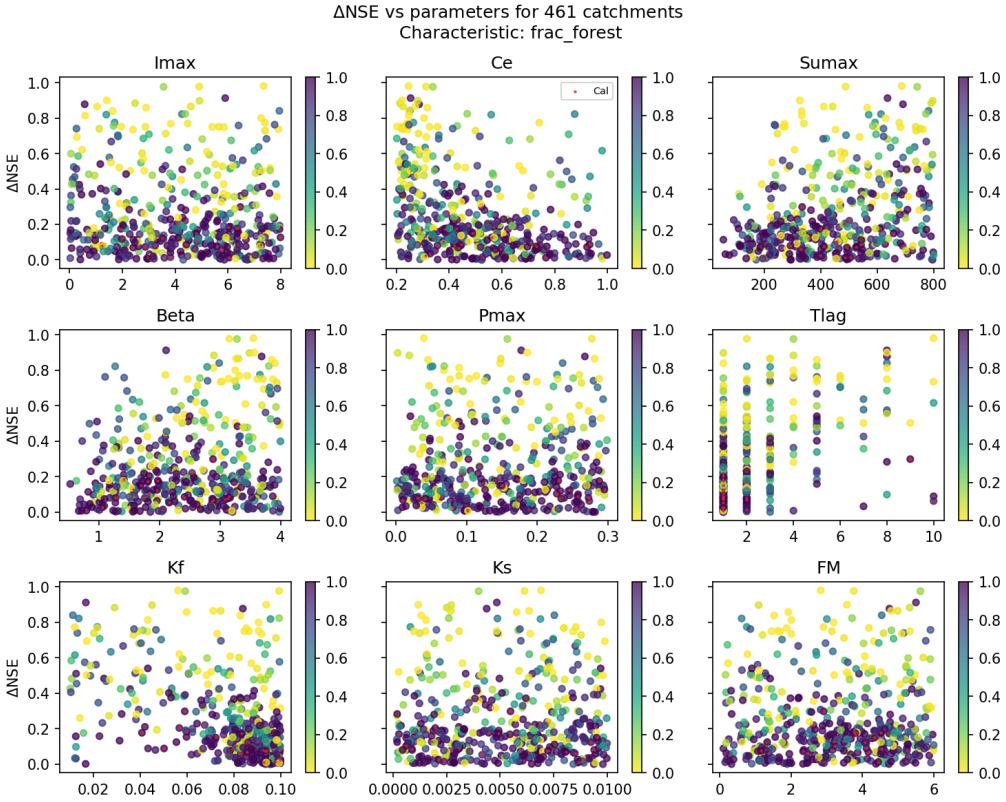
**(b)** Map showing dominant vegetation

**Figure 3.7:** Comparison of maps of ΔNSE and dominant vegetation type

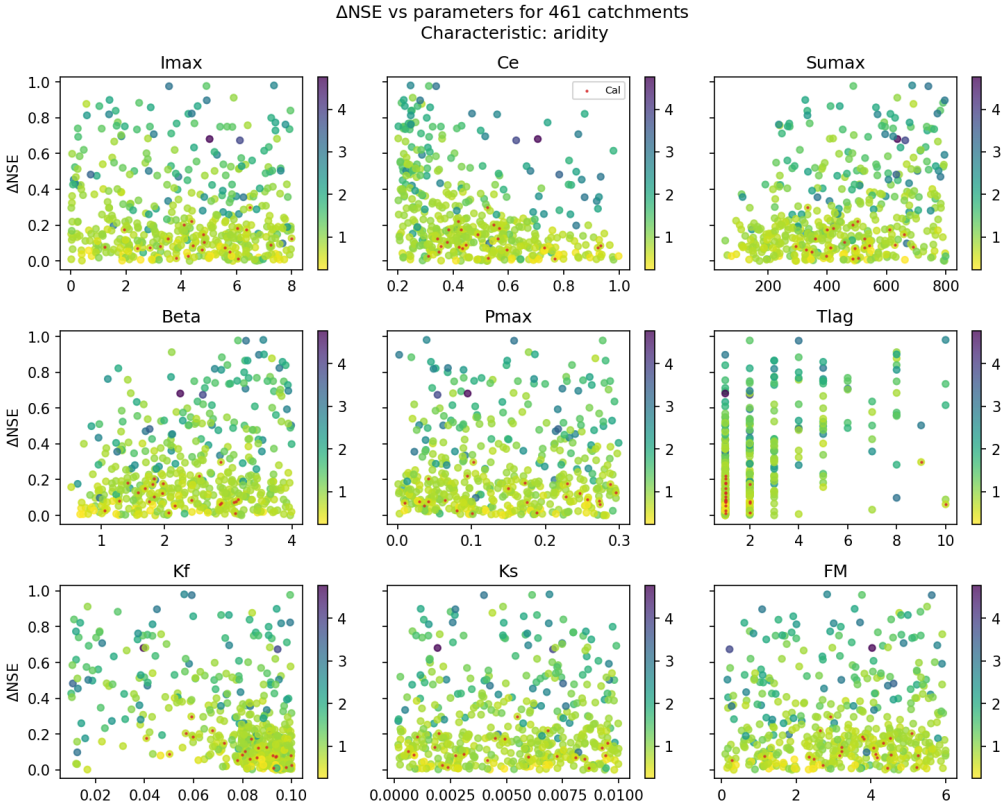
Another characteristic that also links to the shown map is aridity. Aridity is defined as the ratio of mean potential evaporation to mean precipitation. Show in figure 3.8b, we see a strong correlation to higher aridity and more improved results from the data assimilation experiments. The other climatic factors also show a strong correlation but are mostly related to aridity. Two more plots (figure C.4) can be found in the appendix. The geological and soil characteristics showed little to no correlation to improvement in NSE. On the map, it looks as if the larger catchments have a better score. This is caused by the over-representation on the map of the large catchments. When plotted on a graph there is no clear trend.

From the comparison of data assimilation to the baseline experiment we see that catchments with similar properties have a similar improvement in NSEs when the set of hyperparameters is used to apply data assimilation. All of the forest based catchments improve between 0 and 0.4. More arid catchments improve between 0.4 and 0.8. One important factor is that the combination of hyperparameters used was calibrated on 26 catchments with predominantly forests with a mean discharge of around 4mm/day as shown in figure 3.9a. Figure 3.10a compares the mean discharge to the ΔNSE. We see that across 574 catchments the relationship holds where a lower mean discharge leads to a better improvement in NSE.

The more arid catchments have a much lower mean discharge of 1mm/day. Using the ensemble member with the best predicted discharge at every time step is in this case biased towards the catchments with a low mean discharge. This bias is because if the same amount of noise is added, the basins with a lower discharge have a relatively larger spread. If the spread is very large, it is much easier for the data assimilation algorithm to model the correct discharge. Investigating this spread can be done very roughly by computing the spread as $Q_{spread}$ as given in equation 3.1.

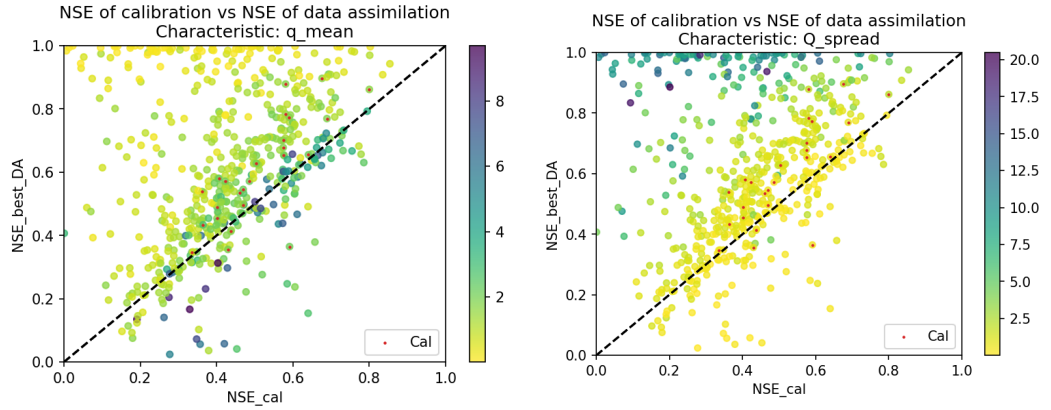



**(a)** ΔNSE compared to the calibrated parameters and forest fraction




**(b)** ΔNSE compared to the calibrated parameters and aridity

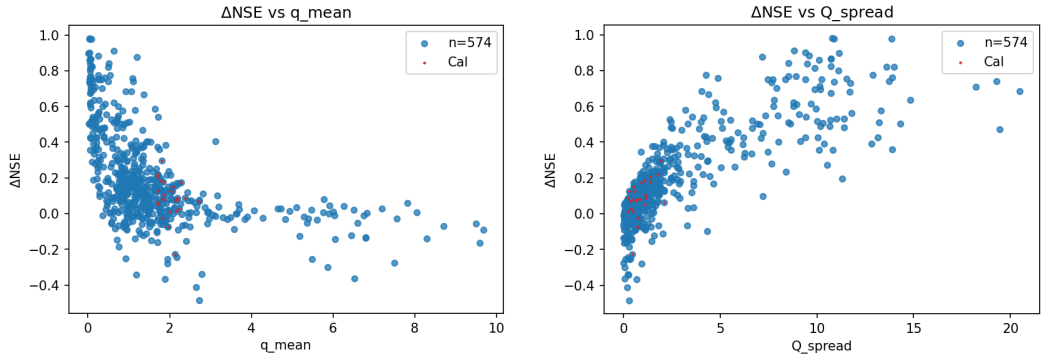
**Figure 3.8:** Comparison of two characteristics sets to ΔNSE




**(a)** NSE of the calibration experiment compared to the NSE of the data assimilation experiment with mean streamflow

**(b)** NSE of the calibration experiment compared to the NSE of the data assimilation experiment with ensemble spread

**Figure 3.9:** Fit (NSE) of calibration vs data assimilation with mean streamflow and ensemble spread as colored variables. Red dots show catchments used for hyperparameter calibration. Each dot represents a catchment.




**(a)** ΔNSE compared to daily mean observed streamflow ($q_{mean}$)

**(b)** ΔNSE compared to the experiment ensemble spread $Q_{spread}$ given in equation 3.1

**Figure 3.10:** Comparison of mean streamflow and ensemble spread to the improvement in fit (ΔNSE) due to data assimilation.

$$Q_{spread} = \frac{Q_{max} - Q_{min}}{Q_{mean}}. \tag{3.1}$$

In reality, the posterior distribution of the ensemble is more complex, but only three indicators were stored during the experiments and not the whole posterior. The posterior distribution can be seen in figure 3.9b. Here we see that all the values which have a very large improvement, also have a very large spread in the ensemble. The spread in the ensemble can also be compared to the ΔNSE as shown in figure 3.10b. We see a relation between a higher spread and higher values of ΔNSE.

From this analysis, we see that there is a relation between the improvement data assimilation offers and the background characteristics of catchments. In my view, this relation is mainly due to the bias introduced when using the same hyperparameters across different catchments which in this case favours the catchments with lower stream flows. Due to this bias, no large-scale conclusions can be drawn about model deficiency. We can conclude that catchments with the same hydrological and vegetation signatures will

### 3.2.3. Detailed analysis of two catchments

To analyse how the changes in parameters and states show deficiencies in the hydrological model, the changes are analysed on a catchment level. Again the catchments chosen were not used for the hyperparameter calibration.

### SF Clearwater river - Idaho

The first catchment analysed is one with a larger spread and is situated in the northern part of Idaho. It demonstrates the conclusion made about this larger spread. The hydrographs and the residuals are shown in figure 3.11.

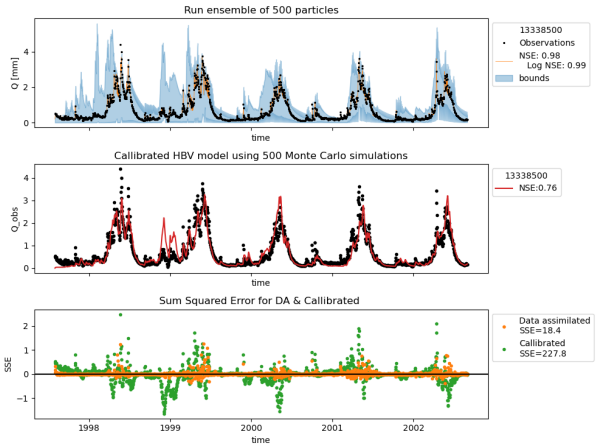



**Figure 3.11:** Hydrograph for data assimilation experiment, the baseline and both of their sum squared errors (SSE)

By choosing the best at every time step the fit has a value of NSE of 0.98. During the autumn flood peaks the spread is fairly acceptable. During the base flow periods, the spread can sometimes be larger. Although in 2002 the spread in base flow is acceptable. If we look at the changes in parameters and states, as shown in figure 3.12, we see some variation but the general distribution is quite constant. We do see some step changes in the spring of 1999, although less dominant than in the catchment analysed in section 3.2.1. A larger version can be found in appendix C.7. The main flood peaks, which can clearly be seen in the SSE plot are underestimated by the normally calibrated model. Although all the parameters do vary, they vary too much in the best case and too little in the mean to draw any meaningful conclusions. All the parameters of the best ensemble member in this catchment move frequently between the highest and lowest value, thus no trend can be observed. The main visible change is the adjustments in the slow ($S_s$) and fast ($S_f$) flow storage levels, which allow the model to capture the peak correctly. This was also observed in other catchments with a large spread in the ensemble.

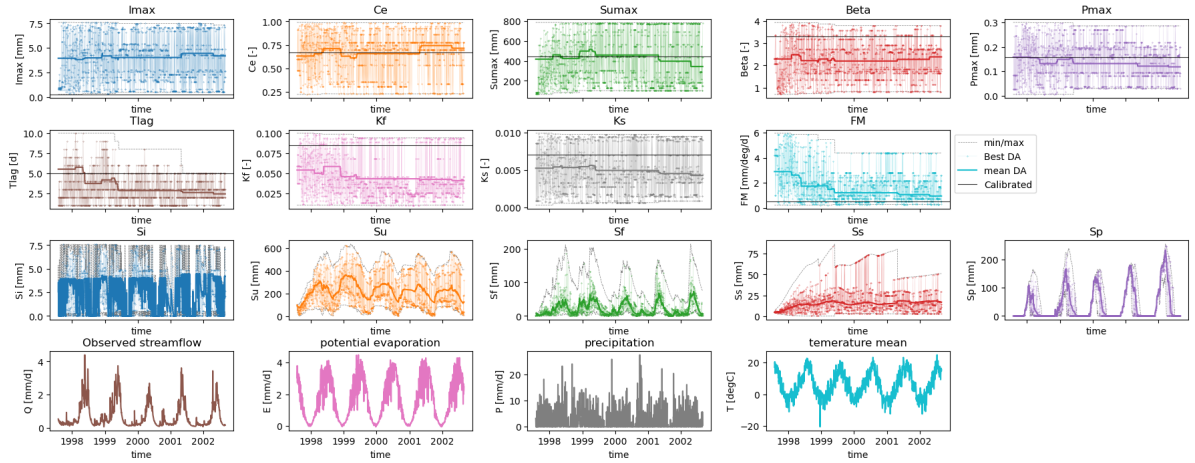



**Figure 3.12:** Parameters and states with time for 5 years in the SF Clearwater river catchment - Idaho. The nine parameters are: $I_{max}$, $C_e$, $Su_{max}$, β ,$P_{max}$, $T_{lag}$, $K_f$, $K_s$ and $FM$. The five states are: $S_i$, $S_u$, $S_f$ and $S_s$. The bottom row shows the observed stream flow and three forcing are: potential evaporation, precipitation and mean temperature. Dotted grey lines show the min and max, the grey line shows the reference calibrated model (for parameters), the solid coloured line shows the mean of the ensemble and the points show the ensemble member best predicting the observed streamflow. Appendix C.7 contains a larger version of this figure.

### Tahquamenon River - Michigan

Another catchment with a relatively large spread is in Michigan. This does get almost all of the flood peaks right with an NSE of 0.97 as shown in figure 3.13. The states show more of a step change for this catchment as seen in figure 3.14. A larger version can be found in appendix C.8.

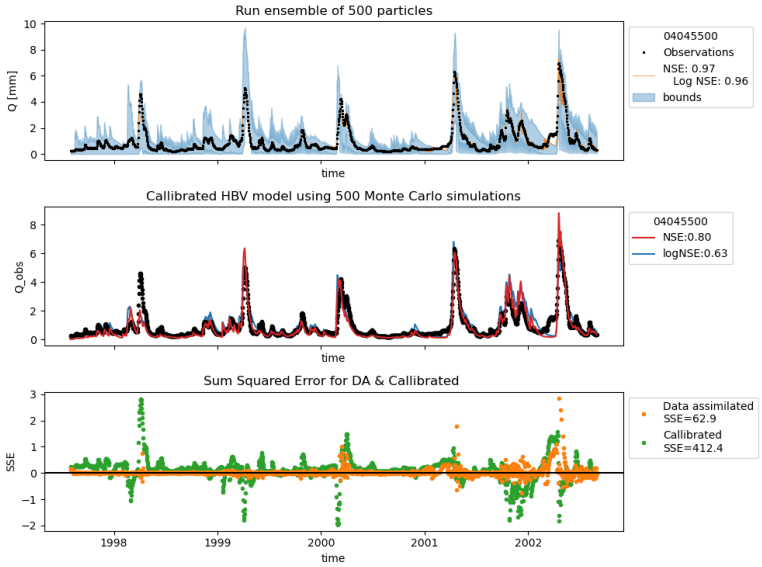



**Figure 3.13:** Hydrograph for data assimilation experiment, the baseline and both their sum squared error (SSE)

The increase in the slow storage is much less, however, the interception storage seems to increase as the flood peaks approach. Here the fast flow storage ($S_f$) then also increases strongly, combined with changes in $K_f$. This reinforces the theory shown in figure 2.12, where the assimilation scheme can control either of two paths in assimilating the observed streamflow. In the Tahquamenon River, we see that the main flood peaks happen at the start of the year as the snow melts. In 2001 and 2002 the calibrated model struggles to capture this. In both cases the data assimilation scheme increases the $\beta$ parameter, whilst decreasing the $Su_{max}$, meaning the soil holds less water and more is diverted to the fast flow reservoir. We see this filling up in the $S_f$ which holds over 200mm, similar to the storage size of the snow storage but much more than the precipitation peak of 20mm. At the same time, the $K_f$ value is very low, which increases with time as the flood peak increases. $T_{lag}$ drops down to one, meaning the computed output is not delayed, instead the outflow factor $K_f$ is adjusted to match the observed streamflow. The data assimilation scheme essentially changes the behaviour of the hydrological model to more accurately capture the flood peak. This increase in the slow and or fast flow reservoir was seen in several more experiments.

From the analysis of the catchment scale we see the adjustments in parameters and states, mainly concentrated around the flood peaks. In some catchments, these changes can be more clearly observed whilst in others making this distinction is more difficult. In the catchments analysed we see that the data assimilation scheme adjusts mainly the fast flow or runoff and slow flow or groundwater to account for the deficiencies the model has when capturing observed streamflow. This suggests the conceptual models are deficient in capturing all the processes during flood peaks.

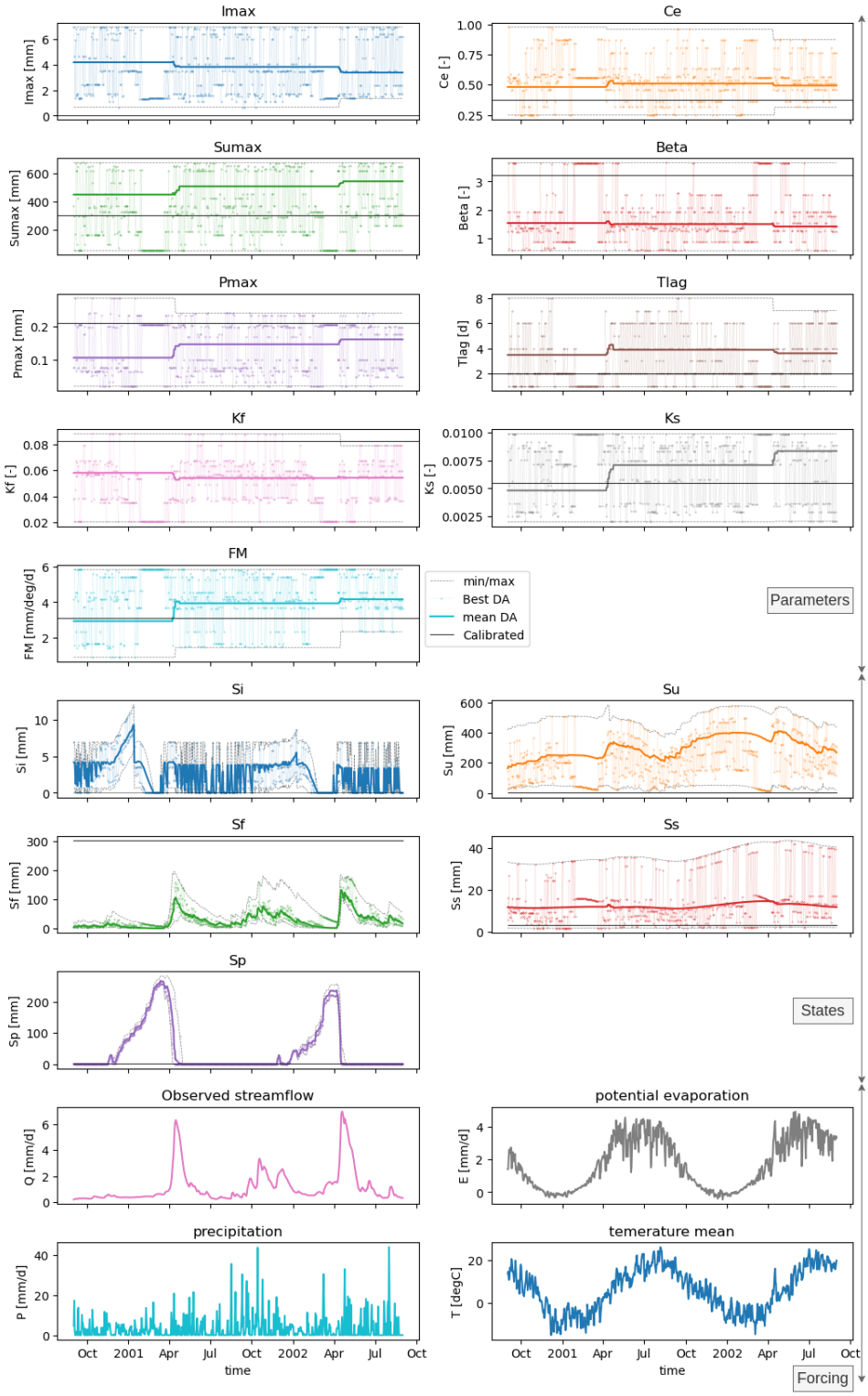



**Figure 3.14:** Parameters and states in the Tahquamenon River catchment - Michigan, Zoomed in on two flood peaks in 2001 and 2002. The nine parameters are: $I_{max}$, $C_e$, $Su_{max}$, β ,$P_{max}$, $T_{lag}$, $K_f$, $K_s$ and $FM$. The five states are: $S_i$, $S_u$, $S_f$ and $S_s$. The bottom row shows the observed stream flow and three forcing are: potential evaporation, precipitation and mean temperature. Dotted grey lines show the min and max, the grey line shows the reference calibrated model (for parameters), the solid coloured line shows the mean of the ensemble and the points show the ensemble member best predicting the observed streamflow.

# 4

# Conclusion, discussion and recommendations

## 4.1. Conclusion

Using the framework developed in a FAIR way as part of this thesis, data assimilation can be applied to any hydrological models using eWaterCycle. This framework was used to answer the research question: ”What do the adjustments in parameters and states imposed by data assimilation say about the deficiencies in the hydrological models of observed streamflow?”. Assimilating the HBV model using particle filtering allowed the comparison of data assimilation results to a calibrated HBV model run. This comparison was done for all of the 671 catchments in the CAMELS dataset. 461 of these catchments showed improvements compared to the calibrated model without data assimilation.

There is no clear connection between the magnitude of the calibrated parameter values and the improvement in model fit when using data assimilation. When comparing the catchment characteristics, there is some correlation between areas of similar properties and the improvements in model fit when applying data assimilation. The main catchment characteristics showing a correlation to the improvement in NSE are the climatic conditions and hydrological conditions. Comparing the daily mean discharge of each catchment to the improvement in NSE of that particular catchment shows that, with the hyperparameters used, catchments with a low mean discharge are favoured. The main reason for this is the large spread in the posterior distribution of modelled discharges the ensemble has in these catchments, making the selection of the best ensemble member trivial. The bias introduced by selecting the best ensemble member to calculate the NSE means no large-scale conclusions can be drawn about model deficiency. We do observe that catchments of similar characteristics improve similarly when data assimilation is applied to the HBV model compared to a calibrated model.

On a catchment level, the parameters and states can be analysed more closely. Analysing these in detail is unfeasible for 461 catchments. For this reason, three catchments are explored in depth in this thesis. All show step changes in parameters around flood peaks, often on the transition from or to base flow. We see that either the slow or fast flow storage reservoirs fill up to a much larger level than the baseline model in order to capture the flood peaks. The model then adjusts the outflow of the fast or slow flow storage to match the observed streamflow.

## 4.2. Discussion and recommendations

The framework developed in this thesis has been tested extensively with the HBV model and particle filtering. In addition, the Lorenz model was tested to check how a different model would behave. As the eWaterCycle models all use the Basic Model Interface, the data assimilation can be applied to any model that can be implemented in eWaterCycle. Currently, this has not been tested in detail. Before further integration into the eWaterCycle platform, this should be tested, at least with another 2D hydrological model. Adding more data assimilation schemes was tested by adding a Kalman Filter. The results the model produced were a bad fit, due to lack of time these were not improved upon. It did update the state vector, showing it is possible to other data assimilation schemes.

The experiments conducted show that it is possible to use a single combination of hyperparameters for a large number of catchments. The results show a strong relationship between the improvement the data assimilation scheme made compared to a baseline and both the mean discharge and the spread. Catchments with lower mean discharges, have more spread and thus improve more when data assimilation is applied. In the implementation used in this thesis, the mean discharge was not a factor in determining the variation in the amount of perturbation or jitter added after resampling. Using

the mean discharge as a factor in determining the amount of jitter added is likely a practical way to deal with this. This is often done in literature to the observational noise, for example by Vrugt et al. (2005), but not for the jitter of the state vector.

The method developed to add jitter to the state vector based on a two-year spin-up period is somewhat rudimentary. It estimates the variance of the posterior distribution using the maximum and minimum values of the posterior distribution. The shape of this distribution is more complex than a uniform distribution. Future research could look at improving upon this approach. A spin-up period was used to better grasp the scale of the parameters in each catchment. This allowed for catchment-specific knowledge without requiring user input. This is in my opinion a very useful approach to deal with state vector quantities of different units, strongly varying catchments but still using one hyperparameter set to control the whole experiment.

Throughout the thesis, a set of initial parameter values was used for calibration and as input for the data assimilation scheme. These are based on literature but can still vary strongly across catchments. For this reason, repeating the experiment with a wider initial parameter range could improve results, especially for catchments with higher discharges which currently were not captured well by the data assimilation scheme.

In the implementation used in this thesis, only one forcing data source was used for precipitation, evaporation and temperature. The CAMELS data set also provides two additional data sources. This makes it a fairly easy improvement to combine the three sources together. One-third of the particles can each receive the same data source. The particle filter will then choose particles from the best source at a given time.

The hyperparameter set used was optimised on 26 catchments. Repeating this optimisation for a larger number of catchments could help to improve the results. Also selecting catchments next to each other favoured this type of catchment, this also led to interesting results, but not to a general data assimilation experiment that can be applied to the whole CAMELS dataset. The discussion also remains on whether it is even feasible to select one set of hyperparameters for a very large range of catchments. Grouping catchments and optimising the parameters for a given type or characteristic would likely be more fruitful, as shown in this research.

In the results, the ensemble member with the highest weight was used for analysis. The mean gives a better representation of the ensemble as a whole. The data assimilation does still struggle to correctly capture the streamflow, thus using the highest weight ensemble member was a way to still analyse deficiencies. Using the highest weight ensemble member was a reasonable approach when compared to the best result of the calibration of the baseline model. However, for more confidence in the data assimilation experiments, in future research, the weighted mean could be a better metric as this includes information for the whole ensemble rather than just picking the best ensemble member at a given timestep. With adjustments in the hyperparameters and data assimilation scheme, this should be achievable.

In this thesis, the goodness of fit of the calibration period was used. This in itself has flaws, in hydrology a validation period following the calibration is more commonly used. During the validation period, the model is applied to unseen data. In future research of this type, the recommendation is to use the results of a validation period as a baseline to compare to.

As a final outlook, this thesis shows that applying data assimilation to hydrological models on a large set of catchments is possible. Data assimilation does have its roots in atmospheric models which are more chaotic than hydrological models. This implies that the state variable (e.g., temperature) in an ensemble of atmospheric models naturally spread more than the streamflow variable in hydrological models. The data assimilation algorithms and best practices must therefore be adjusted accordingly to ensure the same theory can be applied within hydrology.

# References


Ali, M. H., Popescu, I., Jonoski, A., & Solomatine, D. P. (2023). Remote Sensed and/or Global Datasets for Distributed Hydrological Modelling: A Review. *Remote Sensing*, *15*(6), 1642. https://doi.org/10.3390/rs15061642

Aubert, D., Loumagne, C., & Oudin, L. (2003). Sequential assimilation of soil moisture and streamflow data in a conceptual rainfall–runoff model. *Journal of Hydrology*, *280*(1), 145–161. https://doi.org/10.1016/S0022-1694(03)00229-4

Bárdossy, A. (2007). Calibration of hydrological model parameters for ungauged catchments. *Hydrology and Earth System Sciences*, *11*(2), 703–710. Retrieved December 1, 2023, from http://www.hydrol-earth-syst-sci.net/11/703/2007/hess-11-703-2007.pdf

Bergström, S. (1976). *Development and application of a conceptual runoff model for Scandinavian catchments*. Retrieved May 7, 2024, from https://urn.kb.se/resolve?urn=urn:nbn:se:smhi:diva-5738

Cudennec, C., Lins, H., Uhlenbrook, S., & Arheimer, B. (2020). Editorial – Towards FAIR and SQUARE hydrological data. *Hydrological Sciences Journal*, *65*(5), 681–682. https://doi.org/10.1080/02626667.2020.1739397

Daymet. (2024). Retrieved May 29, 2024, from https://daymet.ornl.gov/

Evensen, G., Raanes, P. N., Stordal, A. S., & Hove, J. (2019). Efficient Implementation of an Iterative Ensemble Smoother for Data Assimilation and Reservoir History Matching. *Frontiers in Applied Mathematics and Statistics*, *5*. https://doi.org/10.3389/fams.2019.00047

Evensen, G., Vossepoel, F. C., & van Leeuwen, P. J. (2022, April). *Data Assimilation Fundamentals* (1st ed.). Springer Cham. Retrieved March 6, 2024, from https://doi.org/10.1007/978-3-030-96709-3

Hall, C. A., Saia, S. M., Popp, A. L., Dogulu, N., Schymanski, S. J., Drost, N., van Emmerik, T., & Hut, R. (2022). A hydrologist's guide to open science. *Hydrology and Earth System Sciences*, *26*(3), 647–664. https://doi.org/10.5194/hess-26-647-2022

Hut, R., Drost, N., van de Giesen, N., van Werkhoven, B., Abdollahi, B., Aerts, J., Albers, T., Alidoost, F., Andela, B., Camphuijsen, J., Dzigan, Y., van Haren, R., Hutton, E., Kalverla, P., van Meersbergen, M., van den Oord, G., Pelupessy, I., Smeets, S., Verhoeven, S., … Weel, B. (2022). The eWaterCycle platform for open and FAIR hydrological collaboration. *Geoscientific Model Development*, *15*(13), 5371–5390. https://doi.org/10.5194/gmd-15-5371-2022

Hutton, E. W. h., Piper, M. D., & Tucker, G. E. (2020). The Basic Model Interface 2.0: A standard interface for coupling numerical models in the geosciences. *Journal of Open Source Software*, *5*(51), 2317. https://doi.org/10.21105/joss.02317

Kim, S. S. H., Marshall, L. A., Hughes, J. D., Sharma, A., & Vaze, J. (2021). Jointly Calibrating Hydrologic Model Parameters and State Adjustments. *Water Resources Research*, *57*(8). https://doi.org/10.1029/2020WR028499

Kirchner, J. W. (2006). Getting the right answers for the right reasons: Linking measurements, analyses, and models to advance the science of hydrology. *Water Resources Research*, *42*(3). https://doi.org/10.1029/2005WR004362

Kuptametee, C., & Aunsri, N. (2022). A review of resampling techniques in particle filtering framework. *Measurement*, *193*, 110836. https://doi.org/10.1016/j.measurement.2022.110836

Lindström, G., Johansson, B., Persson, M., Gardelin, M., & Bergström, S. (1997). Development and test of the distributed HBV-96 hydrological model. *Journal of Hydrology*, *201*(1), 272–288. https://doi.org/10.1016/S0022-1694(97)00041-3

Liu, Y., Weerts, A. H., Clark, M., Franssen, H.-J. H., Kumar, S., Moradkhani, H., Seo, D.-J., Schwanenberg, D., Smith, P., Dijk, A. I. J. M. v., Velzen, N. v., He, M., Lee, H., Noh, S. J., Rakovec, O., & Restrepo, P. (2012). Advancing data assimilation in operational hydrologic forecasting: Progresses, challenges, and emerging opportunities. *Hydrology and Earth System Sciences*, *16*(10), 3863–3887. https://doi.org/10.5194/hess-16-3863-2012

Lorenz, E. N. (1995). Predictability: A problem partly solved. Retrieved May 7, 2024, from https://www.ecmwf.int/en/elibrary/75462-predictability-problem-partly-solved

Maurer, E., Wood, A., Adam, J., Lettenmaier, D., & Nijssen, B. (2002). A Long-Term Hydrologically-Based Data Set of Land Surface Fluxes and States for the Conterminous United States. *15*, 3237–3251. Retrieved May 29, 2024, from https://www.engr.scu.edu/~emaurer/gridded_obs/index_gridded_obs.html

Moradkhani, H., Hsu, K.-L., Gupta, H., & Sorooshian, S. (2005). Uncertainty assessment of hydrologic model states and parameters: Sequential data assimilation using the particle filter. *Water Resources Research*, *41*(5). https://doi.org/10.1029/2004WR003604

Nash, J. E., & Sutcliffe, J. V. (1970). River flow forecasting through conceptual models part I — A discussion of principles. *Journal of Hydrology*, *10*(3), 282–290. https://doi.org/10.1016/0022-1694(70)90255-6

Newman, Andrew. (2014). A large-sample watershed-scale hydrometeorological dataset for the contiguous USA. https://doi.org/10.5065/D6MW2F4D

NLDAS. (2024). Retrieved May 29, 2024, from https://ldas.gsfc.nasa.gov/nldas

Oudin, L., Andréassian, V., Mathevet, T., Perrin, C., & Michel, C. (2006). Dynamic averaging of rainfall-runoff model simulations from complementary model parameterizations. *Water Resources Research*, *42*(7). https://doi.org/10.1029/2005WR004636

Piazzi, G., Thirel, G., Perrin, C., & Delaigue, O. (2021). Sequential Data Assimilation for Streamflow Forecasting: Assessing the Sensitivity to Uncertainties and Updated Variables of a Conceptual Hydrological Model at Basin Scale. *Water Resources Research*, *57*(4). https://doi.org/10.1029/2020WR028390

Rakovec, O., Weerts, A. H., Hazenberg, P., Torfs, P. J. J. F., & Uijlenhoet, R. (2012). State updating of a distributed hydrological model with Ensemble Kalman Filtering: Effects of updating frequency and observation network density on forecast accuracy. *Hydrology and Earth System Sciences*, *16*(9), 3435–3449. https://doi.org/10.5194/hess-16-3435-2012

Seibert, J. (1997). Estimation of Parameter Uncertainty in the HBV Model. *Hydrology Research*, *28*(4-5), 247–262. https://doi.org/10.2166/nh.1998.15

Seibert, J., & Bergström, S. (2022). A retrospective on hydrological catchment modelling based on half a century with the HBV model. *Hydrology and Earth System Sciences*, *26*(5), 1371–1388. https://doi.org/10.5194/hess-26-1371-2022

Seo, D.-J., Cajina, L., Corby, R., & Howieson, T. (2009). Automatic state updating for operational streamflow forecasting via variational data assimilation. *Journal of Hydrology*, *367*(3), 255–275. https://doi.org/10.1016/j.jhydrol.2009.01.019

SURF. (2023). Surf Research Cloud. Retrieved January 15, 2024, from https://portal.live.surfresearchcloud.nl/

van den Oord, G., Verhoeven, S., & Pelupessy, I. (2023, October). gRPC wrapper for model with a Basic modeling interface. Retrieved May 8, 2024, from https://zenodo.org/records/10053743

Van Leeuwen, P. J., Cheng, Y., & Reich, S. (2015). *Nonlinear Data Assimilation* (1st ed.). Springer International Publishing. Retrieved May 8, 2024, from https://doi.org/10.1007/978-3-319-18347-3

Varsi, A., Taylor, J., Kekempanos, L., Pyzer Knapp, E., & Maskell, S. (2020). A Fast Parallel Particle Filter for Shared Memory Systems. *IEEE Signal Processing Letters*, *27*, 1570–1574. https://doi.org/10.1109/LSP.2020.3014035

Vrugt, J. A., Diks, C. G. H., Gupta, H. V., Bouten, W., & Verstraten, J. M. (2005). Improved treatment of uncertainty in hydrologic modeling: Combining the strengths of global optimization and data assimilation. *Water Resources Research*, *41*(1). https://doi.org/10.1029/2004WR003059

Vrugt, J. A., ter Braak, C. J. F., Diks, C. G. H., & Schoups, G. (2013). Hydrologic data assimilation using particle Markov chain Monte Carlo simulation: Theory, concepts and applications. *Advances in Water Resources*, *51*, 457–478. https://doi.org/10.1016/j.advwatres.2012.04.002

Weerts, A. H., & El Serafy, G. Y. H. (2006). Particle filtering and ensemble Kalman filtering for state updating with hydrological conceptual rainfall-runoff models. *Water Resources Research*, *42*(9). https://doi.org/10.1029/2005WR004093

Wood, E. F., Roundy, J. K., Troy, T. J., van Beek, L. P. H., Bierkens, M. F. P., Blyth, E., de Roo, A., Döll, P., Ek, M., Famiglietti, J., Gochis, D., van de Giesen, N., Houser, P., Jaffé, P. R., Kollet, S., Lehner, B., Lettenmaier, D. P., Peters-Lidard, C., Sivapalan, M., … Whitehead, P. (2011). Hyperresolution global land surface modeling: Meeting a grand challenge for monitoring Earth's terrestrial water. *Water Resources Research*, *47*(5). https://doi.org/10.1029/2010WR010090

# A

# Appendix - Research flow diagrams

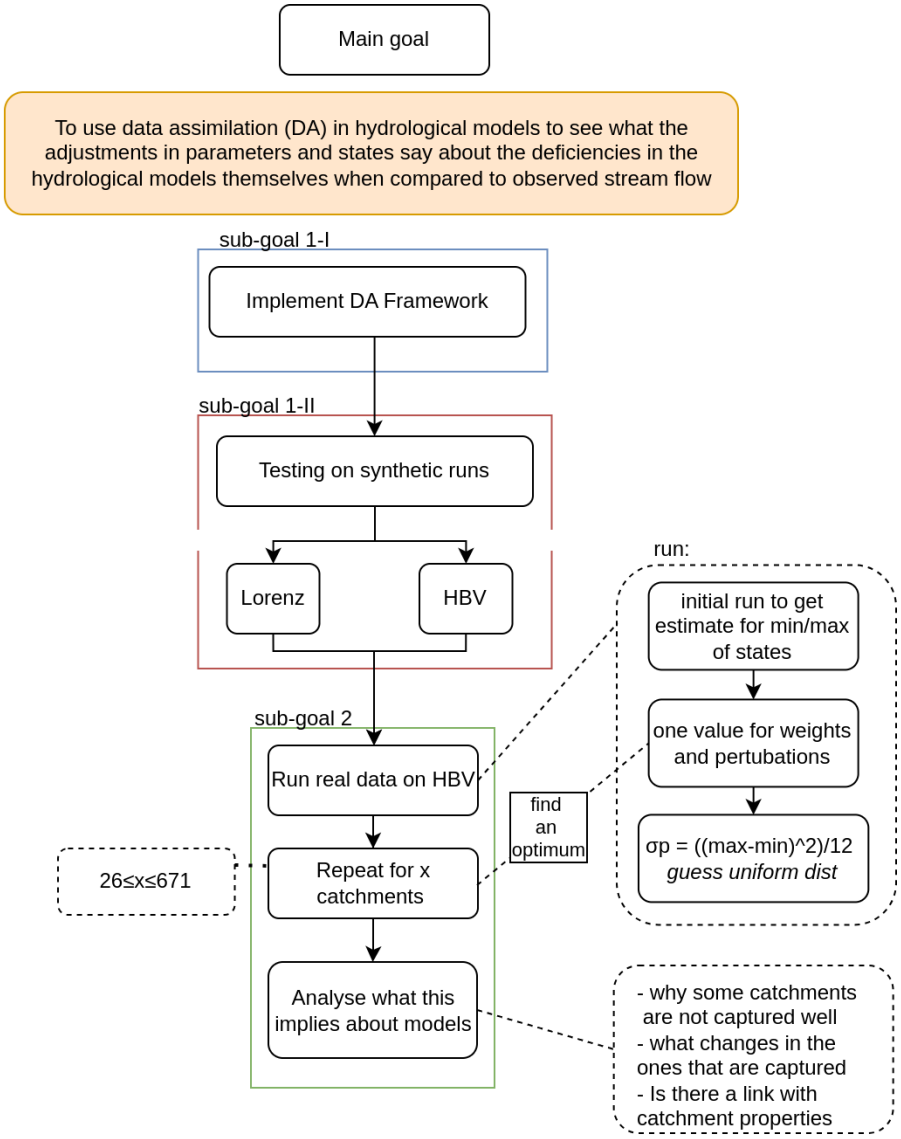



**Figure A.1:** Flow chart showing the goals of this thesis

# B

# Appendix - Data assimilation structure diagrams

Two design options were made to explore what the options were for the framework, these are found in figures B.1 and B.2. To allow for flexibility and adjust ability, the centralised approach was implemented. The decentralised approach can be useful if the state vector is very large. It would also allow a ensemble member to live on a different machine allowing for very large ensemble run on large models. As the scope of eWaterCycle is not this big yet, this is not a consideration to be made. The existing framework could be adapted in future to still allow for this.

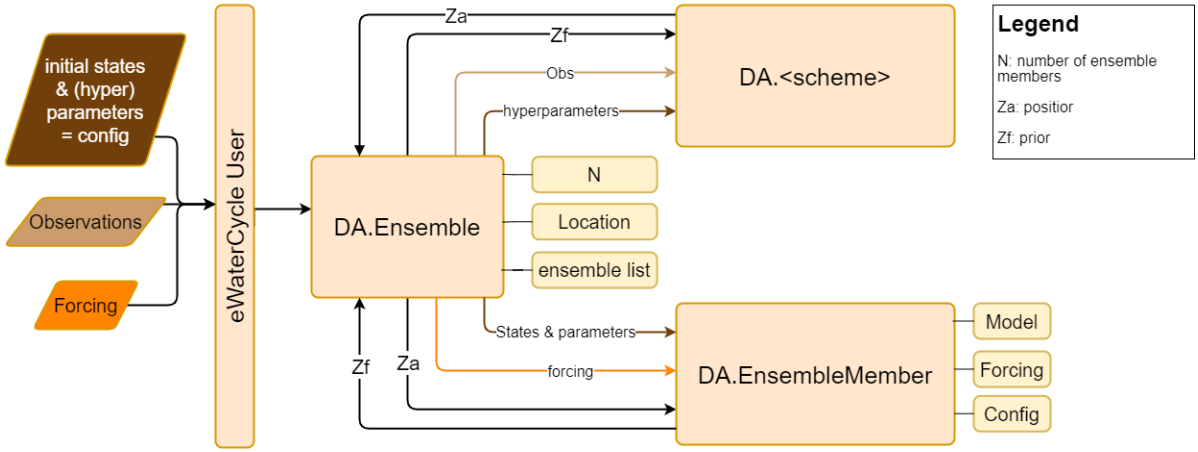



**Figure B.1:** Centralised option for data assimilation framework

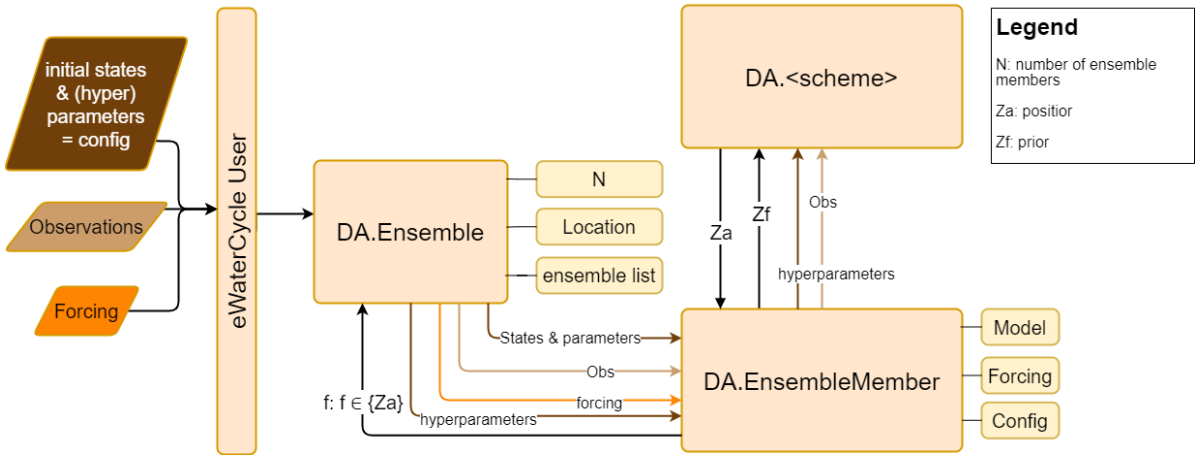



**Figure B.2:** Decentralised design for data assimilation framework

Figures B.3 through B.7 outline how different techniques operate. To make these, pseudo code and explanations from Evensen et al., 2022 to understand the concepts at a high level. Figure B.3 contains more detail where the framework sits in relation to the experiments conducted. .

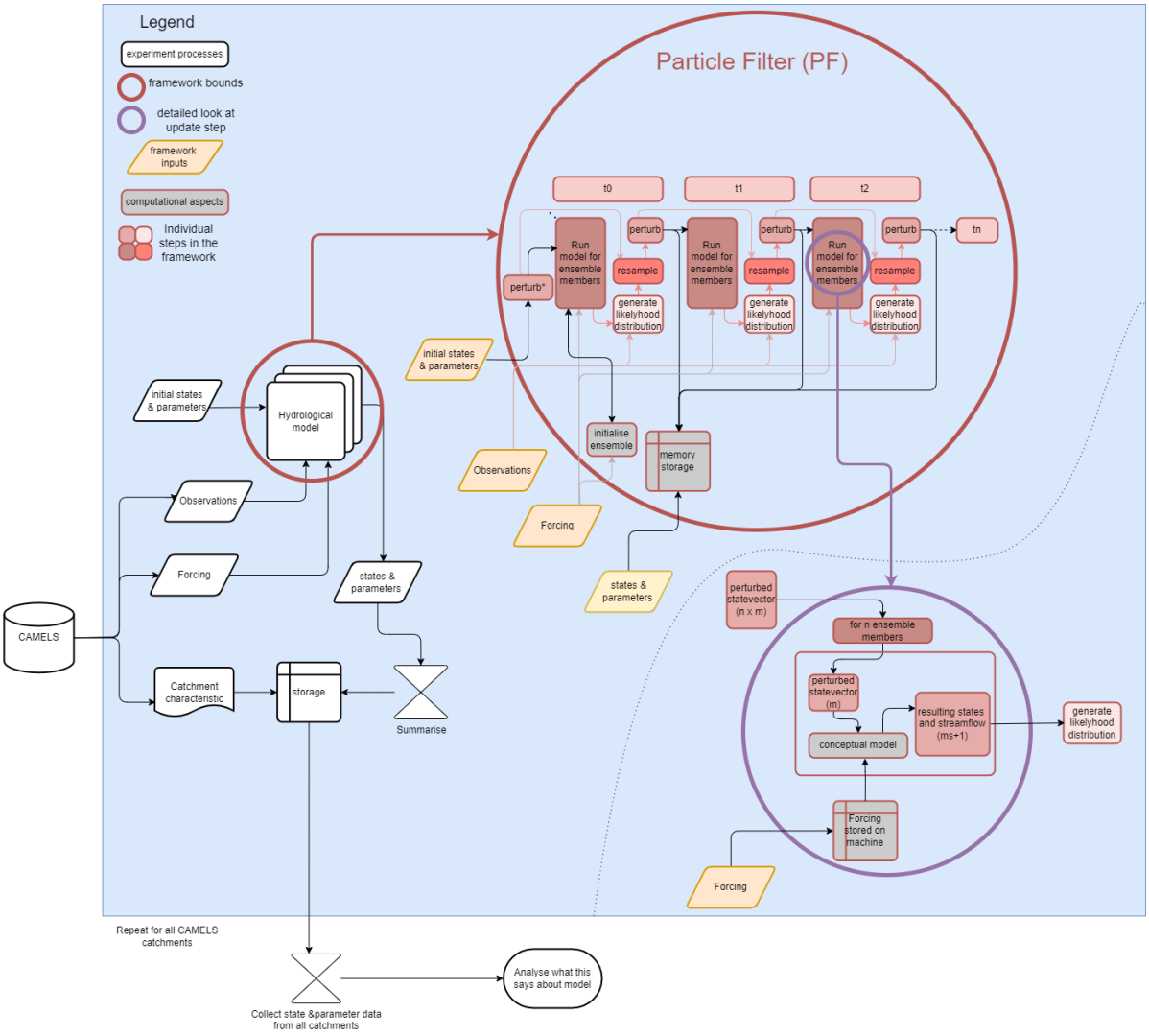



**Figure B.3:** Flow chart showing data flow of a Particle Filter (PF)

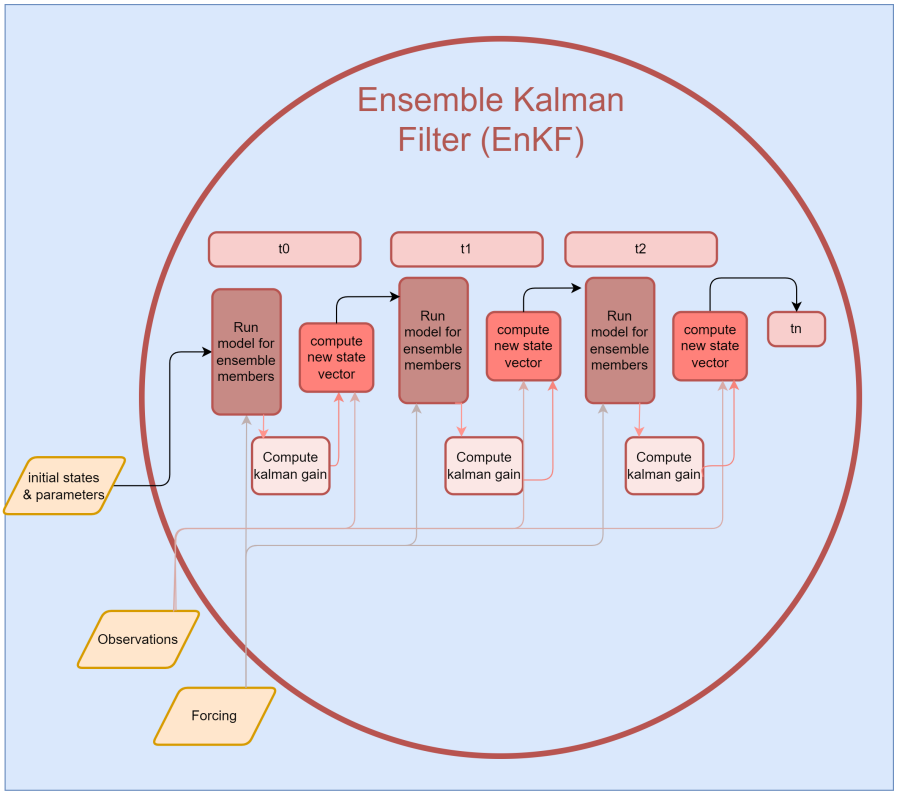



**Figure B.4:** Flow chart showing data flow of an Ensemble Kalman Filter (EnKF)

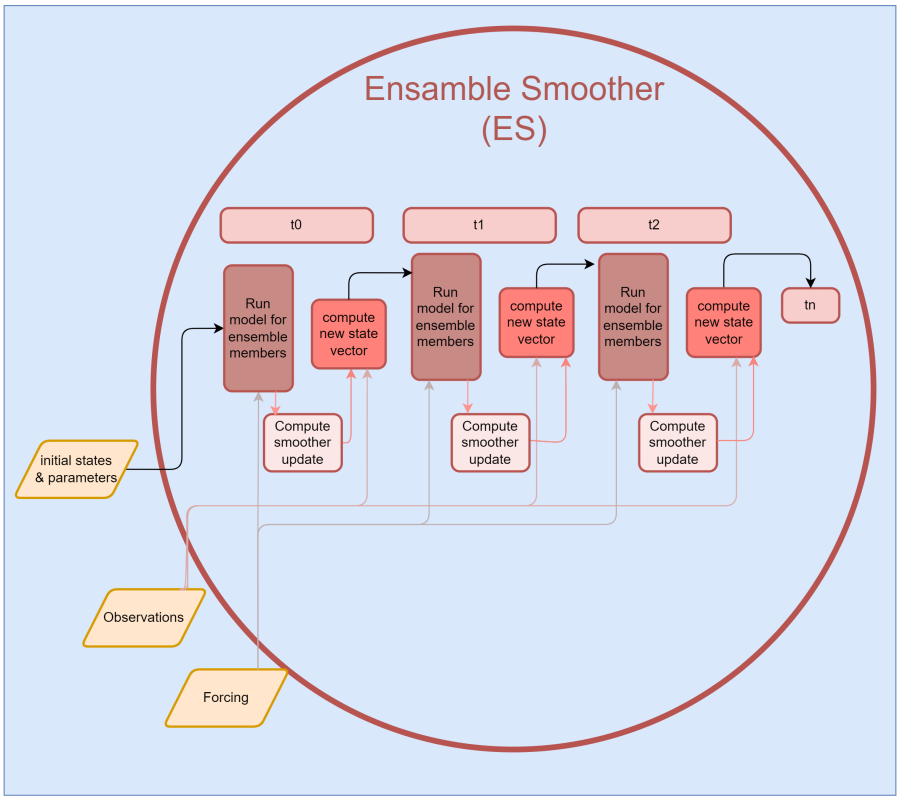



**Figure B.5:** Flow chart showing data flow of an Ensemble Smoother (ES)

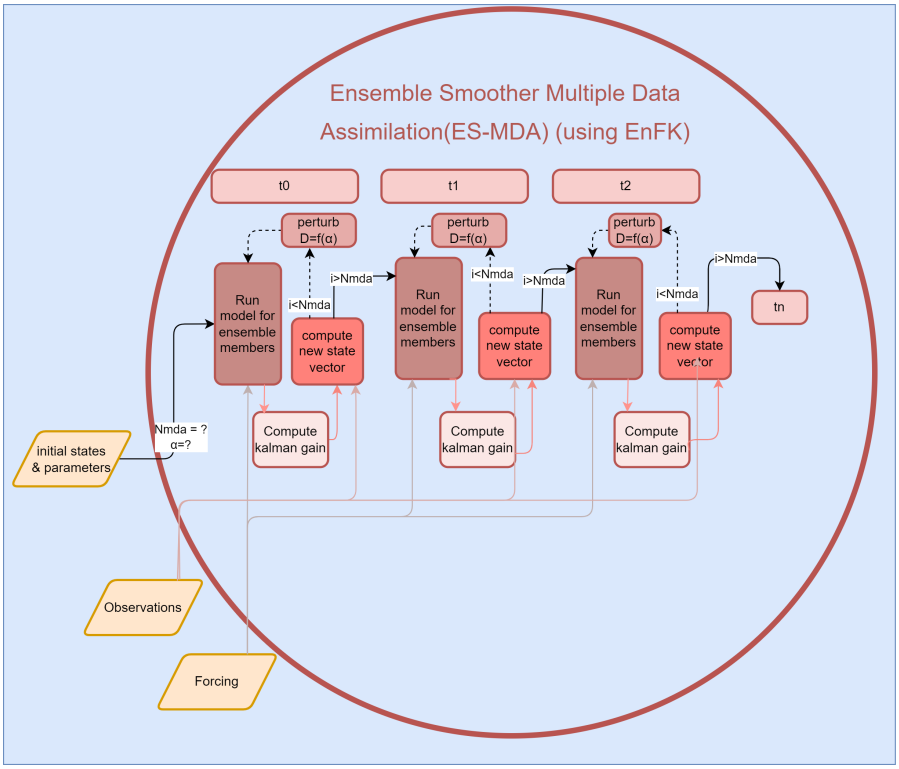



**Figure B.6:** Flow chart showing data flow of an Ensemble Smoother Multiple Data Assimilation (ES-MDA)

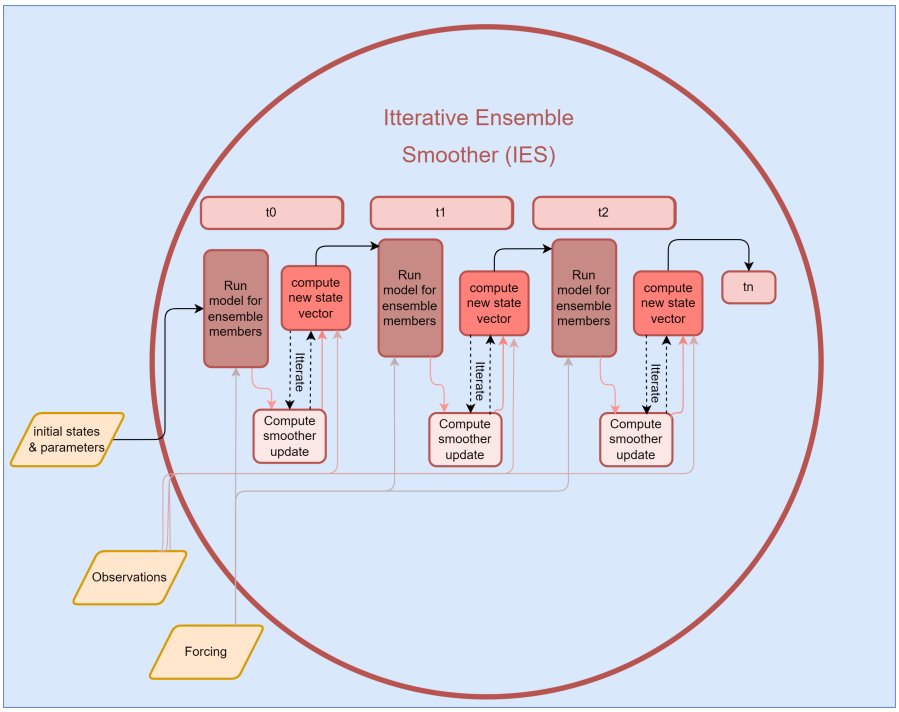



**Figure B.7:** Flow chart showing data flow of an Iterative Ensemble Smoother (IES)

# C

# Appendix - results

## C.1. Comparison of NSE to catchment characteristics

Including and analysing all the 57 characteristics in the CAMELS dataset is not feasible. A few more than those shown in the results chapter are given here. These extra plots demonstrate that mainly the vegetation, climatic and hydrological signatures show a correlation between the improvements. Geological and soil characteristics of catchments show no relation if any. Topographic do show some correlation: in particular elevation and longitude (which can also be seen on the map), but area does not for example.

The resulting improved NSE's from the experiment still have a large spread (figure 3.6). If the data assimilation experiment captures the observed streamflow better than the baseline experiment, this could indicate a deficiency in the model. The difference in NSE between the data assimilation and the baseline experiment can be calculated. This is the ΔNSE. In 461 catchments the ΔNSE is greater than zero, meaning the data assimilation experiment has a higher NSE than that of the baseline model experiment. A high ΔNSE indicates that the data assimilation scheme made a large improvement to the modelled streamflow. In figure C.1 the ΔNSE is compared to the different model parameters in the corresponding baseline experiment. Each dot represents a catchment with the y-axis showing the ΔNSE and the x-axis showing one of the corresponding parameters in the model.

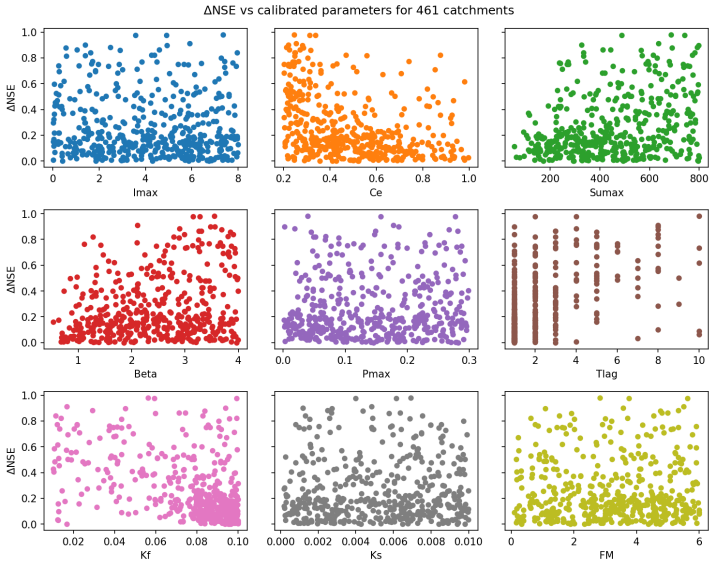
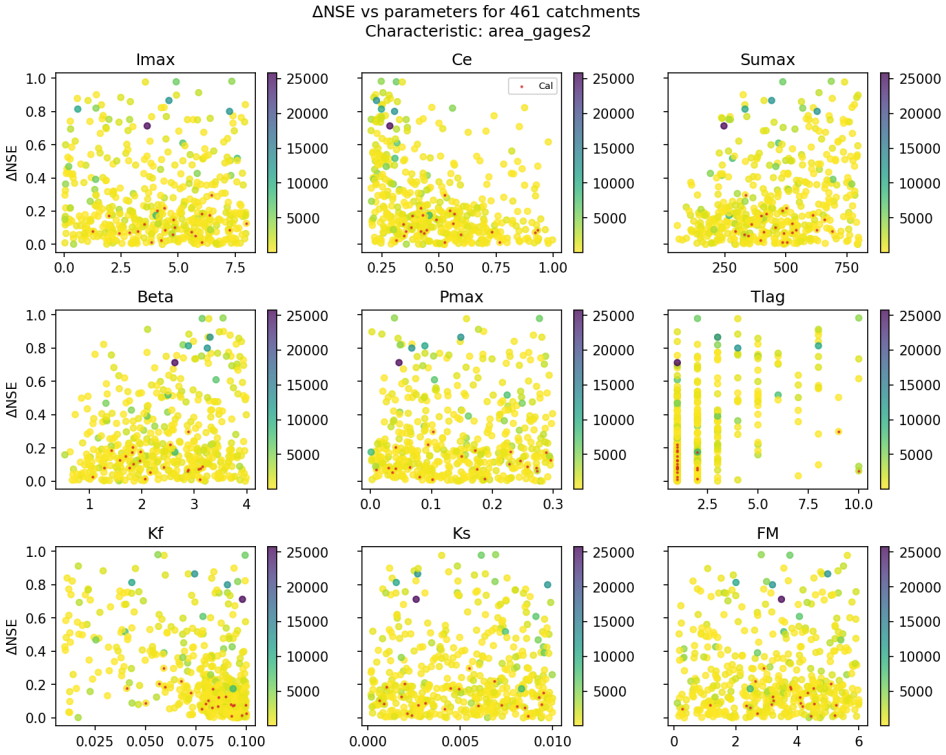
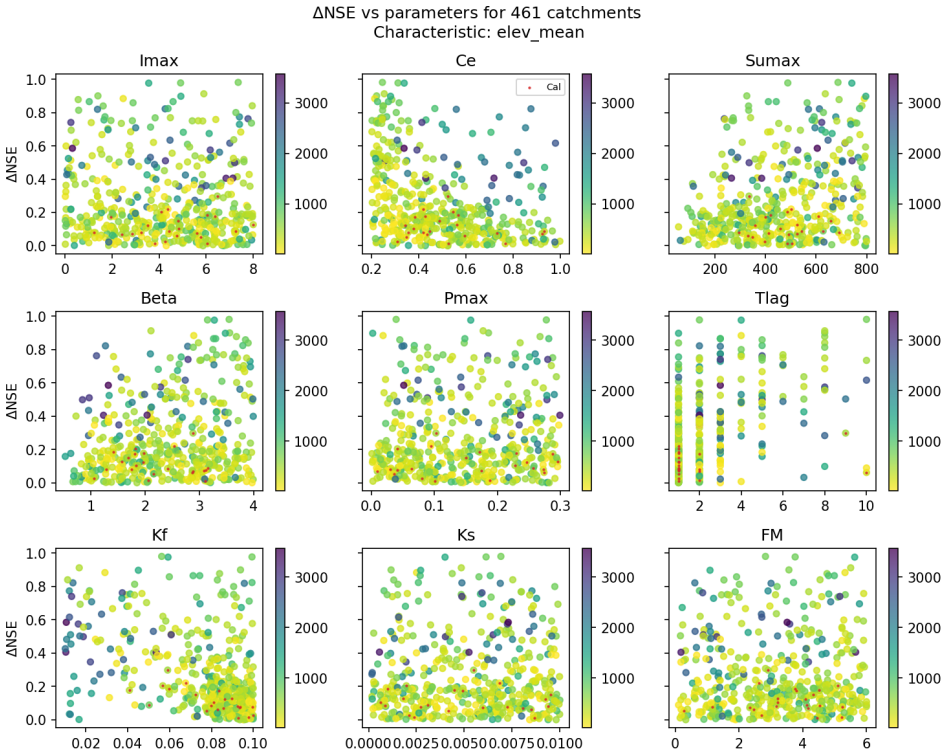



**Figure C.1:** ΔNSE compared to the calibrated parameter values for 461 catchments

The parameters $Ce$, $Su_{max}$ and $\beta$ which are responsible for parts of the soil processes in the model show a vague trend. This is not a clear enough relationship to draw any conclusions but does show some insight. At $K_f$ values of between 0.08 and 0.1 we can see some clustering between ΔNSE values of 0 and 0.2. Note that the distribution of $T_{lag}$ is because this is the lag in days the model applies and must be an integer. The $T_{lag}$ values are skewed to the lower end, but show no general pattern. The values of $I_{max}$, $P_{max}$, $K_s$ and $FM$ show no particular trend other than a concentration of values between ΔNSE 0 and 0.2. This can also be observed in figure 3.6, where most of the observations lie fairly close to the dotted line.

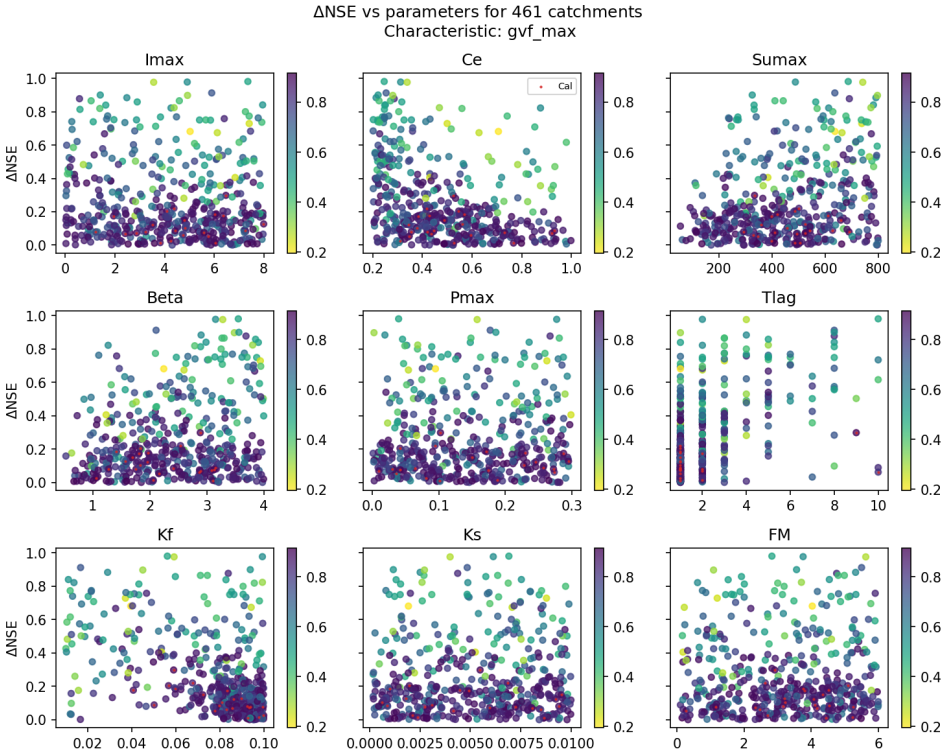



**(a)** ΔNSE compared to the calibrated parameters and catchment area

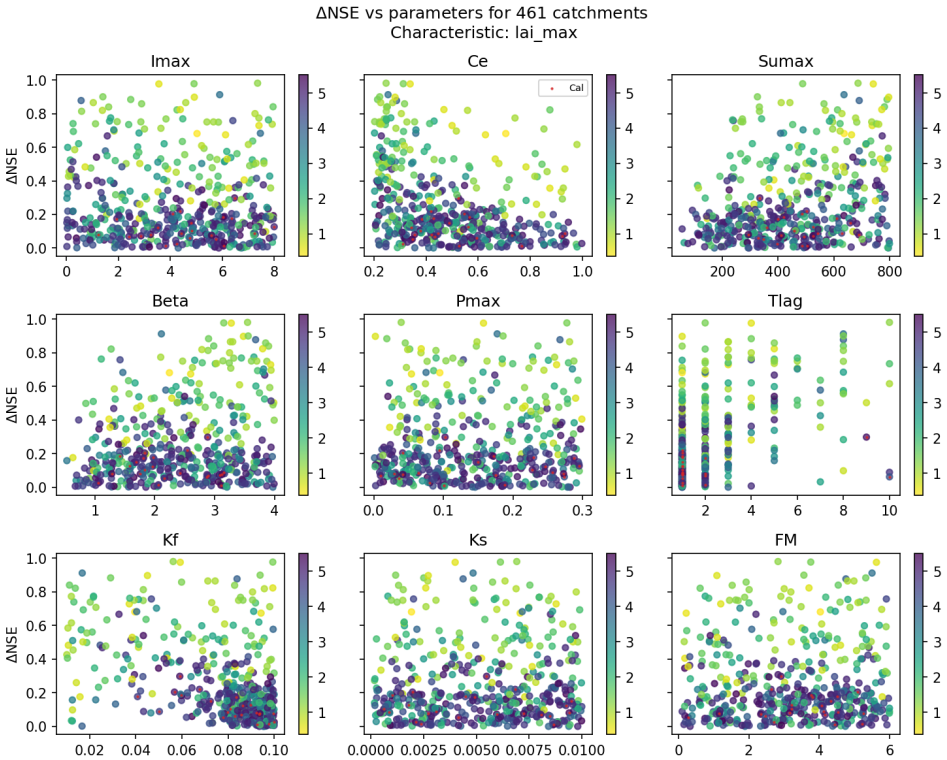



**(b)** ΔNSE compared to the calibrated parameters and mean elevation

**Figure C.2:** Topographic: no correlation on area, correlation with Kf on mean elevation

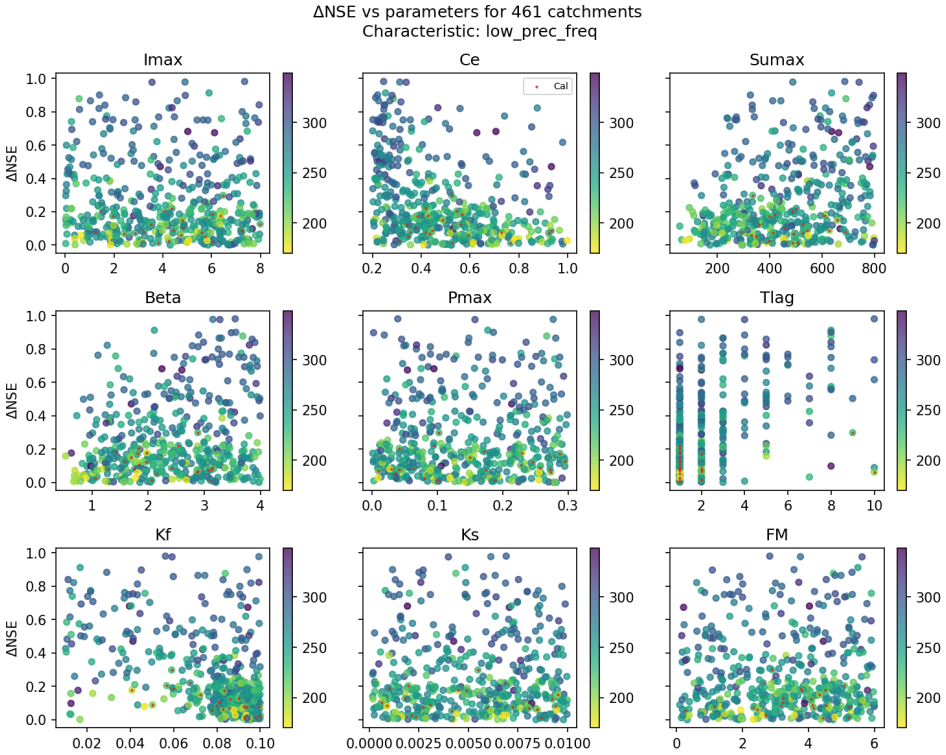



**(a)** ΔNSE compared to the calibrated parameters and green vegetation fraction

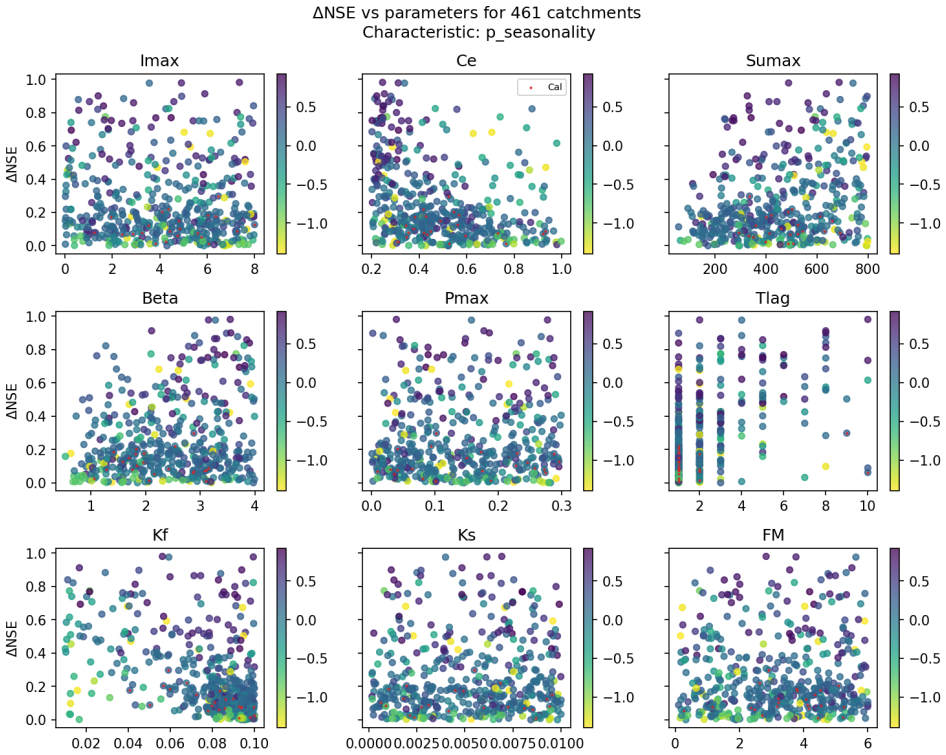



**(b)** ΔNSE compared to the calibrated parameters and lead area index

**Figure C.3:** Vegetation: clear correlation

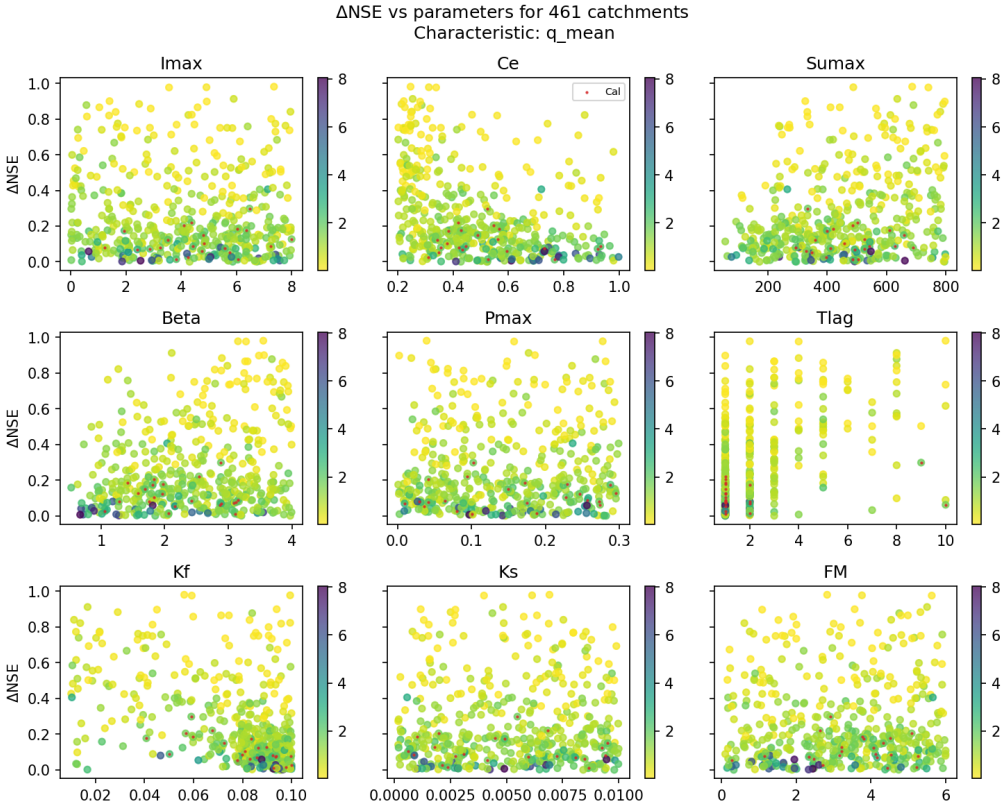



**(a)** ΔNSE compared to the calibrated parameters and low precipitation frequency: frequency of dry days, defined as less than 1 mm/d

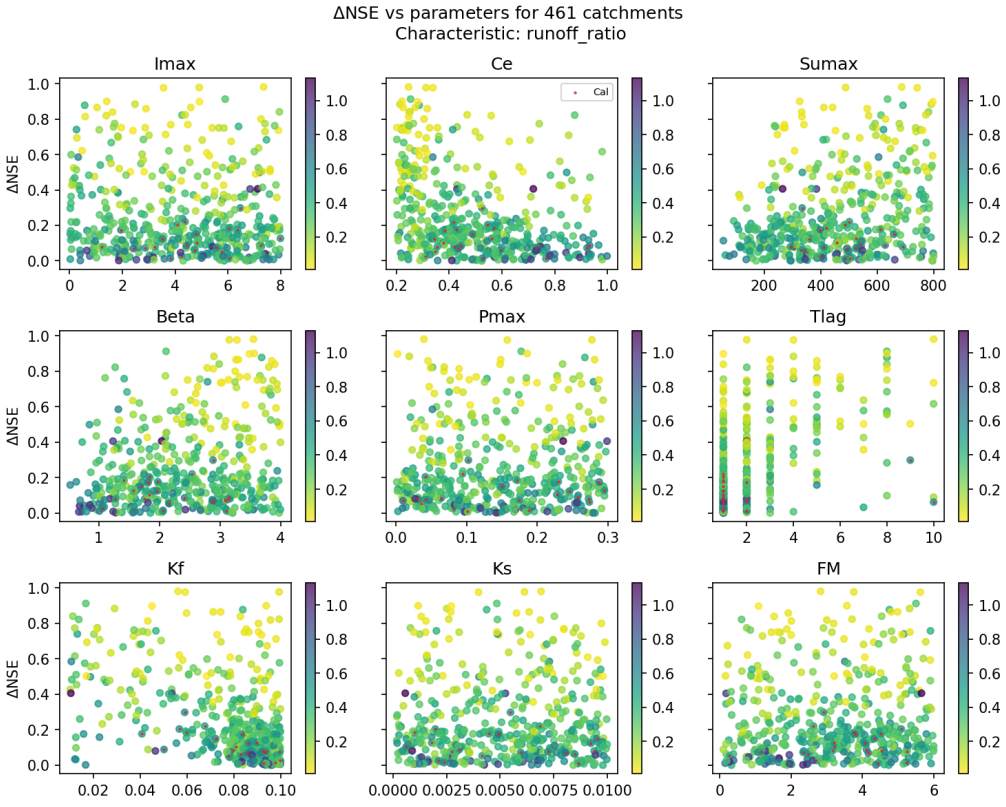



**(b)** ΔNSE compared to the calibrated parameters and p-seasonality: seasonality and timing of precipitation

**Figure C.4:** Climatic: clear correlation

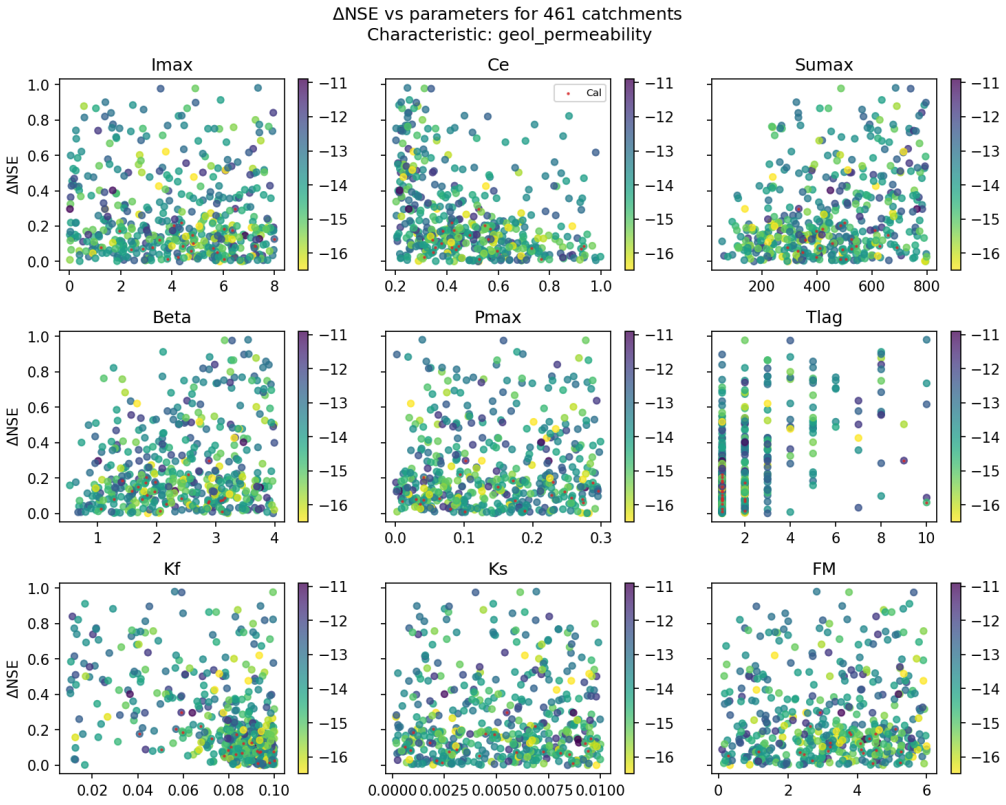



**(a)** ΔNSE compared to the calibrated parameters and mean discharge

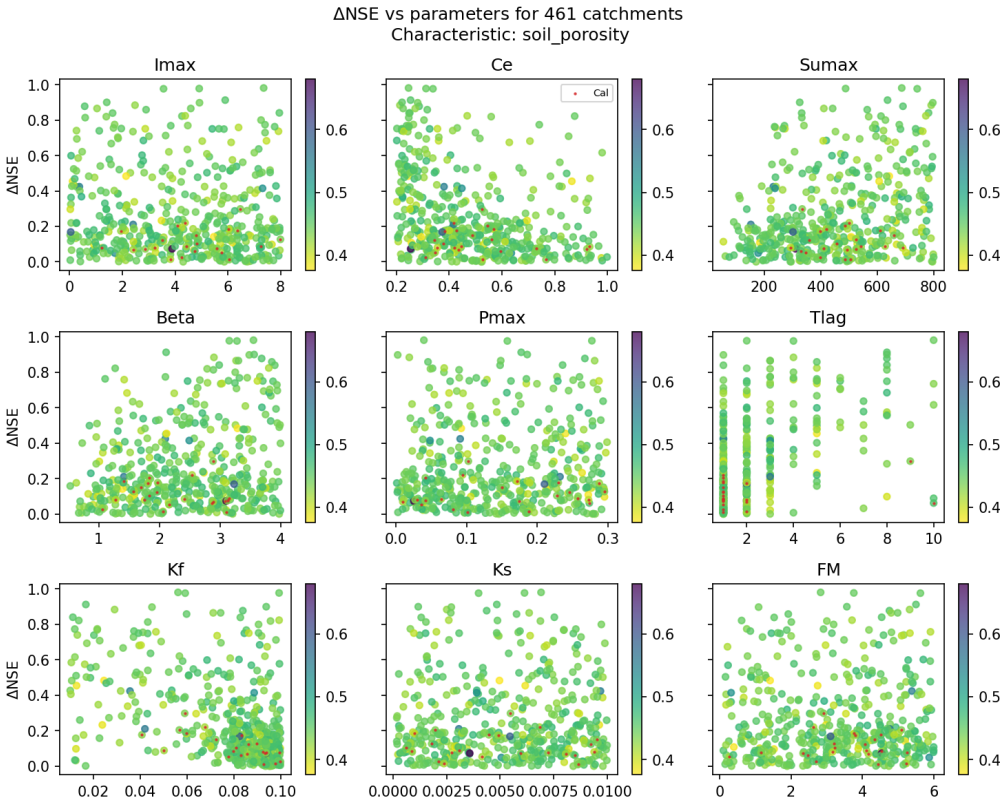



**(b)** ΔNSE compared to the calibrated parameters and runoff ratio of mean daily discharge to daily precipitation

**Figure C.5:** Hydrological: clear correlation




**(a)** ΔNSE compared to the calibrated parameters and geological permeability




**(b)** ΔNSE compared to the calibrated parameters and soil porosity

**Figure C.6:** Soil & geological: the characteristics showed no correlation to the parameters

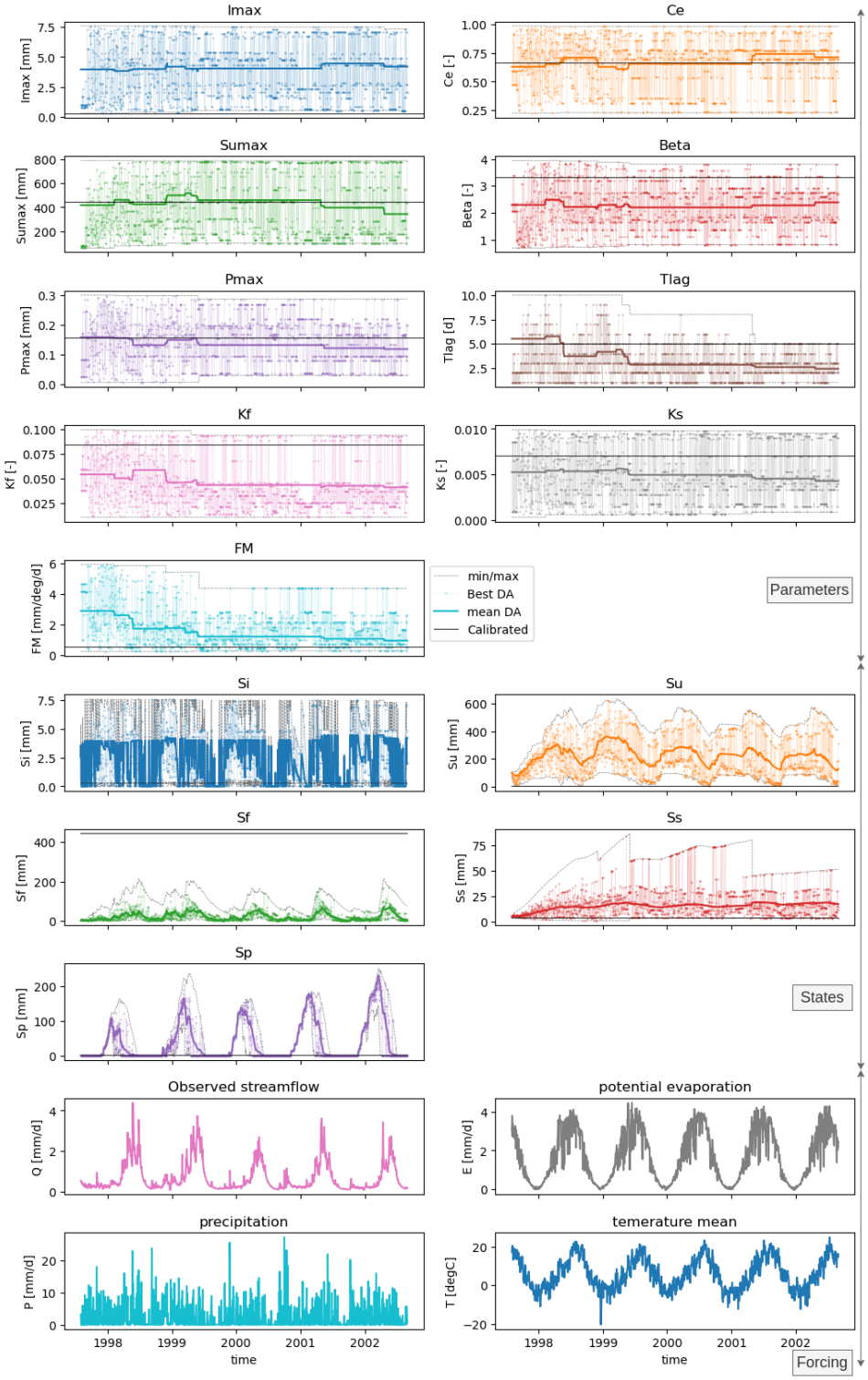



**Figure C.7:** Parameters and states with time for 5 years in SF Clearwater river catchment - Idaho (133338500). The nine parameters are: $I_{max}$, $C_e$, $Su_{max}$, β ,$P_{max}$, $T_{lag}$, $K_f$, $K_s$ and $FM$. The five states are: $S_i$, $S_u$, $S_f$ and $S_s$. The bottom row shows the observed stream flow and three forcing are: potential evaporation, precipitation and mean temperature. Dotted grey lines show the min and max, grey line shows the reference calibrated model (for parameters), solid colored line shows the mean of the ensemble and the points show the ensemble member best predicting the observed streamflow.

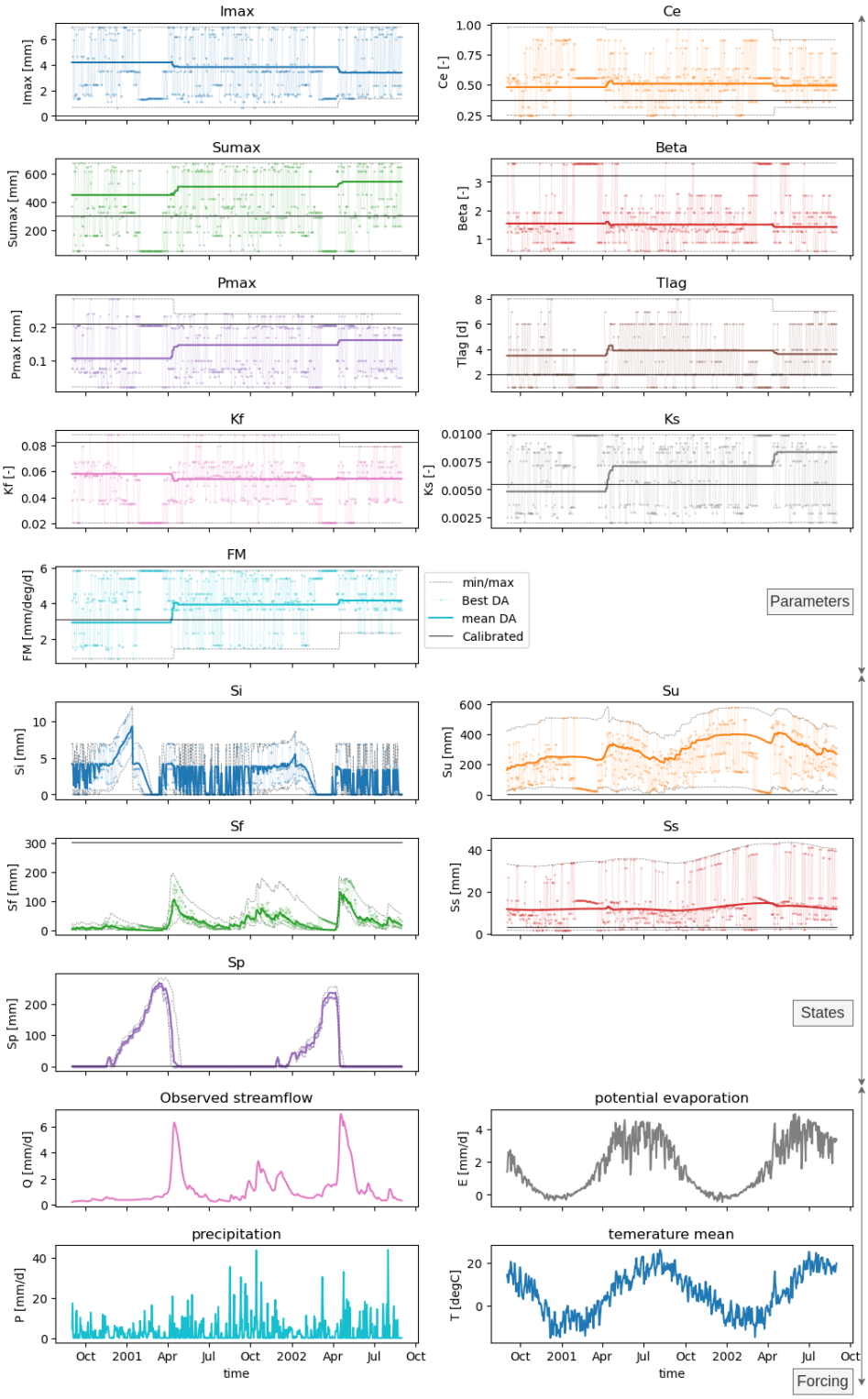



**Figure C.8:** Parameters and states, zoomed in on two flood peaks in 2001 and 2002 for Tahquamenon River - Michigan catchment (04045500). The nine parameters are: $I_{max}$, $C_e$, $Su_{max}$, β ,$P_{max}$, $T_{lag}$, $K_f$, $K_s$ and $FM$. The five states are: $S_i$, $S_u$, $S_f$ and $S_s$. The bottom row shows the observed stream flow and three forcing are: potential evaporation, precipitation and mean temperature. Dotted grey lines show the min and max, grey line shows the reference calibrated model (for parameters), solid colored line shows the mean of the ensemble and the points show the ensemble member best predicting the observed streamflow.

# D

# Appendix - Notebooks

In the writing of this thesis Python code has been developed to carry out the research tasks. This has all been done in a FAIR way, mainly in annotated Jupyter notebooks. A collection of notebooks has been added to demonstrate this. All the notebooks displayed here and more can also be found at https://github.com/Daafip/Msc-Thesis-Notebooks. All the code used in generating the results can also be found there.

- The Data assimilation framework can be found at https://github.com/Daafip/eWaterCycle-DA. This framework can be used for different use cases. Three examples of running DA are added in section D.1:
  - A classical example where all (hyper)parameters are defined beforehand in D.1.1
  - On the fly data assimilation where the (hyper)parameter for the data assimilation experiment are added after the model has already run. D.1.2
  - A calibration run for a hydrological model saving all the states and parameters specified. D.1.3
- The Lorenz model added to eWaterCycle can be found at with accompanying https://github.com/Daafip/ewatercycle-lorenz found in section D.2
- The Carvan dataset was made accessible in eWaterCycle. An interactive map showcasing the catchments can be found at www.ewatercycle.org/caravan-map/. The pull request can be found at https://github.com/eWaterCycle/ewatercycle/pull/407. An example notebook is given in section D.3.

## D.1. Data assimilation framework

For the purpose of the demonstration the HBV model has used.

### D.1.1. Classical data asssimilation

This is the classical or linear approach to run a data assimilation scheme. Notebook on the next page.

# Data assimilation Classical - HBV model with Particle filter

June 12, 2024

## 1 a Classical Data assimilation on a HBV model using a Particle Filter

[1]:
```
import warnings, glob
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import xarray as xr
from tqdm import tqdm
```

[2]:
```
import ewatercycle
import ewatercycle.forcing
import ewatercycle.models
```

Download plugin model

Ensure the HBV model is loaded and the Data Assimilation model is present:

```
pip install ewatercycle-HBV
pip install ewatercycle-DA
```

This notebook uses the local HBV model

`pip install HBV`

If instead you want to use the docker model, use:

`from ewatercycle.models import HBV`

and replace `HBVLocal` with `HBV`

**set up paths**

[3]:
```
path = Path.cwd()
forcing_path = path / "Forcing"
camels_path= path / "Camels"
observations_path = path / "Observations"
figure_path = path / "Figures"
output_path = path / "Output"
```

```
for path in [forcing_path, observations_path, figure_path, output_path,camels_path]:
    path.mkdir(exist_ok=True)
```

[4]:
```
experiment_start_date = "1997-08-01T00:00:00Z"
experiment_end_date = "2000-09-01T00:00:00Z"
HRU_id = 1620500
alpha = 1.26
```

[5]:
```
from ewatercycle.forcing import sources
```

This forcing file is availible at https://github.com/Daafip/Msc-Thesis-Notebooks

[6]:
```
camels_forcing = sources.HBVForcing(start_time = experiment_start_date,
                            end_time = experiment_end_date,
                            directory = forcing_path,
                            camels_file = f'0{HRU_id}_lump_cida_forcing_leap.txt',
                            alpha = alpha,
                            )
```

**import model**

[7]:
```
from ewatercycle_DA.local_models.HBV import HBVLocal
```

[8]:
```
truth_model = HBVLocal(forcing=camels_forcing)
```

[9]:
```
truth_parameters = np.array([3.2,   0.9,  700,    1.2,   .16,   4,      .08,  .0075, 0.5])
truth_initial_storage = np.array([20,  10,  50,  100, 10])
```

[10]:
```
config_file, _ = truth_model.setup(
            parameters=','.join([str(p) for p in truth_parameters]),
            initial_storage=','.join([str(s) for s in truth_initial_storage]),
           )
```

[11]:
```
truth_model.initialize(config_file)
```

[12]:
```
Q_m = []
time = []
while truth_model.time < truth_model.end_time:
    truth_model.update()
    Q_m.append(truth_model.get_value("Q"))
    time.append(truth_model.time_as_datetime.date())
truth_model.finalize()
```

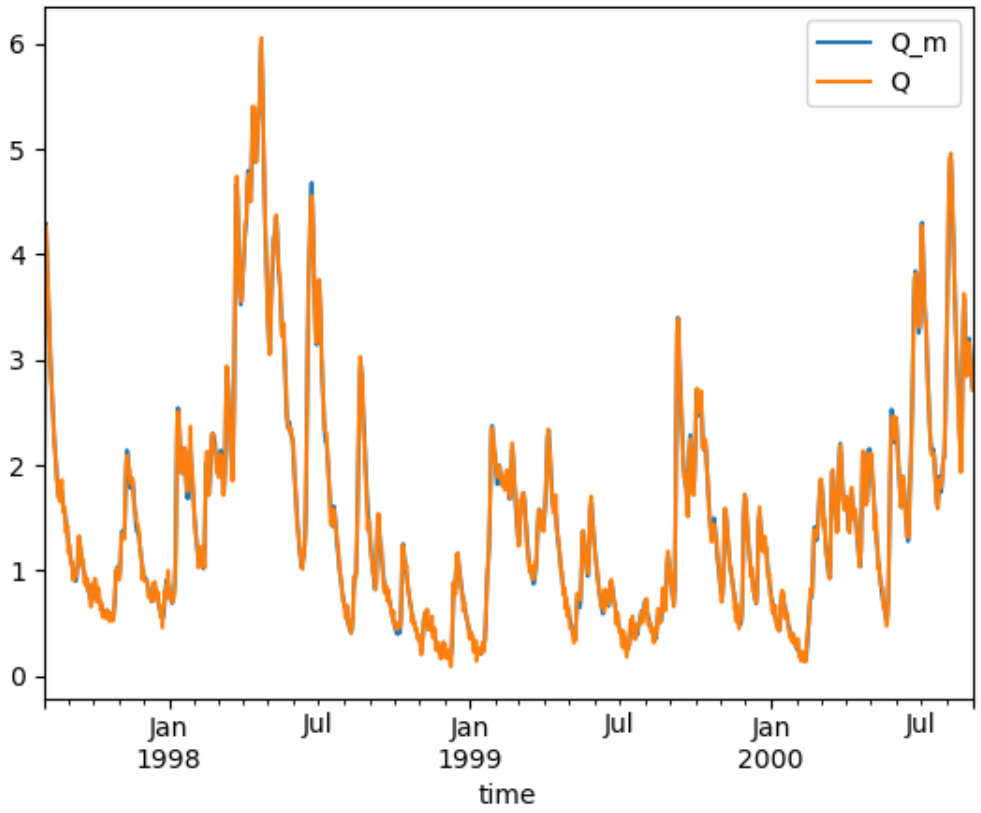
[13]:
```
truth = pd.DataFrame(Q_m, columns=["Q_m"],index=time)
```

[14]:
```
rng = np.random.default_rng()  # Initiate a Random Number Generator
def add_normal_noise(like_sigma) -> float:
    """Normal (zero-mean) noise to be added to a state"""
    return rng.normal(loc=0, scale=like_sigma)
```

[15]:
```
truth['Q'] = truth.apply(lambda x: x.Q_m + add_normal_noise(0.05) , axis=1)
truth.index = truth.apply(lambda x: pd.Timestamp(x.name),axis=1)
truth.index.name = "time"
```

[16]:
```
truth.plot()
```

[16]:
```
<Axes: xlabel='time'>
```




[17]:
```
ds_obs = xr.Dataset(data_vars=truth[['Q']])
```

[18]:
```
current_time = str(datetime.now())[:-10].replace(":","_")
ds_obs_dir = observations_path / f'truth_model_HBV_{current_time}.nc'
if not ds_obs_dir.is_file():
    ds_obs.to_netcdf(ds_obs_dir)
```

**import DA function:**

[19]:
```
from ewatercycle_DA import DA
```

[20]:
```
n_particles = 150
```

[21]:
```
ensemble = DA.Ensemble(N=n_particles)
ensemble.setup()
```

[22]:
```
# ## Array of initial storage terms - we keep these constant for now
# ##                Si,  Su, Sf, Ss
s_0 = np.array([0,  100,  0,  5, 0])
## Array of parameters min/max bounds as a reference
##                        Imax,  Ce,  Sumax, beta,  Pmax,  T_lag,   Kf,   Ks, FM
p_min_initial= np.array([0,   0.2,  40,    .5,   .001,   1,     .01,  .0001, 6])
p_max_initial = np.array([8,    1,  800,   4,    .3,     10,    .1,   .01, 0.1])
p_names = ["$I_{max}$",  "$C_e$",  "$Su_{max}$", " ",  "$P_{max}$",
 ↪"$T_{lag}$",   "$K_f$",   "$K_s$", "FM"]
S_names = ["Interception storage", "Unsaturated Rootzone Storage", "Fastflow
 ↪storage", "Groundwater storage", "Snowpack storage"]
param_names = ["Imax","Ce",  "Sumax", "Beta",  "Pmax",  "Tlag",   "Kf",   "Ks",
 ↪"FM"]
stor_names = ["Si", "Su", "Sf", "Ss", "Sp"]
par_0 = (p_min_initial + p_max_initial)/2 # # set initial as mean of max,min
```

[23]:
```
array_random_num = np.array([[np.random.random() for i in
 ↪range(len(p_max_initial))] for i in range(n_particles)])
p_intial = p_min_initial + array_random_num * (p_max_initial-p_min_initial)
```

[24]:
```
# values which you set
setup_kwargs_lst = []
for index in range(n_particles):
    setup_kwargs_lst.append({'parameters':','.join([str(p) for p in
 ↪p_intial[index]]),
                             'initial_storage':','.join([str(s) for s in s_0]),
                              })
```

[25]:
```
from ewatercycle_DA.local_models.HBV import HBVLocal
```

[26]:
```
# this initializes the models for all ensemble members.
ensemble.initialize(model_name=["HBVLocal"]*n_particles,
                    forcing=[camels_forcing]*n_particles,
                    setup_kwargs=setup_kwargs_lst)
```

## 1.1 setup DA

This sets up all the require data assimilation information

[27]:
```
lst_like_sigma = [0.005] * 14 + [0]
hyper_parameters = {'like_sigma_weights' : 0.05,
                    'like_sigma_state_vector' : lst_like_sigma,
                    'f_n_particles': 1}
```

[28]:
```
def H(Z):
    if len(Z) == 15:
        return Z[-1]
    else:
        raise SyntaxWarning(f"Length of statevector should be 13 but is␣
 ↪{len(Z)}")
```

[29]:
```
ensemble.initialize_da_method(ensemble_method_name = "PF",
                              hyper_parameters=hyper_parameters,
                              state_vector_variables = "all", # the next three␣
 ↪are keyword arguments but are needed.
                              observation_path = ds_obs_dir,
                              observed_variable_name = "Q",
                              measurement_operator = H)
```

### 1.2 Run

[30]:
```
ref_model = ensemble.ensemble_list[0].model
units = {}
var_names = param_names + stor_names + ['Q']
for var in var_names:
    units.update({var: ref_model.bmi.get_var_units(var)})
```

[31]:
```
n_timesteps = int((ref_model.end_time - ref_model.start_time) /  ref_model.
 ↪time_step)
time = []
assimilated_times = []
lst_state_vector = []
lst_Q_prior = []
lst_Q_obs = []
lst_Q = []
for i in tqdm(range(n_timesteps)):
    time.append(pd.Timestamp(ref_model.time_as_datetime.date()))
    lst_Q_prior.append(ensemble.get_value("Q").flatten())
    if i % 3 == 0: # update every 3 steps
        assimilate = True
        assimilated_times.append(pd.Timestamp(ref_model.time_as_datetime.
 ↪date()))
    else:
        assimilate = False
    ensemble.update(assimilate=assimilate)
    lst_state_vector.append(ensemble.get_state_vector())
```

```
    lst_Q.append(ensemble.get_value("Q").flatten())
    lst_Q_obs.append(ensemble.ensemble_method.obs)
ensemble.finalize()# end model - IMPORTANT! when working with dockers
```

```
100%|                    | 1127/1127 [02:02<00:00,  9.17it/s]
```

[32]:
```
Q_m_arr = np.array(lst_Q).T
Q_m_arr_prior = np.array(lst_Q_prior).T
state_vector_arr = np.array(lst_state_vector)
```

### 1.2.1 process the numpy data into easily acessed data types

[33]:
```
save = False
current_time = str(datetime.now())[:-10].replace(":","_")
```

[34]:
```
df_ensemble = pd.DataFrame(data=Q_m_arr[:,:len(time)].
 ↪T,index=time,columns=[f'particle {n}' for n in range(n_particles)])
df_ensemble_prior = pd.DataFrame(data=Q_m_arr_prior[:,:len(time)].
 ↪T,index=time,columns=[f'particle {n}' for n in range(n_particles)])
```

### 1.2.2 process states and parameters into xarrys

[35]:
```
if save:
    df_ensemble.to_feather(output_path /f'df_ensemble_{current_time}.feather')
```

[36]:
```
data_vars = {}
for i, name in enumerate(param_names + stor_names+ ["Q"]):
    storage_terms_i = xr.DataArray(state_vector_arr[:,:,i].T,
                                   name=name,
                                   dims=["EnsembleMember","time"],
                                  coords=[np.arange(n_particles),df_ensemble.
 ↪index],
                                  attrs={"title": f"HBV storage terms data over␣
 ↪time for {n_particles} particles ",
                                         "history": f"Storage term results␣
 ↪from ewatercycle_HBV.model",
                                      "description":"Moddeled values",
                                          "units": "mm"})
    data_vars[name] = storage_terms_i

ds_combined = xr.Dataset(data_vars,
                         attrs={"title": f"HBV storage terms data over time for␣
 ↪{n_particles} particles ",
                                "history": f"Storage term results from␣
 ↪ewatercycle_HBV.model",}
                          )
```

[37]:
```
if save:
    ds_combined.to_netcdf(output_path / f'combined_ds_{current_time}.nc')
```

### 1.3 Plotting

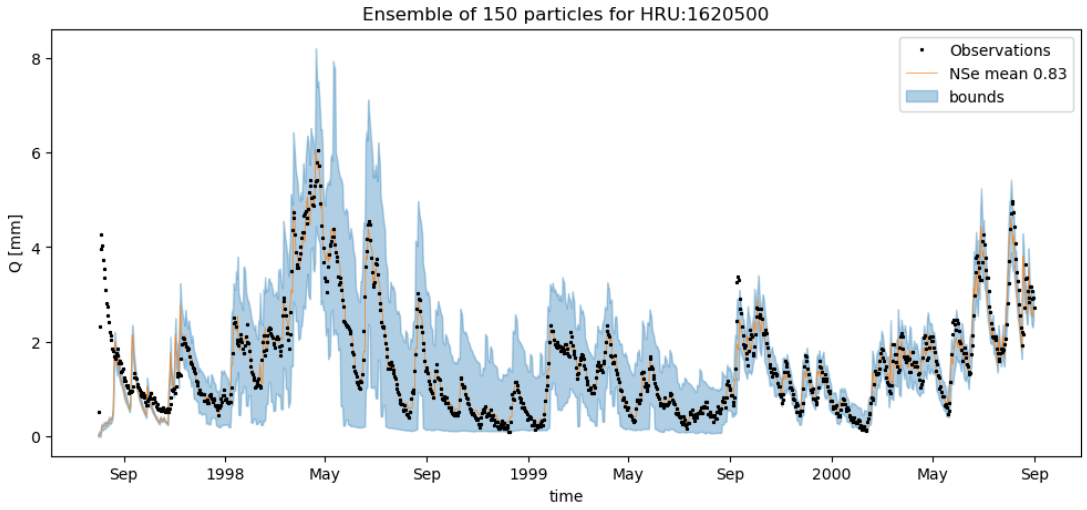
Can calculate the NSE as a reference fit

[38]:
```
def calc_NSE(Qo, Qm):
    QoAv  = np.mean(Qo)
    ErrUp = np.sum((Qo - Qm)**2)
    ErrDo = np.sum((Qo - QoAv)**2)
    return 1 - (ErrUp / ErrDo)
```

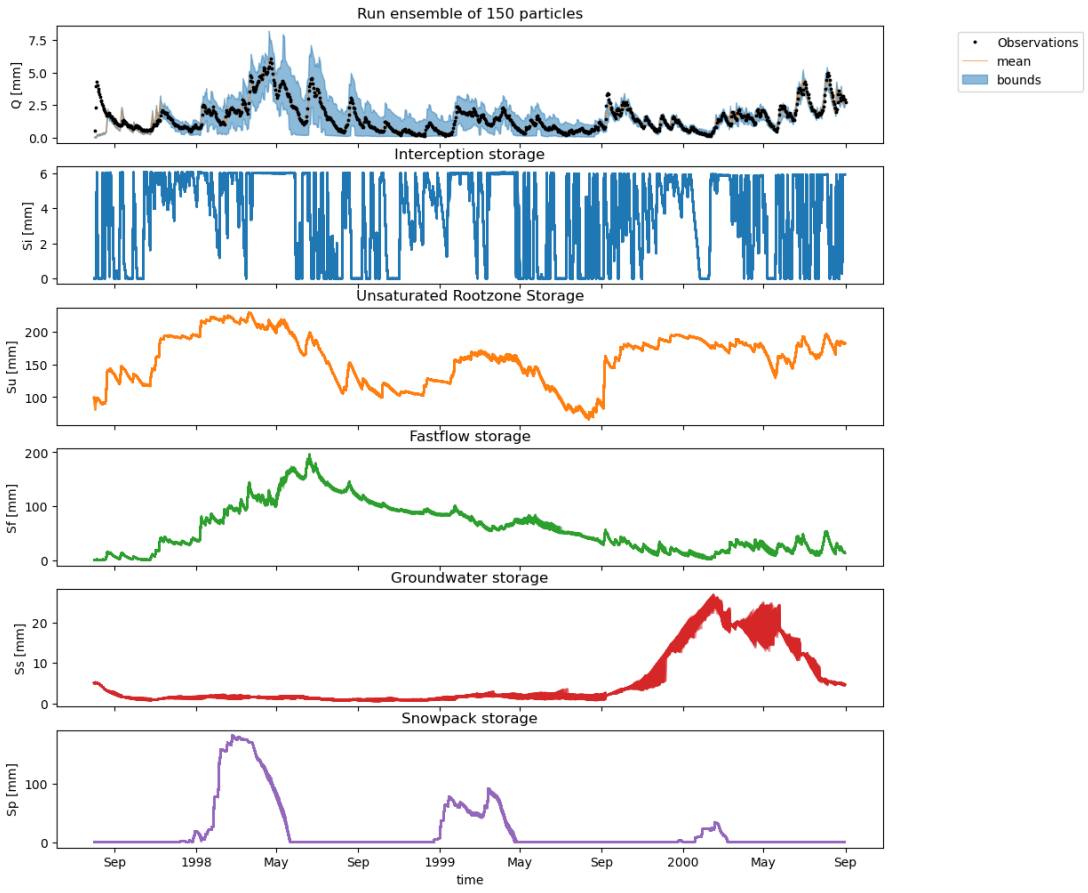
[39]:
```
mean_ensemble = df_ensemble.T.mean()
NSE_mean_ens = calc_NSE(ds_obs['Q'].values,mean_ensemble.loc[time])
```

Result isn't perfect, but this demonstrates the dificulty of working with real data

[40]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
ds_obs['Q'].plot(ax=ax,lw=0,marker="*",ms=2.
 ↪0,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"NSe mean {NSE_mean_ens:.
 ↪2f}",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.35,zorder=-10,label="bounds")
ax.legend()
ax.set_ylabel("Q [mm]")
ax.set_title(f"Ensemble of {n_particles} particles for HRU:{HRU_id}");
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for_{n_particles}_particles_{current_time}.
 ↪png",bbox_inches="tight",dpi=400);
```




[41]:
```
n=6
fig, axs = plt.subplots(n,1,figsize=(12,n*2),sharex=True)

ax = axs[0]
ds_obs['Q'].plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"mean",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.5,zorder=-10,label="bounds")
ax.legend(bbox_to_anchor=(1.25,1))
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
for i, S_name in enumerate(S_names):
    for j in range(n_particles):
        ds_combined[stor_names[i]].isel(EnsembleMember=j).
 ↪plot(ax=axs[i+1],color=f"C{i}",alpha=0.5)
        axs[i+1].set_title(S_name)
        axs[i+1].set_ylabel(f'{stor_names[i]} [{units[stor_names[i]]}]')
[ax.set_xlabel(None) for ax in axs[:-1]]# remove all unncecearry xlabels
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for__{n_particles}_particles_storages_{current_time}.
 ↪png",bbox_inches="tight",dpi=400)
```


Run ensemble of 150 particles
Observations
mean
bounds
Q [mm]
Interception storage
Si [mm]
Unsaturated Rootzone Storage
Su [mm]
Fastflow storage
Sf [mm]
Groundwater storage
Ss [mm]
Snowpack storage
Sp [mm]
Sep
1998
May
Sep
1999
May
Sep
2000
May
Sep
time

[ ]:

### D.1.2. On the fly data assimilation

This is the on the fly approach to run a data assimilation scheme. The model can be run up to a point, after which the modeler can then pass the wanted info, then the data assimilation can run. Notebook on the next page.

# Data assimilation on the Fly - HBV model with Particle filter

June 10, 2024

## 1 'On the fly' Data assimilation on HBV model using particle filtering

[1]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import xarray as xr
from tqdm import tqdm
from rich import print
```

[2]:
```
import ewatercycle
import ewatercycle.forcing
import ewatercycle.models
```

Ensure the HBV model is loaded and the Data Assimilation model is present:

```
pip install ewatercycle-HBV
pip install ewatercycle-DA
```

**set up paths**

[3]:
```
path = Path.cwd()
forcing_path = path / "Forcing"
observations_path = path / "Observations"
figure_path = path / "Figures"
output_path = path / "Output"
for path in [forcing_path, observations_path, figure_path, output_path]:
    path.mkdir(exist_ok=True)
```

**add parameter info**

[4]:
```
## Array of initial storage terms - we keep these constant for now
##              Si,  Su, Sf, Ss
s_0 = np.array([0,  100,  0,  5, 0])
## Array of parameters min/max bounds as a reference
```

```
##                        Imax,  Ce,  Sumax, beta,  Pmax,  T_lag,   Kf,   Ks, FM
p_min_initial= np.array([0,   0.2,  40,    .5,   .001,   1,      .01,  .0001, 6])
p_max_initial = np.array([8,    1,  800,   4,    .3,     10,     .1,   .01, 0.1])
p_names = ["$I_{max}$",  "$C_e$",  "$Su_{max}$", " ",  "$P_{max}$", ␣
 ↪"$T_{lag}$",   "$K_f$",   "$K_s$", "FM"]
S_names = ["Interception storage", "Unsaturated Rootzone Storage", "Fastflow␣
 ↪storage", "Groundwater storage", "Snowpack storage"]
param_names = ["Imax","Ce",  "Sumax", "Beta",  "Pmax",  "Tlag",   "Kf",   "Ks",␣
 ↪"FM"]
stor_names = ["Si", "Su", "Sf", "Ss", "Sp"]
par_0 = (p_min_initial + p_max_initial)/2 # set initial as mean of max,min
```

[5]:
```
experiment_start_date = "1997-08-01T00:00:00Z"
experiment_end_date = "2000-09-01T00:00:00Z"
alpha = 1.26
```

[6]:
```
from ewatercycle.forcing import sources
```

The test forcing shown here is availible from github.com/Daafip/ewatercycle-hbv

[7]:
```
test_forcing = sources.HBVForcing(start_time = experiment_start_date,
                            end_time = experiment_end_date,
                            directory = forcing_path,
                            camels_file = f'test_forcing.txt',
                            test_data_bool = True
                            )
```

**import model**

[8]:
```
from ewatercycle.models import HBV
```

**import DA function:**

[9]:
```
from ewatercycle_DA import DA
```

[10]:
```
n_particles = 50
```

[11]:
```
ensemble = DA.Ensemble(N=n_particles)
ensemble.setup()
```

[12]:
```
array_random_num = np.array([[np.random.random() for i in␣
 ↪range(len(p_max_initial))] for i in range(n_particles)])
p_intial = p_min_initial + array_random_num * (p_max_initial-p_min_initial)
```

[13]:
```
# values wihch you
setup_kwargs_lst = []
for index in range(n_particles):
    setup_kwargs_lst.append({'parameters':','.join([str(p) for p in␣
 ↪p_intial[index]]),
```

```
                                'initial_storage':','.join([str(s) for s in s_0]),
                                })
```

[14]:
```
# this initializes the models for all ensemble members.
ensemble.initialize(model_name=["HBV"]*n_particles,
                    forcing=[test_forcing]*n_particles,
                    setup_kwargs=setup_kwargs_lst)
```

### 1.1 Import observations

[15]:
```
# create a reference model
ref_model = ensemble.ensemble_list[0].model
```

[16]:
```
# testing data
ds_obs_dir = forcing_path / ref_model.forcing.pr
ds_obs = xr.load_dataset(ds_obs_dir)
```

[17]:
```
units = {}
var_names = param_names + stor_names + ['Q']
for var in var_names:
    units.update({var: ref_model.bmi.get_var_units(var)})
```

### 1.2 setup DA

This sets up all the require data assimilation information

[18]:
```
def H(Z):
    if len(Z) == 15:
        return Z[-1]
    else:
        raise SyntaxWarning(f"Length of statevector should be 9 but is␣
 ↪{len(Z)}")
```

### 1.3 Run

#### 1.3.1 First half without assimilating:

[19]:
```
n_timesteps = int((ref_model.end_time - ref_model.start_time) /  ref_model.
 ↪time_step)
half_n_timesteps = int(n_timesteps/2)
lst_Q = []
time = []
for _ in tqdm(range(half_n_timesteps)):
    time.append(pd.Timestamp(ref_model.time_as_datetime.date()))
    ensemble.update(assimilate=False)
    lst_Q.append(ensemble.get_value("Q").flatten())
    # TODO: adjust so that tLag ? currently still often 3
```

```
100%|                    | 563/563 [00:30<00:00, 18.49it/s]
```

### 1.3.2 Then second half with

[20]:
```
ensemble_method_name = "PF"
lst_like_sigma = [0.0075] * 14 + [0]
hyper_parameters = {
                'like_sigma_weights' : 0.005,
                'like_sigma_state_vector' : lst_like_sigma,
                'f_n_particles': 1,
                  }

state_vector_variables = "all"
measurement_operator = H
```

[21]:
```
lst_state_vector = []
for _ in tqdm(range(half_n_timesteps)):
    time.append(pd.Timestamp(ref_model.time_as_datetime.date()))
    ensemble.update(assimilate=False)
    ensemble.assimilate(ensemble_method_name = ensemble_method_name,
                    obs = ds_obs["Q"].sel(time=time[-1]).values,
                    measurement_operator = measurement_operator,
                    hyper_parameters = hyper_parameters,
                    state_vector_variables = state_vector_variables)

    lst_state_vector.append(ensemble.get_state_vector())
    lst_Q.append(ensemble.get_value("Q").flatten())
ensemble.finalize() # end model - IMPORTANT! when working with dockers
```

```
100%|                    | 563/563 [08:36<00:00,  1.09it/s]
```

### 1.3.3 process the numpy data into easily acessed data types

[22]:
```
Q_m_arr = np.array(lst_Q).T
state_vector_arr = np.array(lst_state_vector)
state_vector_arr_copy = state_vector_arr.copy()
```

[23]:
```
df_ensemble = pd.DataFrame(data=Q_m_arr[:,:len(time)].
 ↪T,index=time,columns=[f'particle {n}' for n in range(n_particles)])
```

### 1.3.4 process states and parameters into xarrys

[24]:
```
empty_state_vector = []
for _ in range(half_n_timesteps):
    arr = np.zeros((state_vector_arr.shape[1],state_vector_arr.shape[2]))
    arr[:,:] = np.nan
    empty_state_vector.append(arr)
```

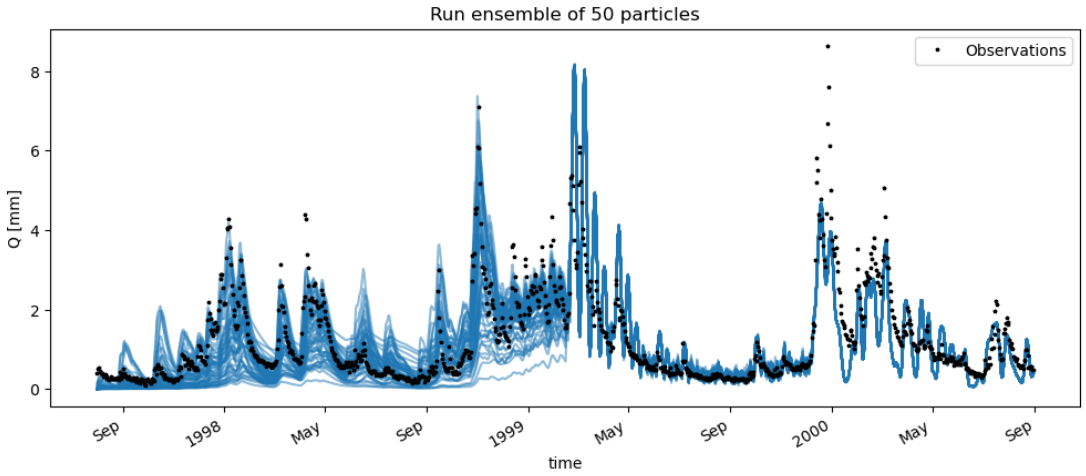
```
empty_state_vector_arr = np.hstack(empty_state_vector).
 ↪reshape(half_n_timesteps,state_vector_arr.shape[1],state_vector_arr.shape[2])
state_vector_arr = np.vstack([empty_state_vector_arr, state_vector_arr_copy])
```

[25]:
```
data_vars = {}
for i, name in enumerate(param_names + stor_names):
    storage_terms_i = xr.DataArray(state_vector_arr[:,:,i].T,
                                   name=name,
                                   dims=["EnsembleMember","time"],
                                  coords=[np.arange(n_particles),df_ensemble.
 ↪index],
                                  attrs={"title": f"HBV storage terms data over␣
 ↪time for {n_particles} particles ",
                                         "history": f"Storage term results␣
 ↪from ewatercycle_HBV.model",
                                      "description":"Moddeled values",
                                         "units": "mm"})
    data_vars[name] = storage_terms_i

ds_combined = xr.Dataset(data_vars,
                         attrs={"title": f"HBV storage terms data over time for␣
 ↪{n_particles} particles ",
                                "history": f"Storage term results from␣
 ↪ewatercycle_HBV.model",}
                          )
```

[26]:
```
save = False
if save:
    ds_combined.to_netcdf(output_path / f'combined_ds_{current_time}.nc')
```

### 1.4 Plotting

[27]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
ds_obs['Q'].sel(time=time).plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')
ax.legend(bbox_to_anchor=(1,1))
df_ensemble.plot(ax=ax,alpha=0.5,zorder=-1,legend=False,color="C0")
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for_{n_particles}_particles_{current_time}.png")
```

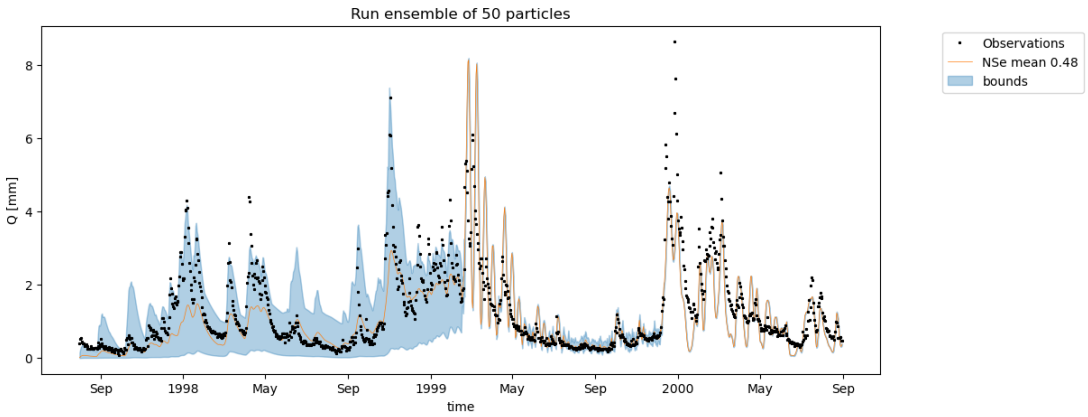



Can calculate the NSE

[28]:
```
def calc_NSE(Qo, Qm):
    QoAv  = np.mean(Qo)
    ErrUp = np.sum((Qm - Qo)**2)
    ErrDo = np.sum((Qo - QoAv)**2)
    return 1 - (ErrUp / ErrDo)
```

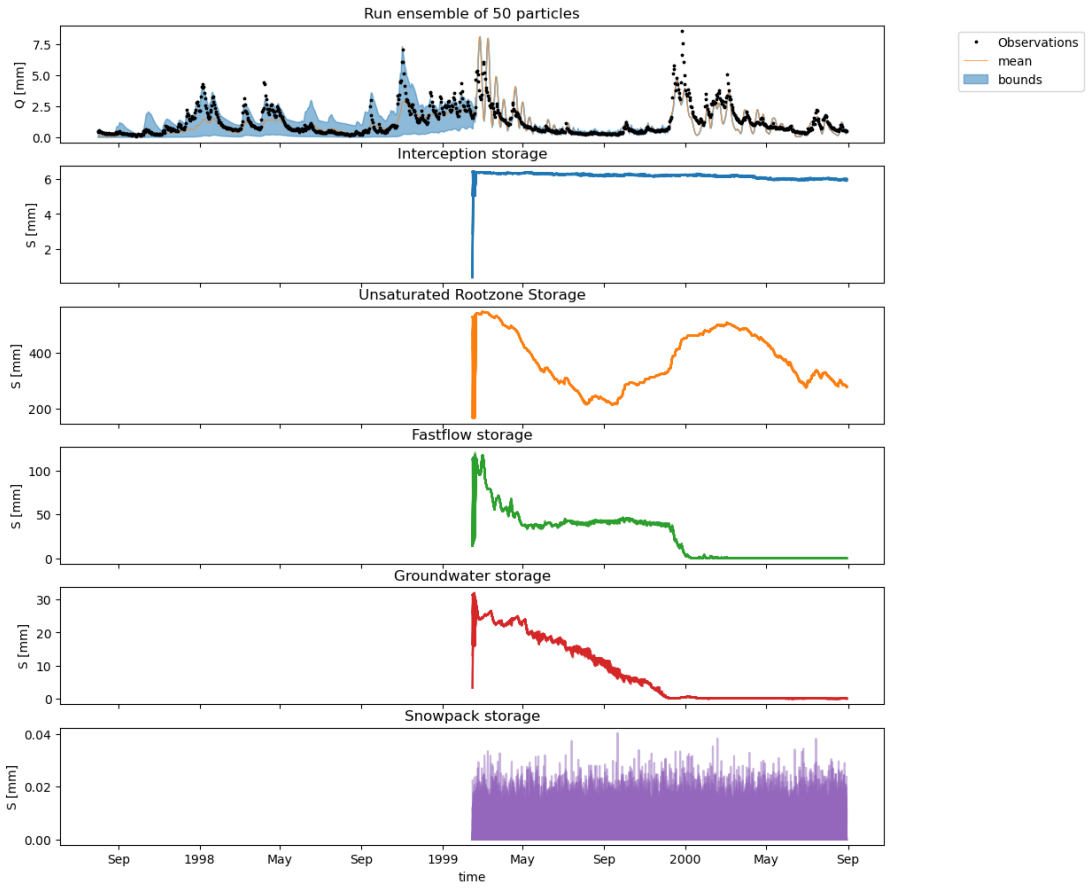
[29]:
```
mean_ensemble = df_ensemble.T.mean()
NSE_mean_ens = calc_NSE(ds_obs['Q'].sel(time=time).values,mean_ensemble.
 ↪loc[time])
```

[30]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
ds_obs['Q'].sel(time=time).plot(ax=ax,lw=0,marker="*",ms=2.
 ↪0,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"NSe mean {NSE_mean_ens:.
 ↪2f}",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.35,zorder=-10,label="bounds")
ax.legend(bbox_to_anchor=(1.25,1))
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for_{n_particles}_particles_{current_time}.
 ↪png",bbox_inches="tight",dpi=400);
```




Here we see the transition half way through from running an ensemble of models to applying data assimilation. In the newest version of the eWaterCyclce-Da is it possible to retrieve the model state before assimilating, this is an older notebook thus doesn't include this aspect.

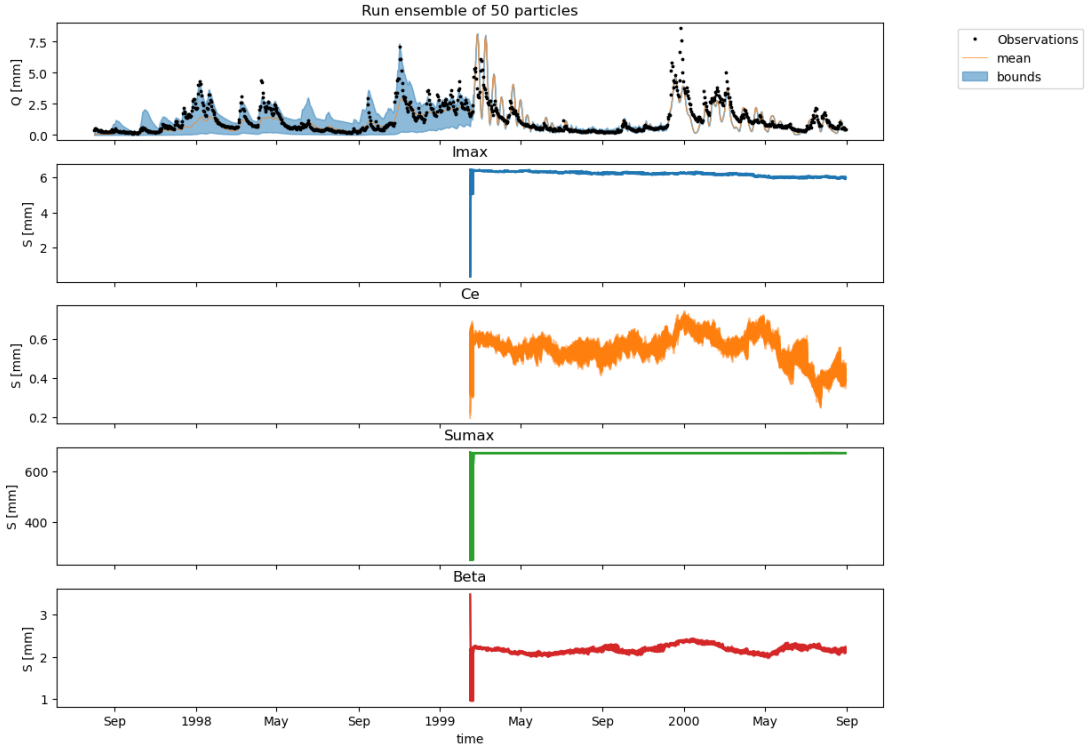
[31]:
```
n=6
fig, axs = plt.subplots(n,1,figsize=(12,n*2),sharex=True)
ax = axs[0]
ds_obs['Q'].sel(time=time).plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"mean",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.5,zorder=-10,label="bounds")
ax.legend(bbox_to_anchor=(1.25,1))
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
for i, S_name in enumerate(S_names):
    for j in range(n_particles):
        ds_combined[stor_names[i]].isel(EnsembleMember=j).
 ↪plot(ax=axs[i+1],color=f"C{i}",alpha=0.5)
        axs[i+1].set_title(S_name)
[ax.set_xlabel(None) for ax in axs[:-1]]# remove all unncecearry xlabels
[ax.set_ylabel("S [mm]") for ax in axs[1:]]
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for__{n_particles}_particles_storages_{current_time}.
 ↪png",bbox_inches="tight",dpi=400)
```




[32]:
```
param_names_0 = param_names[:4]
param_names_1 = param_names[4:]
```

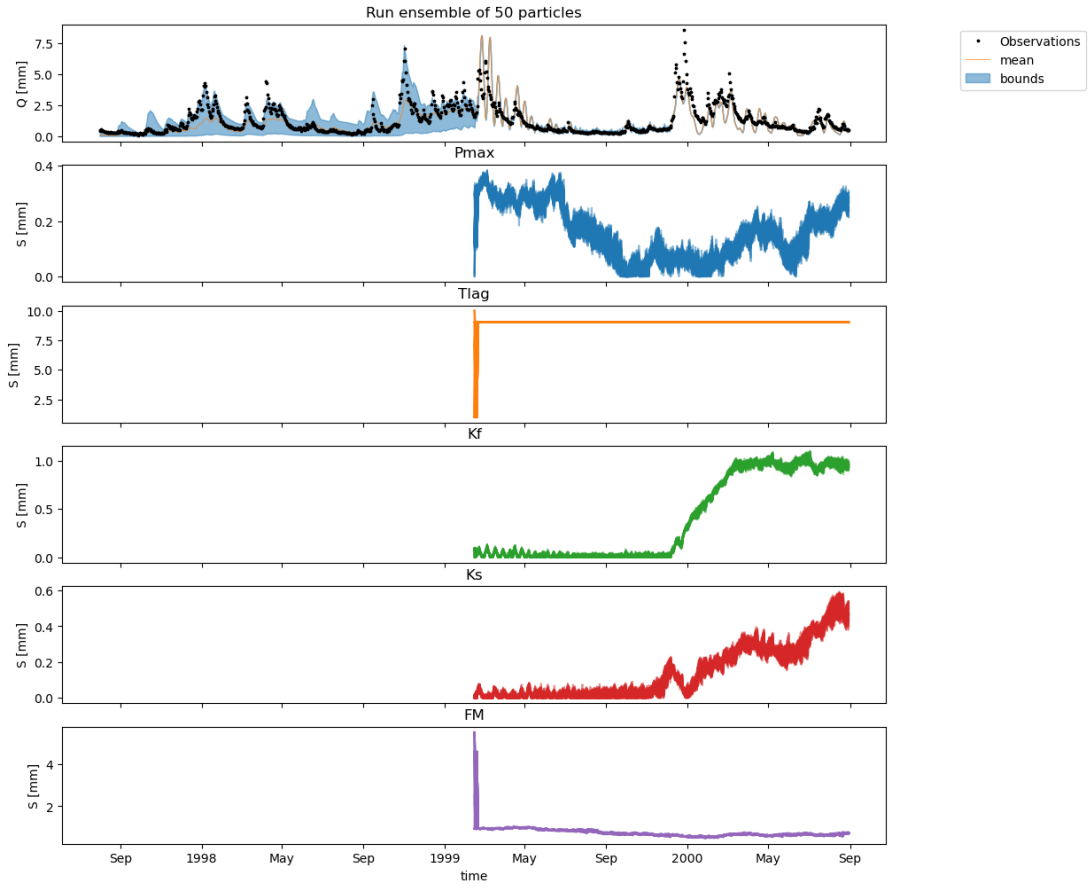
[33]:
```
n=5
fig, axs = plt.subplots(n,1,figsize=(12,n*2),sharex=True)
ax = axs[0]
ds_obs['Q'].sel(time=time).plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"mean",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.5,zorder=-10,label="bounds")
ax.legend(bbox_to_anchor=(1.25,1))
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
for i, parameter in enumerate(param_names_0):
    for j in range(n_particles):
```

```
        ds_combined[parameter].isel(EnsembleMember=j).
 ↪plot(ax=axs[i+1],color=f"C{i}",alpha=0.5)
        axs[i+1].set_title(parameter)

[ax.set_xlabel(None) for ax in axs[:-1]]
[ax.set_ylabel("S [mm]") for ax in axs[1:]]
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for__{n_particles}_particles_storages_{current_time}.
 ↪png",bbox_inches="tight",dpi=400)
```




[34]:
```
n=6
fig, axs = plt.subplots(n,1,figsize=(12,n*2),sharex=True)
ax = axs[0]
ds_obs['Q'].sel(time=time).plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')

ax.plot(mean_ensemble,color="C1",lw=0.5,label=f"mean",zorder=-1)
ax.fill_between(df_ensemble.index,df_ensemble.T.min(),df_ensemble.T.
 ↪max(),color="C0", alpha=0.5,zorder=-10,label="bounds")
ax.legend(bbox_to_anchor=(1.25,1))
ax.set_ylabel("Q [mm]")
```

```
ax.set_title(f"Run ensemble of {n_particles} particles");
for i, parameter in enumerate(param_names_1):
    for j in range(n_particles):
        ds_combined[parameter].isel(EnsembleMember=j).
 ↪plot(ax=axs[i+1],color=f"C{i}",alpha=0.5)
        axs[i+1].set_title(parameter)

[ax.set_xlabel(None) for ax in axs[:-1]]
[ax.set_ylabel("S [mm]") for ax in axs[1:]]
if save:
    fig.savefig(figure_path /␣
 ↪f"ensemble_run_for__{n_particles}_particles_storages_{current_time}.
 ↪png",bbox_inches="tight",dpi=400)
```



### D.1.3. Calibration run

This is the how the data assimilation framwork can be used to calibrate a model. The framework handles parallelisation and getting the states for the user. Note in the new implementation the statevector can be specified earlier allowing for getting and setting at the start too, this is not show. Notebook on the next page.

# Using Data Assimilation framework for a callibration run of HBV

June 10, 2024

## 1 Callibration run

[2]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import xarray as xr
from tqdm import tqdm
from rich import print
```

[3]:
```
import ewatercycle
import ewatercycle.forcing
import ewatercycle.models
```

```
pip install ewatercycle-HBV
pip install ewatercycle-DA
```

[4]:
```
path = Path.cwd()
forcing_path = path / "Forcing"
observations_path = path / "Observations"
figure_path = path / "Figures"
output_path = path / "Output"
for path in [forcing_path, observations_path, figure_path, output_path]:
    path.mkdir(exist_ok=True)
```

Simple example using HBV:

[5]:
```
from ewatercycle.forcing import sources
```

[6]:
```
from ewatercycle_DA import DA
```

[8]:
```
n_particles = 200
```

[10]:
```
experiment_start_date = "1997-08-01T00:00:00Z"
experiment_end_date = "1999-03-01T00:00:00Z"
HRU_id = 1620500
```

```
alpha = 1.26
```

This forcing file is availible at https://github.com/Daafip/Msc-Thesis-Notebooks

[18]:
```
camels_forcing = sources.HBVForcing(start_time = experiment_start_date,
                            end_time = experiment_end_date,
                            directory = forcing_path,
                            camels_file = f'0{HRU_id}_lump_cida_forcing_leap.txt',
                            alpha = alpha,
                            )
```

[19]:
```
## Array of initial storage terms - we keep these constant for now
##              Si,  Su, Sf, Ss
s_0 = np.array([0,  100,  0,  5, 0])
## Array of parameters min/max bounds as a reference
##                       Imax,  Ce,  Sumax, beta,  Pmax,  T_lag,   Kf,   Ks, FM
p_min_initial= np.array([0,   0.2,  40,    .5,   .001,   1,     .01,  .0001, 6])
p_max_initial = np.array([8,    1,  800,   4,    .3,     10,    .1,   .01, 0.1])
p_names = ["$I_{max}$",  "$C_e$",  "$Su_{max}$", " ",  "$P_{max}$", ␣
 ↪"$T_{lag}$",   "$K_f$",   "$K_s$", "FM"]
S_names = ["Interception storage", "Unsaturated Rootzone Storage", "Fastflow␣
 ↪storage", "Groundwater storage", "Snowpack storage"]
param_names = ["Imax","Ce",  "Sumax", "Beta",  "Pmax",  "Tlag",   "Kf",   "Ks",␣
 ↪"FM"]
stor_names = ["Si", "Su", "Sf", "Ss", "Sp"]
```

[20]:
```
ensemble = DA.Ensemble(N=n_particles)
ensemble.setup()
```

[21]:
```
array_random_num = np.array([[np.random.random() for i in␣
 ↪range(len(p_max_initial))] for i in range(n_particles)])
p_intial = p_min_initial + array_random_num * (p_max_initial-p_min_initial)
```

[22]:
```
# values wihch you
setup_kwargs_lst = []
for index in range(n_particles):
    setup_kwargs_lst.append({'parameters':','.join([str(p) for p in␣
 ↪p_intial[index]]),
                             'initial_storage':','.join([str(s) for s in s_0]),
                              })
```

[23]:
```
from ewatercycle_DA.local_models.HBV import HBVLocal
```

[24]:
```
# this initializes the models for all ensemble members.
ensemble.initialize(model_name=["HBVLocal"]*n_particles,
                    forcing=[camels_forcing]*n_particles,
                    setup_kwargs=setup_kwargs_lst)
```

[25]:
```
ensemble.set_state_vector_variables('all')
```

[26]:
```
ref_model = ensemble.ensemble_list[0].model
```

[27]:
```
n_timesteps = int((ref_model.end_time - ref_model.start_time) /  ref_model.
 ↪time_step)

time = []
assimilated_times = []
lst_state_vector = []
lst_Q = []
for i in tqdm(range(n_timesteps)):
    time.append(pd.Timestamp(ref_model.time_as_datetime.date()))
    ensemble.update(assimilate=False)

    lst_state_vector.append(ensemble.get_state_vector())
    lst_Q.append(ensemble.get_value("Q").flatten())

# end model - IMPORTANT! when working with dockers
ensemble.finalize()
```

```
100%|                    | 577/577 [00:57<00:00, 10.11it/s]
```

[28]:
```
Q_m_arr = np.array(lst_Q).T
state_vector_arr = np.array(lst_state_vector)
df_ensemble = pd.DataFrame(data=Q_m_arr[:,:len(time)].
 ↪T,index=time,columns=[f'particle {n}' for n in range(n_particles)])
```

Load obs

[31]:
```
ds = xr.open_dataset(forcing_path / ref_model.forcing.pr)
observations = observations_path / f'0{HRU_id}_streamflow_qc.txt'
cubic_ft_to_cubic_m = 0.0283168466
new_header = ['GAGEID','Year','Month', 'Day', 'Streamflow(cubic feet per␣
 ↪second)','QC_flag']
new_header_dict = dict(list(zip(range(len(new_header)),new_header)))

df_Q = pd.read_fwf(observations,delimiter=' ',encoding='utf-8',header=None)
df_Q = df_Q.rename(columns=new_header_dict)
df_Q['Streamflow(cubic feet per second)'] = df_Q['Streamflow(cubic feet per␣
 ↪second)'].apply(lambda x: np.nan if x==-999.00 else x)
df_Q['Q (m3/s)'] = df_Q['Streamflow(cubic feet per second)'] *␣
 ↪cubic_ft_to_cubic_m
df_Q['Q'] = df_Q['Q (m3/s)'] / ds.attrs['area basin(m^2)'] * 3600 * 24 * 1000 #␣
 ↪m3/s -> m/s ->m/d -> mm/d
df_Q.index = df_Q.apply(lambda x: pd.Timestamp(f'{int(x.Year)}-{int(x.
 ↪Month)}-{int(x.Day)}'),axis=1)
df_Q.index.name = "time"
```

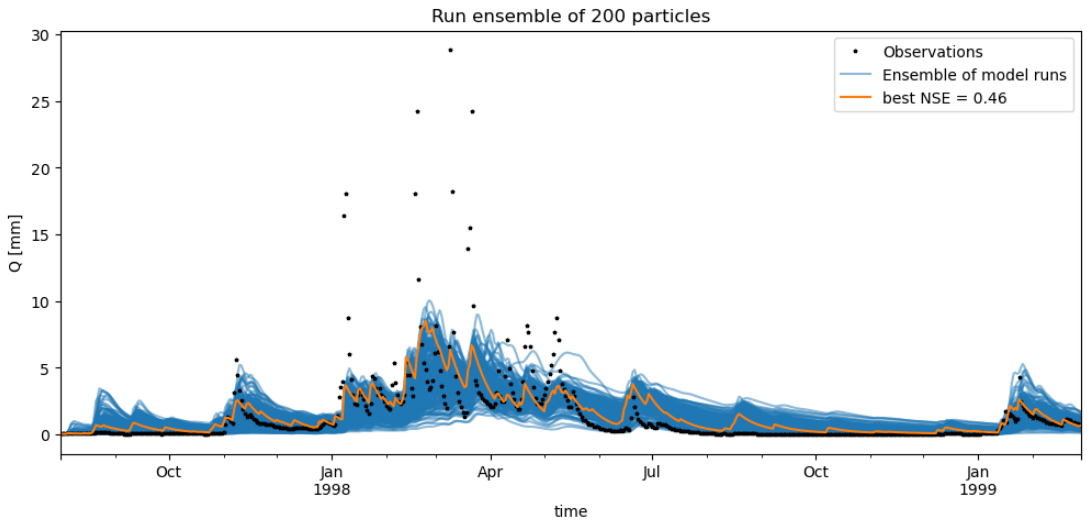
```
df_Q.drop(columns=['Year','Month', 'Day','Streamflow(cubic feet per␣
 ↪second)'],inplace=True)
df_Q = df_Q.dropna(axis=0)
df_Q = df_Q.loc[time]
```

plot

[32]:
```
def calc_NSE(Qo, Qm):
    QoAv  = np.mean(Qo)
    ErrUp = np.sum((Qm - Qo)**2)
    ErrDo = np.sum((Qo - QoAv)**2)
    return 1 - (ErrUp / ErrDo)
```

[33]:
```
lst_nse = []
for i in range(n_particles):
    lst_nse.append(calc_NSE(df_Q['Q'],df_ensemble[f'particle {i}']))
```

[41]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
df_Q['Q'].plot(ax=ax,lw=0,marker="*",ms=2.
 ↪5,zorder=0,label="Observations",color='k')
df_ensemble.plot(ax=ax,alpha=0.5,zorder=-1,legend=False,color="C0")
df_ensemble[f'particle {np.array(lst_nse).argmax()}'].
 ↪plot(ax=ax,color="C1",label=f'best NSE = {np.array(lst_nse).max():.2f}')
handles, labels = ax.get_legend_handles_labels()
ax.legend([handles[0],handles[1],handles[-1]],[labels[0],'Ensemble of model␣
 ↪runs',labels[-1]],bbox_to_anchor=(1,1))
ax.set_ylabel("Q [mm]")
ax.set_title(f"Run ensemble of {n_particles} particles");
```



## D.2. Lorenz example data assimilation

See next page.

# Data assimilation Lorenz model Particle filter

June 10, 2024

## 1 Running the Lorenz model with synthetic observation using Data Assimilation in eWaterCycle

[1]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from datetime import datetime
import xarray as xr
from tqdm import tqdm
from rich import print
```

[2]:
```
import ewatercycle
import ewatercycle.forcing
import ewatercycle.models
```

Ensure ewatercycle-DA is installed

```
pip install ewatercycle-DA
```

**set up paths**

[3]:
```
path = Path.cwd()
forcing_path = path / "Forcing"
observations_path = path / "Observations"
figure_path = path / "Figures"
output_path = path / "Output"
for path in [forcing_path, observations_path, figure_path, output_path]:
    path.mkdir(exist_ok=True)
```

**add parameter info**

[4]:
```
experiment_start_date = "1997-10-01T00:00:00Z"
experiment_end_date = "1997-10-10T00:00:00Z"
```

[5]:
```
from ewatercycle.forcing import sources
```

[6]:
```
lorenz_forcing = sources.LorenzForcing(start_time = experiment_start_date,
                                       end_time = experiment_end_date,
                                       directory = forcing_path,
                                       F = 8,
                                       dt=1e-3)
```

**import model**

[7]:
```
from ewatercycle_DA import DA
from ewatercycle_DA.local_models.lorenz import LorenzLocal
```

The ewatercycle-Da package comes with a local lorenz model, however a docker version is also availible. To use this ensure the package is installed:

`pip install ewatercycle-lorenz`

then replace `LorenzLocal` with `Lorenz`

[8]:
```
ewatercycle.models.sources
```

[8]:
```
ModelSources[
    "HBV",
    "HBVLocal",
    "Lorenz"
]
```

[9]:
```
n_particles = 10
```

[10]:
```
ensemble = DA.Ensemble(N=n_particles)
ensemble.setup()
```

[11]:
```
J = 40
common_state = np.zeros(J)
common_state[19] = 0.01
```

**petrub state every so slightly to start with:**

[12]:
```
rng = np.random.default_rng()
def add_normal_noise(like_sigma) -> float:
    """Normal (zero-mean) noise to be added to a state"""
    return rng.normal(loc=0, scale=like_sigma)
```

[13]:
```
# values which you peturb
setup_kwargs_lst = []
for index in range(n_particles):
    peturbed_state = common_state.copy()
    peturbed_state[5] += add_normal_noise(0.01)
    setup_kwargs_lst.append({'J':J,
                             'start_state':list(peturbed_state),
                             })
```

[14]:
```
ensemble.initialize(model_name=["LorenzLocal"]*n_particles,
                    forcing=[lorenz_forcing]*n_particles,
                    setup_kwargs=setup_kwargs_lst)
```

### 1.1 run truth model as observations

[15]:
```
truth_model = LorenzLocal(forcing=lorenz_forcing)
config, _ = truth_model.setup(J=J,
                        start_state=list(common_state))
truth_model.initialize(config)
```

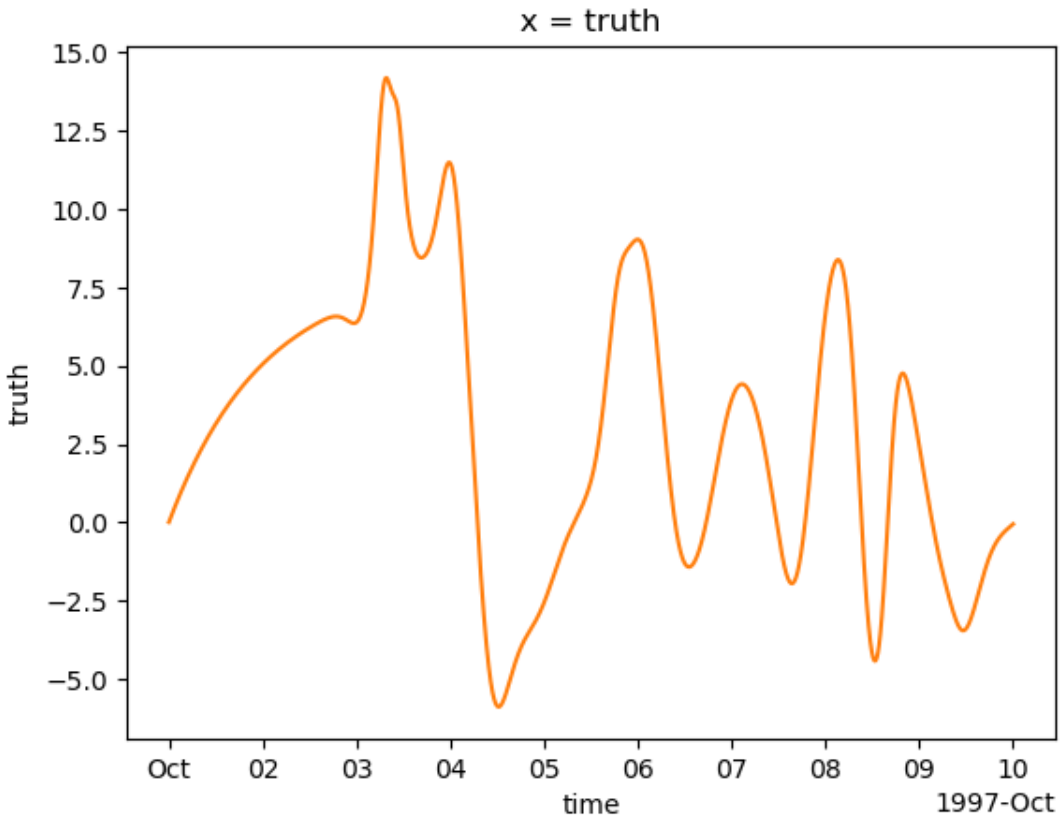
[16]:
```
n_timesteps = int((truth_model.end_time - truth_model.start_time) / ␣
 ↪truth_model.time_step)
output = pd.DataFrame(columns =['truth'])
output_lst = []
for _ in tqdm(range(n_timesteps)):
    truth_model.update()
    output.loc[truth_model.time_as_datetime] = truth_model.get_value("state")[5]
    output_lst.append(truth_model.get_value("state"))
truth_model.finalize()
```

```
100%|                    | 9000/9000 [00:11<00:00, 812.34it/s]
```

[17]:
```
output.index.name = "time"
current_time = str(datetime.now())[:-10].replace(":","_")
ds_obs_dir = observations_path / f'truth_model_lorenz96_{current_time}.nc'
ds_obs = xr.Dataset({'truth':xr.DataArray(data=pd.DataFrame(output,index=output.
 ↪index),dims=["time","x"])})
if not ds_obs_dir.exists():
    ds_obs.to_netcdf(ds_obs_dir)
```

Highly chaotic system

[30]:
```
ds_obs['truth'].plot(color="C1");
```




### 1.2 setup DA

This sets up all the require data assimilation information

[19]:
```
def H(Z):
    """Function which takes the state vector Z and returns part of that state.
    Returned should be same shape as data provided"""
    if len(Z) == J:
        return Z[5]
    else:
        raise SyntaxWarning(f"Length of statevector should be {J} but is␣
↪{len(Z)}")
```

[20]:
```
ensemble.initialize_da_method(ensemble_method_name = "PF",
                              hyper_parameters = {
                                                  'like_sigma_weights' : 0.05,
                                                  'like_sigma_state_vector' : 0.8,
                                                  'f_n_particles':1,
                                                  },
                              state_vector_variables = ["state"],
```

```
                                observation_path = ds_obs_dir,
                                observed_variable_name = "truth",
                                measurement_operator= H
                                 )
```

## 2 Run

[21]:
```
ref_model = ensemble.ensemble_list[0].model
```

[22]:
```
assimilation_window = 10
```

[23]:
```
n_timesteps = int((ref_model.end_time - ref_model.start_time) /  ref_model.
 ↪time_step)
time = []
lst_state_vector = []
lst_Q = []
for i in tqdm(range(n_timesteps)):
    time.append(pd.Timestamp(ref_model.time_as_datetime.date()))
    if i % assimilation_window == 0:  assimilate = True
    else:                             assimilate = False
    ensemble.update(assimilate)
    lst_state_vector.append(ensemble.get_state_vector())
ensemble.finalize()
```

```
100%|                    | 9000/9000 [00:58<00:00, 153.40it/s]
```

[24]:
```
Q_m_arr = np.array(lst_Q).T
state_vector_arr = np.array(lst_state_vector)
```

### 2.0.1 process the numpy data into easily acessed data types

[25]:
```
save = False
current_time = str(datetime.now())[:-10].replace(":","_")
```

### 2.0.2 process states and parameters into xarrys

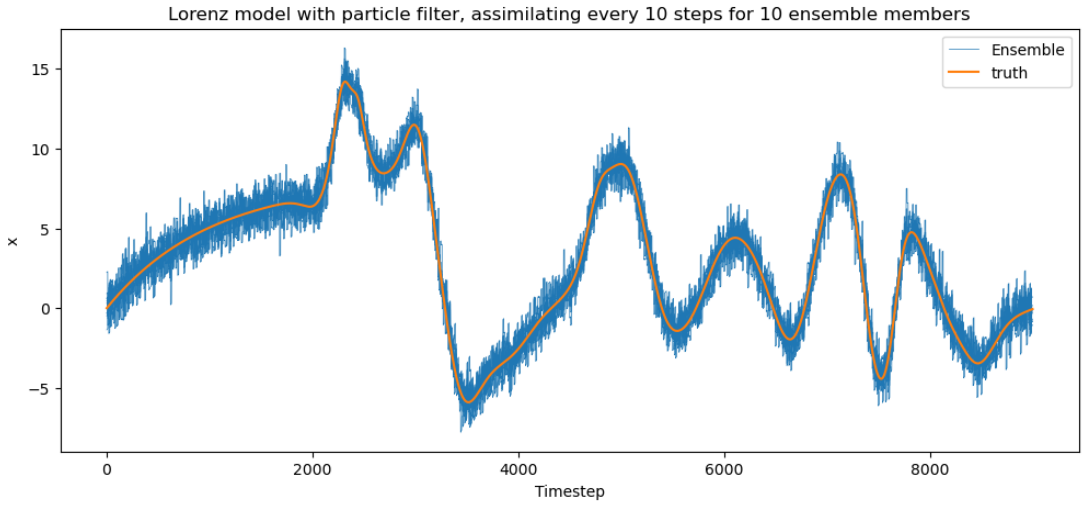
[26]:
```
data_vars = {}
for i, name in enumerate(range(J)):
    storage_terms_i = xr.DataArray(state_vector_arr[:,:,i].T,
                                   name=name,
                                   dims=["EnsembleMember","time"],
                                  coords=[np.arange(n_particles),time[:
 ↪len(state_vector_arr)]],
                                  attrs={"title": f"Lorenz states over time for␣
 ↪{n_particles} particles ",
                                         "history": f"states of lorenz model:␣
 ↪ewatercycle DA",
```

```
                                          "description":"Moddeled values",
                                               "units": "mm"})
    data_vars[name] = storage_terms_i

ds_combined = xr.Dataset(data_vars,
                         attrs={"title": f"Lorenz model over time for␣
 ↪{n_particles} particles ",
                                "history": f"states of lorenz model:␣
 ↪ewatercycle DA",}
                          )
```

[27]:
```
if save:
    ds_combined.to_netcdf(output_path / f'combined_ds_{current_time}.nc')
```

### 2.1 Plotting

[29]:
```
fig, ax = plt.subplots(1,1,figsize=(12,5))
df_truth = ds_obs['truth'].to_pandas()
df_truth.index = range(len(ds_obs.time))
x_values = df_truth.index
x = 1
for i in range(n_particles):
    ax.plot(x_values, ds_combined[x].isel(EnsembleMember=i).values,lw=0.
 ↪5,color='C0',zorder=-1,label="Ensemble")
df_truth.plot(ax=ax,zorder=1, color="C1")
handels, labels =  ax.get_legend_handles_labels()
ax.legend(handels[-2:],labels[-2:],title=None)
ax.set_xlabel('Timestep')
ax.set_ylabel('x')
ax.set_title(f"Lorenz model with particle filter, assimilating every␣
 ↪{assimilation_window} steps for {n_particles} ensemble members")
if save:
    fig.savefig(figure_path / "lorenz_model.png",bbox_inches="tight",dpi=400)
```


Lorenz model with particle filter, assimilating every 10 steps for 10 ensemble members
Ensemble
truth
x
15
10
5
0
−5
0
2000
4000
6000
8000
Timestep

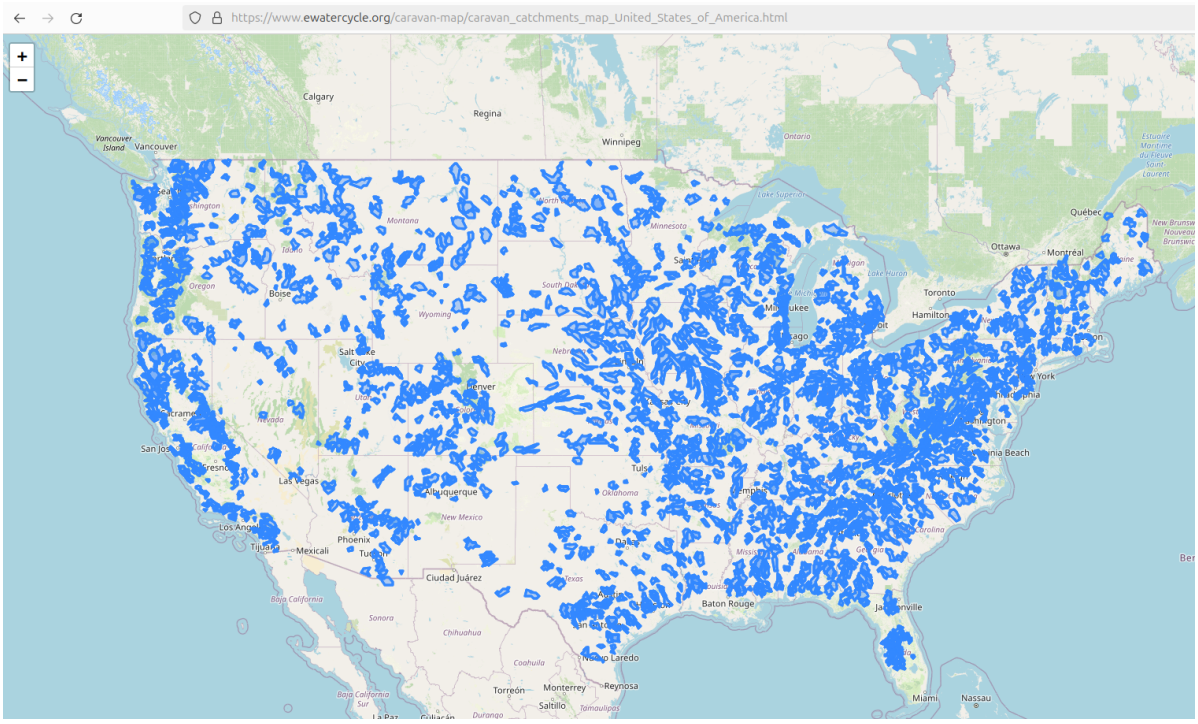



**Figure D.1:** Interactive map

## D.3. Loading caravan data in eWatercycle with OpenDAP

The Carvan dataset is a collection of CAMELS datasets. Specifically it used the USGS data which the CAMELS-USA data set provides and generated matching forcing using Era5-Land. The downside is the original data set can only be accessed currently from zenodo, which results in a large file which then has to be split etc. Using the Directory Access Protocol (DAP) standard which 4TU federation used to host open data files, the user can access the part of the dataset in which they want. In eWaterCycle an code was added whiche which handles this data accessing and retrieval for the user. As part of making the dataset more accessible an interactive map was also made. This allows users to find a catchment on the map which they want to model, find the corresponding basin identifier number and pass this to the generate function in order to retrieve the forcing.

# Caravan from OpenDAP HBV

June 3, 2024

## 1 Retrieve forcing for any CAMELS catchment

In this notebook you will retrieve forcing from the Cravan dataset.The caravan dataset is a collection of streamflow and forcing data. Caravan was prepared by Frederik Kratzert, the forcing is based on the ERA5-Land model. The streamflow is from the USGS. To access it easily, it was stored here on the OPenDAP server from data.4TU.nl . This saves you from downloading and reading the whole dataset hosted on zenodo, instead only the necesarry data is downloaded.

This notebook will show case how to run this for the HBV model.

You have to supply the wanted basin_id. The shapefile will be downloaded automatically. Running the default will download the combined shapefile of all the catchments. Loading this combined shapefile and exploring it will show all availible catchments.

[1]:
```
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
import numpy as np
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt
from rich import print
```

[2]:
```
import ewatercycle
import ewatercycle.forcing
import ewatercycle.models
import ewatercycle.analysis
```

**set up paths**

[3]:
```
path = Path.cwd()
forcing_path = path / "Forcing"
camels_path = forcing_path / "Camels"
forcing_path.mkdir(exist_ok=True)
camels_path.mkdir(exist_ok=True)
```

[4]:
```
experiment_start_date = "1997-08-01T00:00:00Z"
experiment_end_date = "2005-09-01T00:00:00Z"
HRU_id = 3439000
```

### 1.0.1 retrieve forcing

[5]:
```
import ewatercycle.forcing
```

[6]:
```
print(ewatercycle.forcing.sources)
```

```
ForcingSources[
    "CaravanForcing",
    "DistributedMakkinkForcing",
    "DistributedUserForcing",
    "GenericDistributedForcing",
    "GenericLumpedForcing",
    "HBVForcing",
    "LorenzForcing",
    "LumpedMakkinkForcing",
    "LumpedUserForcing",
]
```

[7]:
```
caravan_forcing = ewatercycle.forcing.sources['CaravanForcing'].
 ↪generate(start_time = experiment_start_date,
                                                                  ␣
 ↪end_time = experiment_end_date,
                                                                  ␣
 ↪directory = camels_path,
                                                                  ␣
 ↪basin_id = f"camels_0{HRU_id}",
                                                                    )
```

[8]:
```
caravan_forcing.save()
```

[8]:
```
PosixPath('/media/davidhaasnoot/files/work/Studie TUD/Msc/Year 6/Q3
Thesis/Code/eWaterCycle-Msc-Thesis-
Notebooks/Forcing/Camels/ewatercycle_forcing.yaml')
```

[9]:
```
print(caravan_forcing)
```

```
CaravanForcing(
    start_time='1997-08-01T00:00:00Z',
    end_time='2005-09-01T00:00:00Z',
    directory=PosixPath('/media/davidhaasnoot/files/work/Studie TUD/Msc/Year 6/
 ↪Q3
Thesis/Code/eWaterCycle-Msc-Thesis-Notebooks/Forcing/Camels'),
    shape=PosixPath('/media/davidhaasnoot/files/work/Studie TUD/Msc/Year 6/Q3
Thesis/Code/eWaterCycle-Msc-Thesis-Notebooks/Forcing/Camels/camels_03439000.
 ↪shp'),
    filenames={
        'pr': 'camels_03439000_1997-08-01_2005-09-01_pr.nc',
```

```
        'evspsblpot': 'camels_03439000_1997-08-01_2005-09-01_evspsblpot.nc',
        'tas': 'camels_03439000_1997-08-01_2005-09-01_tas.nc',
        'tasmax': 'camels_03439000_1997-08-01_2005-09-01_tasmax.nc',
        'tasmin': 'camels_03439000_1997-08-01_2005-09-01_tasmin.nc',
        'Q': 'camels_03439000_1997-08-01_2005-09-01_Q.nc'
    }
)
```

[10]:
```
ds_caravan = caravan_forcing.to_xarray()
```

[36]:
```
print(list(ds_caravan.data_vars.keys()))
```

```
['Q', 'evspsblpot', 'pr', 'tas', 'tasmax', 'tasmin']
```

[12]:
```
RENAME_ERA5 = {
    "total_precipitation_sum": "pr",
    "potential_evaporation_sum": "evspsblpot",
    "temperature_2m_mean": "tas",
    "temperature_2m_min": "tasmin",
    "temperature_2m_max": "tasmax",
    "streamflow": "Q",
}
```

[13]:
```
RENAME_ERA5.keys()
```

[13]:
```
dict_keys(['total_precipitation_sum', 'potential_evaporation_sum',
'temperature_2m_mean', 'temperature_2m_min', 'temperature_2m_max',
'streamflow'])
```

[14]:
```
ds_caravan.time
```

[14]:
```
<xarray.DataArray 'time' (time: 2954)> Size: 24kB
array(['1997-08-01T00:00:00.000000000', '1997-08-02T00:00:00.000000000',
       '1997-08-03T00:00:00.000000000', …, '2005-08-30T00:00:00.000000000',
       '2005-08-31T00:00:00.000000000', '2005-09-01T00:00:00.000000000'],
      dtype='datetime64[ns]')
Coordinates: (12/18)
  * time            (time) datetime64[ns] 24kB 1997-08-01 … 2005-09-01
    basin_id        |S64 64B b'camels_03439000'
    timezone        |S64 64B b'America/New_York'
    name            |S64 64B b'FRENCH BROAD RIVER AT ROSMAN, NC'
    country         |S64 64B b'United States of America'
    lat             float64 8B 35.14
    …               …
    moisture_index  float64 8B -0.6997
    seasonality     float64 8B 0.2885
```

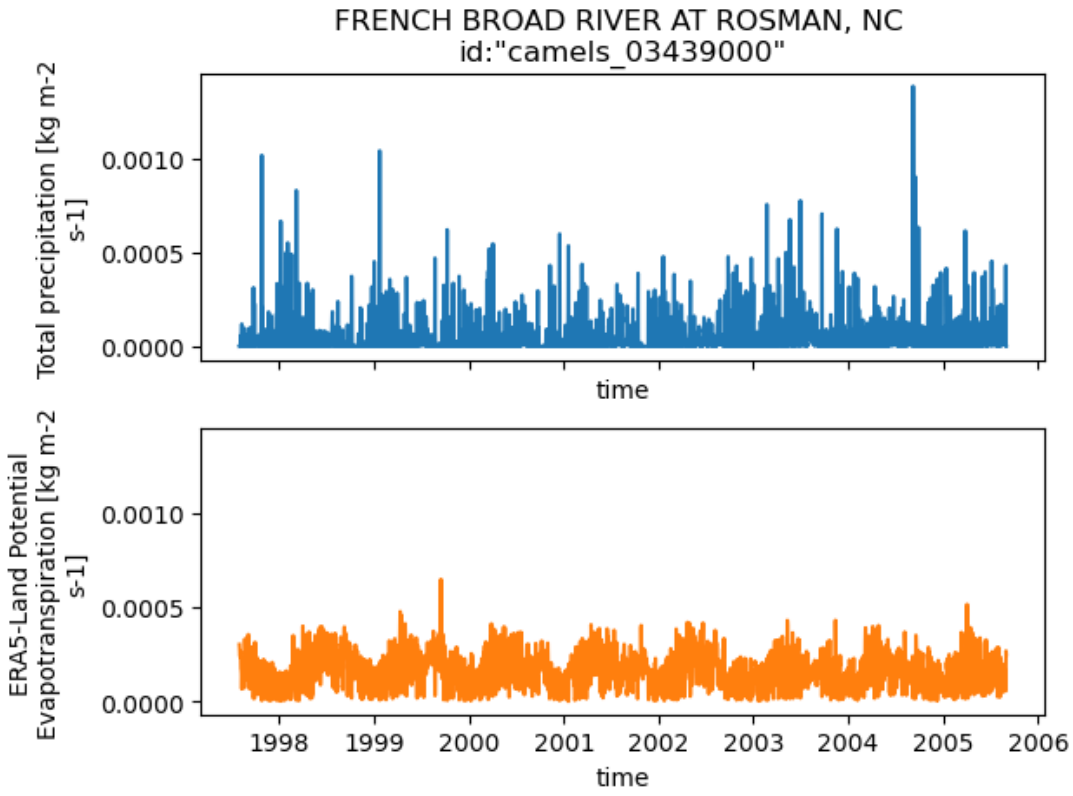
```
    high_prec_freq  float64 8B 0.04833
    high_prec_dur   float64 8B 1.155
    low_prec_freq   float64 8B 0.5741
    low_prec_dur    float64 8B 3.493
```

As you can see it will only download the required data variables needed for modeling

[15]:
```
ds_caravan.data_vars
```

[15]:
```
Data variables:
    Q           (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
    evspsblpot  (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
    pr          (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
    tas         (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
    tasmax      (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
    tasmin      (time) float32 12kB dask.array<chunksize=(2954,),
meta=np.ndarray>
```

[16]:
```
fig, ax = plt.subplots(2,1,sharex=True, sharey=True)
ds_caravan['pr'].plot(ax=ax[0],label="P")
ds_caravan['evspsblpot'].plot(ax=ax[1],label="E",color="C1")
ax[0].set_title(f'{ds_caravan.name.values.astype(str)} \nid:"{ds_caravan.
 ↪basin_id.values.astype(str)}"');
ax[1].set_title(None)
fig.tight_layout()
```




**setup model** I use the ewatercycle-HBV model:

```
pip install ewatercycle-HBV
```

[17]:
```
from ewatercycle.models import HBV
```

[18]:
```
model = HBV(forcing=caravan_forcing)
```

**pass parameters**

[19]:
```
s_0 = np.array([0,  100,  0,  5, 0])
par_0 = np.array([2,    0.8,  460,   1.5,    1.0,     4,    .4,   .04, 3])
```

[20]:
```
config_file, _ = model.setup(parameters=','.join([str(p) for p in par_0]),
                             initial_storage=','.join([str(s) for s in s_0]),
                             )
```

[21]:
```
model.initialize(config_file)
```

**Run the model**

[22]:
```
Q_m = []
time = []
```

```
while model.time < model.end_time:
    model.update()
    Q_m.append(model.get_value("Q"))
    time.append(model.time_as_datetime.date())
```

[23]:
```
hydro_data = pd.DataFrame(data=Q_m,index=time,columns=["model output"])
hydro_data['discharge observations'] = ds_caravan['Q'].to_pandas()
```

[24]:
```
hydro_data
```

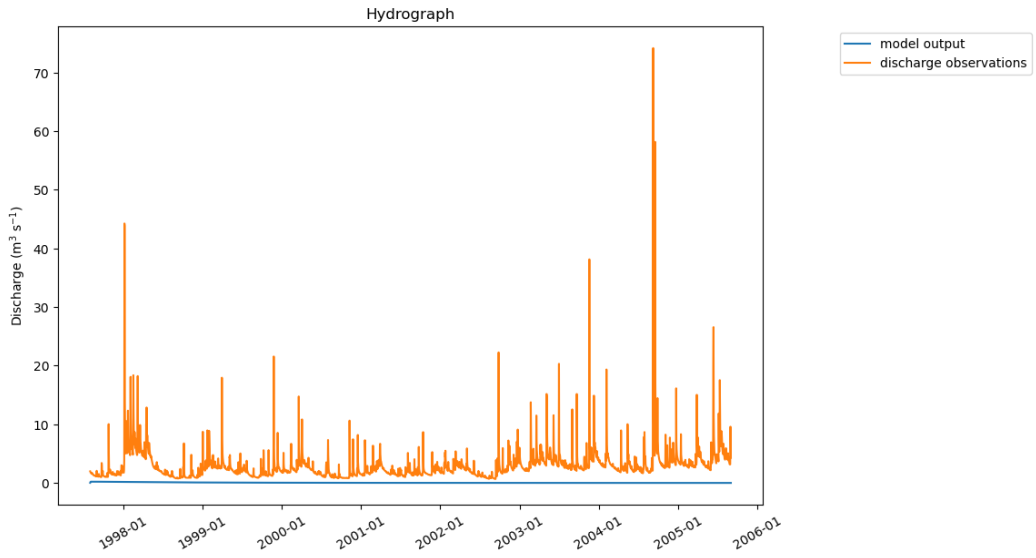
[24]:

| | model output | discharge observations |
|---|---|---|
| 1997-08-02 | 0.026087 | 1.98 |
| 1997-08-03 | 0.104389 | 1.88 |
| 1997-08-04 | 0.182810 | 1.80 |
| 1997-08-05 | 0.209165 | 1.73 |
| 1997-08-06 | 0.209447 | 1.68 |
| … | … | … |
| 2005-08-28 | 0.000376 | 3.15 |
| 2005-08-29 | 0.000375 | 3.17 |
| 2005-08-30 | 0.000374 | 9.59 |
| 2005-08-31 | 0.000374 | 6.15 |
| 2005-09-01 | 0.000373 | 4.30 |

[2953 rows x 2 columns]

Not a great result model wise but it runs and makes some sense!

[25]:
```
ewatercycle.analysis.hydrograph(hydro_data, reference='discharge observations');
```




| | nse | kge_2009 | sa | me |
|---|---|---|---|---|
| model output | -0.92 | -0.72 | 1.18 | -2.93 |

[26]:
```
model.finalize()
```

### 1.0.2 Basins

Using the shapefile we can look at the location of the basin on a map

[27]:
```
import cartopy.crs as ccrs
import cartopy.feature as cfeature
```

I use geopandas:

`pip install geopandas`

as using shapely/cartopy is a hastle

[28]:
```
import geopandas as gpd
```

Alternatively you can also use the html maps hosted here: https://www.ewatercycle.org/caravan-map/ These allow for a interactive exeperience:

[29]:
```
from IPython.display import IFrame
IFrame(src="https://www.ewatercycle.org/caravan-map/
 ↪caravan_catchments_map_United_States_of_America.html",width=600, height=600)
```

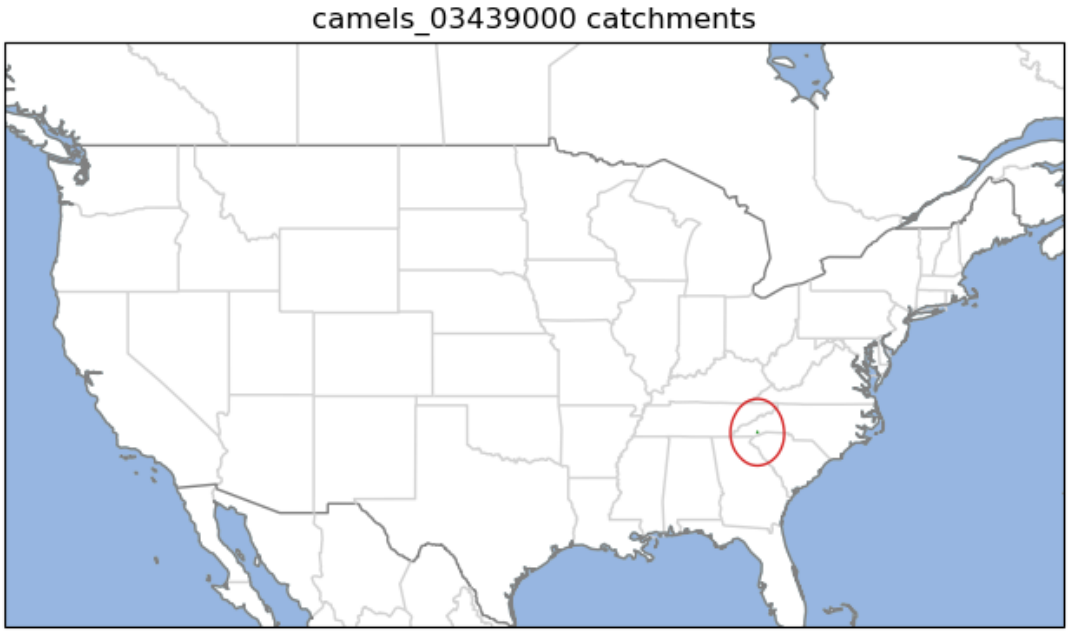
[29]:
```
<IPython.lib.display.IFrame at 0x7d299a9a2e90>
```

[30]:
```
gdf_basin = gpd.read_file(caravan_forcing.shape)
gdf_basin_buffer = gdf_basin.buffer(1.5)
```

[31]:
```
gdf_basin
```

[31]:
```
          gauge_id                                           geometry
0  camels_03439000  POLYGON ((-82.88278 35.30930, -82.87511 35.301…
```

[32]:
```
# fig, ax = plt.subplots()
ax = plt.axes(projection=ccrs.PlateCarree())
country_borders = cfeature.NaturalEarthFeature(
    category='cultural',
    name='admin_0_boundary_lines_land',
    scale='10m',
    facecolor='none')
ax.add_feature(country_borders, edgecolor='gray')

state_borders = cfeature.NaturalEarthFeature(
    category='cultural',
    name='admin_1_states_provinces_lines',
    scale='10m',
    facecolor='none')
ax.add_feature(state_borders, edgecolor='lightgray')
ax.add_feature(cfeature.COASTLINE, edgecolor='gray')
ax.add_feature(cfeature.OCEAN, edgecolor='lightblue')


gdf_basin.plot(ax=ax,facecolor="green")
gdf_basin_buffer.plot(ax=ax,facecolor="None", edgecolor="C3",zorder=10)

ax.set_title(f"{gdf_basin.loc[0,'gauge_id']} catchments")
ax.set_extent([-127.275,  -64.853, 25.864, 50.101], crs=ccrs.Geodetic())

plt.tight_layout()
```




**and with the combined shapefile we see the all the basins**

[33]:
```
gdf_all = gpd.read_file(caravan_forcing.directory / 'shapefiles' / 'combined.
↪shp')
```

[34]:
```
ax = plt.axes(projection=ccrs.PlateCarree())
ax.add_feature(cfeature.COASTLINE, edgecolor='gray')
gdf_all.plot(ax=ax,zorder=1,color="C0")
plt.tight_layout()
```

[35]:
```
gdf_all
```

[35]:
```
             gauge_id                                           geometry
0     camels_01022500  POLYGON ((-67.97836 44.61310, -67.98141 44.614…
1     camels_01031500  MULTIPOLYGON (((-69.31629 45.15325, -69.32144 …
2     camels_01047000  POLYGON ((-70.10847 45.21669, -70.10609 45.213…
3     camels_01052500  POLYGON ((-71.10862 45.12730, -71.10402 45.125…
4     camels_01054200  POLYGON ((-70.97999 44.39574, -70.97657 44.393…
…                 …                                                …
6825      lamah_47840  POLYGON ((9.55616 48.16253, 9.56571 48.15944, …
6826      lamah_76175  POLYGON ((9.88750 48.38333, 9.88529 48.38277, …
6827      lamah_76176  POLYGON ((10.38109 48.87520, 10.37946 48.87519…
6828      lamah_76184  POLYGON ((9.75509 48.31759, 9.75364 48.31563, …
6829      lamah_76276  POLYGON ((10.23328 48.68325, 10.23114 48.67988…

[6830 rows x 2 columns]
```