# cuspy: User and Developer Manual

*Release 1.0.2*

J. Prats-Rodríguez P.-A. Danis

Jan 26, 2023

# CONTENTS:

CHAPTER

ONE

# PRESENTATION

## 1.1 What is the cuspy package?

The package `cuspy` (Calibration, Uncertainty and Sensitivity analysis in PYthon) is a package in Python 3 that provides tools for parameter estimation, and sensitivity and uncertainty analyses of models.

The package is based on the use of the Python package `pyemu` (White et al. 2016) and the PEST++ suite of software tools (White et al. 2019). PEST and PEST++ (Doherty et al. 2018) are software suites designed for and used mainly by the hydrological community. They implement several linear and nonlinear uncertainty analysis, calibration methods and global sensitivity analyses. The package `pyemu` offers a Python interface to use and configure PEST++ tools and to access and process their results.

However, the package `cuspy` does not offer all the functionalities of PEST++ and `pyemu` (for example, tied parameters are not supported). In fact, the package `cuspy` has been designed with the objective of facilitating the application of `pyemu` and PEST++ functionalities to the water temperature models and hydrological models implemented with the packages `okplm`, `glmtools` and `tributary`. Still, the design of the package is sufficiently general to be quite model-independent. The main requirements are that the model can be run through the command line and that it communicates with the user through text files.

## 1.2 Authors

- Jordi Prats-Rodríguez (jprats@segula.es)
- Pierre-Alain Danis (pierre-alain.danis@ofb.gouv.fr)

## 1.3 References

- Doherty, J.; White, J.; Welter, D. (2018) *PEST & PEST++. An Overview.* Watermark Numerical Computing. 48 p.
- White, J.T.; Fienen, M.N.; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling & Software*, 85, 217-228, doi: 10.1016/j.envsoft.2016.08.017.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16.* PEST++ Development Team.

CHAPTER

# TWO

# A BIT OF THEORY

In order to help the user understand better the utilisation and functionalities of this package, we make a short introduction to calibration and uncertainty analysis.

## 2.1 Models and simulations

### 2.1.1 Definitions

Models can be considered as approximations of reality based on mathematical relations and inferences based on data. A model receives forcing data (input data) and transforms it into output data (simulation) by applying a series of mathematical procedures. The output of a model can be of several types, defined in the following table:

Table 2.1: Definitions of types of model output[1]

| Term | Definition |
|---|---|
| Simulation | Quantitative reproduction of the behaviour of a system, given some defined inputs but without reference to any observed outputs. |
| Forecast | Quantitative reproduction of the behaviour of a system ahead of time, but given observations of the inputs, state variables, and outputs up to the forecast origin. |
| Hindcast | Application of forecasting to past data as if the inputs and outputs were only available up to a certain point in time. |
| Projection | Simulation of the future behaviour of a system given prior assumptions about future input data. |

As approximations of reality, models are often used to inform management decisions and to understand better the behaviour of the systems they try to reproduce. By comparing the simulation results to observations it is possible to detect circumstances when the model does not perform well. By analysing these cases, and collecting complementary data, the models may also be used to improve our knowledge of the modelled system.

**Note:** Models are useful tools to analyse the behaviour of a system and to increase our understanding of it.

[1] Based on Beven & Young (2013).

### 2.1.2 The modelling process

The modelling process can be divided in 10 stages (Jakeman et al., 2006):

1. Definition of the model purpose
2. Specification of the modelling context: scope and resources
3. Conceptualisation of the system, specification of data and prior knowledge
4. Selection of model features
5. Determination of the method to identify model structure and parameter values
6. Selection of the performance criteria and algorithm
7. Identification of the model structure and parameter values
8. Conditional verification, including diagnostic testing
9. Quantification of uncertainty
10. Model evaluation

Of these stages, we are interested here in the last ones. However, some of the techniques presented in this document may inform other stages of the process too. Also, the calibrated model may be useful to inform initial stages of the modelling process (e.g., which additional measurements to take) in successive iterations.

At the end of the process, we hope to obtain a model that has the following characteristics (Beck 1987):

- The parameters of the model should not vary with time. Parameter values that vary with time may be indicative of an inadequate model structure or of processes not taken into account.
- The variance of residuals and the covariance of parameters are low, indicating that the model parameters have been well identified.
- The residuals are small and are not attributable to any nonrandom causal mechanism.

### 2.1.3 Evaluation of model performance

If models are to be used as management tools, to provide evidence on which to base decisions, it is only natural to assess their ability to simulate the behaviour of the system. The comparison of simulations to observations or expectations is known as model evaluation and can take several forms.

Table 2.2: Definitions on model evaluation[2]

| Term | Definition |
|---|---|
| Verification | Only for checks that a model computer code properly reproduces the equations it purports to solve. Will generally be only conditional verification dependent on the range of tests carried out. |
| Validation | Conditional evaluation that a model computer code reproduces some observed behaviour of the system being studied to an acceptable degree. |
| Falsification | An assessment that a model computer code does not reproduce some observed behaviour of the system to an acceptable degree. |
| Fit-for-purpose | Conditional evaluation that a computer model code is suitable for use in a particular type of application. |

Bennett et al. (2013) propose a five-step procedure to assess the performance of environmental models:

[2] Modified from Beven & Young (2013, Table 3).

1. Reassessment of the model's aim, scale and scope. Make sure you have a clear idea of the model's objective: which are the main variables of interest and for what they will be used. This includes stating the desired quality of the simulations, i.e., what constitutes good performance in your particular modelling setup.
2. Characterisation of the data for calibrating and testing. Check the quality of the data (presence of outliers, inhomogeneities, etc.) and its extension (is it sufficient?). Select data subsets for calibration and validation.
3. Visual and other analysis to detect bad model behaviour and to characterise overall performance. Use plots, graphs, animations and other visual methods to gain an overview of overall performance.
4. Selection of basic performance criteria. Select metrics, such as RMSE and $r^2$, to measure model performance. Bennett et al. (2013) provide several methods for measuring quantitative model performance. The selected method needs to take into account the modelling objective and available data. For a comprehensive evaluation of models it is advisable to use systems of metrics (or multiple-objective functions) that measure different aspects of model performance.
5. Use of more advanced methods to solve modelling problems. Analyse the evaluation of model performance and assess whether any refinement is necessary in the process, going back to previous stages if necessary.

If the model does not perform well, this may indicate:

- the available data is insufficient to calibrate the model properly, and more data should be obtained;
- there are problems with the input data (there might be unaccounted biases or other issues);
- it could be advisable to use another calibration method (a local maximum might have been attained, instead of the global maximum);
- the model structure needs to be improved.

### 2.1.4 A note of caution

While models are invaluable tools for scientific research and management, they are not perfect and it is necessary for the modeller to be aware of their limitations. The absolute validation and verification of numerical models is impossible for epistemic reasons (Oreskes et al. 1994). A model cannot reproduce exactly the behaviour of a natural system because of our limited knowledge of the system. In fact, a model is an incomplete representation of reality that uses approximations and assumptions. In addition, measurements capture the state of the system in a limited way: they are subject to measurement error and there may be limitations in spatial or temporal resolution.

Thus, validation and verification are only conditional (Boschetti et al. 2011). They are conditional to the ranges of parameter values and input data used: model or data deficiencies not yet detected might be discovered in the future, when the model has been tested in other situations or with more data.

The model output also depends on the implicit and explicit assumptions made in its specification and how the behaviour of the system has been implemented. In fact, the model will not be able to reproduce a behaviour that has not been specified in its code. Also, the calibration data informs the values of the parameters. This means that the model will perform best when the behaviour of the system is similar to the behaviour reflected in the calibration data.

**Note:** A simulation is only as good as data and model specification allow it to be.

In order to guard against model and data imperfections it is best practice to estimate and communicate uncertainty.

## 2.2 Identifiability

An important concept in the modelling process is that of identifiability. Parameter identifiability analysis assesses whether it is possible to determine a unique vector of parameter values that optimises model performance or there are multiple parameter values that result in a similar model performance. The result of a lack of identifiability is the absence of a unique optimum combination of parameter values (the model performs with similar good performance for different combinations of parameter values) and in high error variances and covariances of parameters. Good introductions to the concept of identifiability are given by Beck (1987) and Guillaume et al. (2019).

> "Parameter identifiability analysis assesses whether it is theoretically possible to estimate unique parameter values from data, given the quantities measured, conditions present in the forcing data, model structure (and objective function), and properties of errors in the model and observations. In other words, it tackles the problem of whether the right *type* of data is available to estimate the desired parameter values." Guillaume et al. (2019, p. 418)

Thus, the analysis of identifiability is related to the design of experiments and monitoring programs, so that the collected data ensures optimal identifiability. If the model structure is suitable to the modelled system, obtaining more data should result in a reduced parameter uncertainty. However, this only happens if the measure is of the right type. This is also related to the concept of data worth.

In a linear system of equations, parameter values are identifiable when the number of unknowns is less than the number of observations. When the number of unknowns is greater than the number of observations, the system is underdetermined and the problem is ill-posed (and infinite solutions are possible).

In a complex nonlinear environmental models, instead, the previous rule is not applicable and more sophisticated methods are necessary. In this type of models we often find a lack of identifiability, since the complexity of the model is usually greater than the available information about the behaviour of the system. But having as much (or more) observations as parameters does not ensure they are identifiable. For example, a model may include descriptions of behaviours not present in the calibration data set, and thus their parameter values cannot be informed by observations. In other cases, it may not be possible to distinguish between several overlapping processes from the available data. Also, the model may even simulate state variables for which there are no available observations.

### 2.2.1 Sources of unidentifiability

We can find several types of unidentifiability:

- Structural or a priori unidentifiability. It is due to the model structure (conceptualisation of the system, equations, objective function, model of the error structure). This type of unidentifiability can be analysed irrespective of the availability of data. When there is structural unidentifiability, there is "a lack of information about one or more model parameters and results in unbounded variation of the parameters even in the absence of noise." Solving structural unidentifiability may involve acquiring data about other model variables or using a different model structure.
- Practical or a posteriori unidentifiability: it corresponds to unidentifiability due to other sources than model structure, mainly forcing dataset and model and observation errors.
  - Unidentifiability due to the forcing dataset. Complex models can include formulations describing different types of behaviours. If one of those behaviours is not represented in the calibration data, there is no information to estimate the value of the parameters specifically related to that behaviour. Solving this issue would require collecting data for different experimental or environmental conditions.
  - Unidentifiability due to model and observation errors. The presence of structural errors and measurement noise/error results in parameter uncertainty that may affect parameter identifiability. To improve the situation it may help analysing the influence of measurement error on parameter estimation.

**Note:** Structural unidentifiability implies practical unidentifiability. But structural identifiability does not imply practical identifiability.

### 2.2.2 Methods to analyse identifiability

Structural identifiability can be assessed by analysing the model's equations. For example, in the following equations the parameter values are unidentifiable:

$$Y = (\alpha + \beta)X$$
$$Y = \alpha\beta X$$

Infinite combinations of values for $\alpha$ and $\beta$ give the same result.

Identifiability can also be assessed using sensitivity analysis tools. If a the response of a model is insensitive to a parameter, the parameter is unidentifiable. However, a model may be sensitive to a given parameter and it may still not be possible to identify the parameter value.

Identifiability can also be analysed by visualizing the model's response surface with real or synthetic data. The existence of flat surfaces indicates unidentifiability, while different peaks with equal objective function values are indicative of local identifiability. For simple models, it is possible to visualize the response surface using 2D or 3D plots. For more complex models, it is possible to use dotty plots, which show the variation of the response variable for one or two parameters while varying all other parameters.

Finally, some indices have also been proposed to evaluate identifiability. It is the case of identifiability measure proposed by Doherty & Hunt (2009) based on the error variance analysis.

## 2.3 Calibration

Calibration, parameter estimation and history matching are three synonyms for the process of finding the optimal value of a model's parameters. Optimality is here defined as the minimisation of an objective function, representing some kind of distance between simulated and observed values. Usually the modeller aims to obtain a global optimum, and then assesses parameter and predictive uncertainty with respect to this optimum. The objective is that parameter and predictive uncertainty should be reduced after the calibration process.

**Global optimum:** best solution across the entire parameter space.

**Local optimum:** best solution in its immediate neighbourhood in the parameter space.

When the conditions are optimal, i.e., there is a clear global optimum and there are no local optima, an automatic search algorithm may be used.

However, the identification of an optimum may be hampered by the use of threshold parameters, by correlation between parameters, by autocorrelation and heteroscedasticity in the residuals and by insensitive parameters. These effects may cause local minima, valleys and plateaus in the parameter response surface. In such cases, global calibration methods (such as Differential Evolution) may be more appropriate. But there cases when even these methods may have problems in identifying a simple set of parameter values. Instead of an optimum there are several optima with similar performance. This is known as equifinality (Beven & Freer, 2001).

Some more sophisticated techniques try to address the equifinality problem. For example, Monte Carlo methods such as GLUE (Generalized Likelihood Uncertainty Estimation) (Beven & Binley, 1992). The GLUE method is not implemented in this package, but it can be applied easily by properly treating the output of the `monte_carlo()` function.

Also, the Iterative Ensemble Smoother, rather than finding an optimum set of parameter values, obtains several optimum sets of parameter values. The method draws several random samples of the prior parameter distributions and adjusts their values iteratively, so that each set of parameter values can be considered to be calibrated. The ensemble of optimized sets can be considered as a sampling of the posterior distribution.

## 2.4 Validation

Validation or, more appropriately, conditional validation consists in the evaluation of model performance to conditions that differ from calibration ones. The objective is to test whether the model performs equally well under a different set of circumstances.

If the simulated behaviour of the system is consistent with the observed behaviour, we cannot assume the model has been absolutely validated; it has only been validated for the sets of circumstances included in the calibration and validation data sets.

If the model results are incoherent with the observations, it would be necessary to review the data, model structure and parameter values to identify the source of the mismatch. In a sense, rather than confirming the model validity, the objective of validation may be seen as an attempt to falsify the model, to find areas where it needs improvement.

## 2.5 Sensitivity analysis

Should we vary slightly our modelling assumptions and the input data, the predicted behaviour would also vary. How much would the simulated behaviour vary if we varied model structure, parameter values and input data within reasonable ranges? Answering this question is the aim of the sensitivity analysis.

In other words, sensitivity analysis is the analysis of how the model's output varies with variations in the parameter values. This definition can be translated into practice in different ways:

- Defining sensitivity as the partial derivative of the output $Y$ with respect to a parameter $X_i$: $S_i = \delta Y / \delta X_i$. This is the usage in local sensitivity analyses (sensitivity is evaluated at a given point), and sensitivity indices are often estimated by varying one parameter at a time. However, in nonlinear models sensitivity varies within the parameter space.
- Linear regression coefficients can be considered as sensitivity indices in linear models. In particular, standardized regression coefficients (SRC) indicate the contribution of the variance of each input factor to the model output variance. The data for the linear regression can be obtained from Monte Carlo runs. Since SRCs are calculated over a range of values they may be considered as global sensitivity indices. In addition, linear regression models may be used to assess the degree of linearity of a model. If the coefficient of determination of the model is high (for example, $R^2 \geq 0.7$), the model may be considered to be linear and the SRCs may be used for sensitivity analysis (Hall et al. 2009).
- If the system is nonlinear ($R^2 < 0.7$), more sophisticated (global) methods will be necessary. Good introductions to global sensitivity analysis are given by Saltelli et al. (2004) and Saltelli et al. (2008). We mention here a few options:
  - Morris's method. It is a one-at-a-time method that analyses sensitivity at different points along the range of parameter values to obtain an average sensitivity measure. It has a low computational cost and is thus specially apt for models with a high computational time or there is a large number of uncertain parameters. Morris's method is implemented in this package.
  - Variance decomposition methods. The variance of the model output is decomposed into different components, each one attributable to one parameter (or combination of parameters). We have implemented Sobol's method in this package. It has a higher computational cost than Morris's method.
  - Monte Carlo filtering (also, Hornberger-Spear-Young method or Regional Sensitivity Analysis). A performance threshold is used to separate between "acceptable" and "unacceptable" simulations (and parameter sets). If there are differences between the distribution of parameter values for acceptable and unacceptable

simulations, the model is sensitive to that parameter. They have a very high computational cost (of the order of thousands of Monte Carlo runs).

**Note:** The results of the sensitivity analysis depend on the range of parameter values explored in the analysis, i.e., on the uncertainty of individual parameters.

Sensitivity analysis may be used to answer the following questions (Hall et al. 2009):

- Which are the factors that influence most the model output?
- Are there any factors we need to investigate more in depth to be more confident on the model output?
- Are there any factors which have an insignificant influence on model output, so that they can be discarded from subsequent analyses?
- Does the model reproduce known relations?
- Are there regions in the input space where the variation in the model output is maximum?
- Which are the optimal regions in parameter space for calibration?
- Are there any factors or group of factors that interact between themselves?
- Is model output robust to reasonable discrepancies in the value of input factors? Or is the model highly sensitive to small variations of the value of a parameter (which might indicate overfitting)?

Saltelli et al. (2019) analysed the main pitfalls in the use of sensitivity analysis and gave recommendations for improving its usage.

### 2.5.1 Sensitivity analysis and uncertainty analysis

Uncertainty analysis and sensitivity analysis are related concepts:

- Uncertainty analysis consists in the quantification of the uncertainties in model inputs and propagating them to obtain the uncertainty of model forecasts.
- Sensitivity analysis, instead, consists in assessing how variations in the model inputs, individually or in combination, influence the variation in the model outputs.

Thus, the focus of uncertainty analysis is on determining the limits of our knowledge (and lack of knowledge) about the system and its response. While sensitivity analysis is more centred on determining the factors that affect most the model output.

## 2.6 Uncertainty analysis

Environmental models cannot predict the future. What they provide are possible realistic behaviours that approach the behaviour of the real system. The uncertainty analysis deals with the confidence we can place on such forecasts and which are the ranges within which a predicted quantity may vary.

The uncertainty of model simulations has mainly three sources:

- Structural uncertainty, due to the uncertainty in the structure of the model. Since our knowledge of the system behaviour is incomplete, our implementation of the system processes is not perfect.
- Parametric uncertainty, due to the uncertainty about the value of model parameters. The available data and measurements of the system state are limited and they are a partial representation of the studied system. Thus, the estimated values of the parameters based on them are imprecise.

- Measurement error and natural variability. As a result, there is uncertainty in the boundary conditions of the model (initial state and forcing data) and system state. Natural variability includes different components, such as spatial heterogeneity, environmental variability and genetic variability.

Ideally, with time, we may reduce the uncertainty of simulations by improving our understanding of the system (adapting the structure of the model according to the available data), by reducing measurement error (with more precise data) and by identifying more precisely the value of the parameters (with more and more precise data). However, a certain degree of uncertainty will always remain.

Beck (1987) identified four types of problems that are the object of uncertainty analysis:

- Analysing the uncertainty about model structure (or about the relationships between variables characterizing the behaviour of the system). To answer this question we may use different models or model structures.
- Analysing the uncertainty about the value of the parameters. The objective is to obtain the distribution of parameter values, based on the analysis of which parameter values produce simulations that are consistent with observations of the system behaviour.
- Analysing the uncertainty of forecasts made using the model or predictive uncertainty. A priori predictive uncertainty is the uncertainty of forecasts made before incorporating the information of new measurements. A posteriori predictive uncertainty is the uncertainty of forecasts after incorporating new measurements.
- Designing experiments or monitoring programs to reduce the main uncertainties. This analysis is related with the analysis of identifiability and data worth.

### 2.6.1 Which uncertainty method to use?

We find two main approaches to the analysis of predictive uncertainty:

- First-order methods. Limited to the linear case, because of the intractability of mathematical solutions in nonlinear cases. Often there is an assumption of Gaussian probability density functions for better tractability. They use linear algebra to estimate uncertainty and require few simulations.
- Nonlinear methods. These methods can take into account all the complexity of the models. Some examples:
  - Monte Carlo methods. They are based in obtaining a large number of simulations by varying the value of the parameters of the model. They require a large number of simulations.
  - Iterative Ensemble Smoother. It is a technique specially designed to calibrate and quantify the uncertainty of nonlinear models with large numbers of parameters. The number of ensemble members should be greater than the dimensionality of the solution space. For models with a large number of parameters, the computational cost of IES in terms of number of model runs is much smaller than that of Monte Carlo methods.

Although environmental models are largely nonlinear, linear methods may also be applied to them if the uncertainty of model parameters is low enough. This is based on the fact that the behaviour of a nonlinear (continuously derivable) function in an interval around a certain point (the parameter value) can be assumed to be linear if the interval is small enough. We may consider a model has an approximately linear behaviour when the coefficient of determination of the linear regression model is higher than 0.7.

When choosing a method, we need to take into account computational cost (number of runs and time of calculation of the model) and parameter uncertainty. When the uncertainty about the model parameters is high, it is preferable to use nonlinear methods. In posterior iterations, when we have acquired more data and the uncertainty about the parameter values has decreased enough (and the response of the model can be assumed to be linear), we can use less costly linear methods.

### 2.6.2 Data worth

Data worth or value of information is an assessment of the value of data. The worth of data may be related to a cost-benefit analysis, where additional measurements have worth if the reduction in uncertainty overcomes the increased cost. Thus the concept of data worth is especially important for the design of measurement networks and data collection campaigns.

Data worth measures the potential of reduction of the uncertainty of relevant management predictions. There are two ways to assess data worth:

- as the decrease in predictive uncertainty produced if the data were added to the dataset;
- as the increase in predictive uncertainty produced if the data were removed from the dataset.

However, the most common situation is that new data is added to the system.

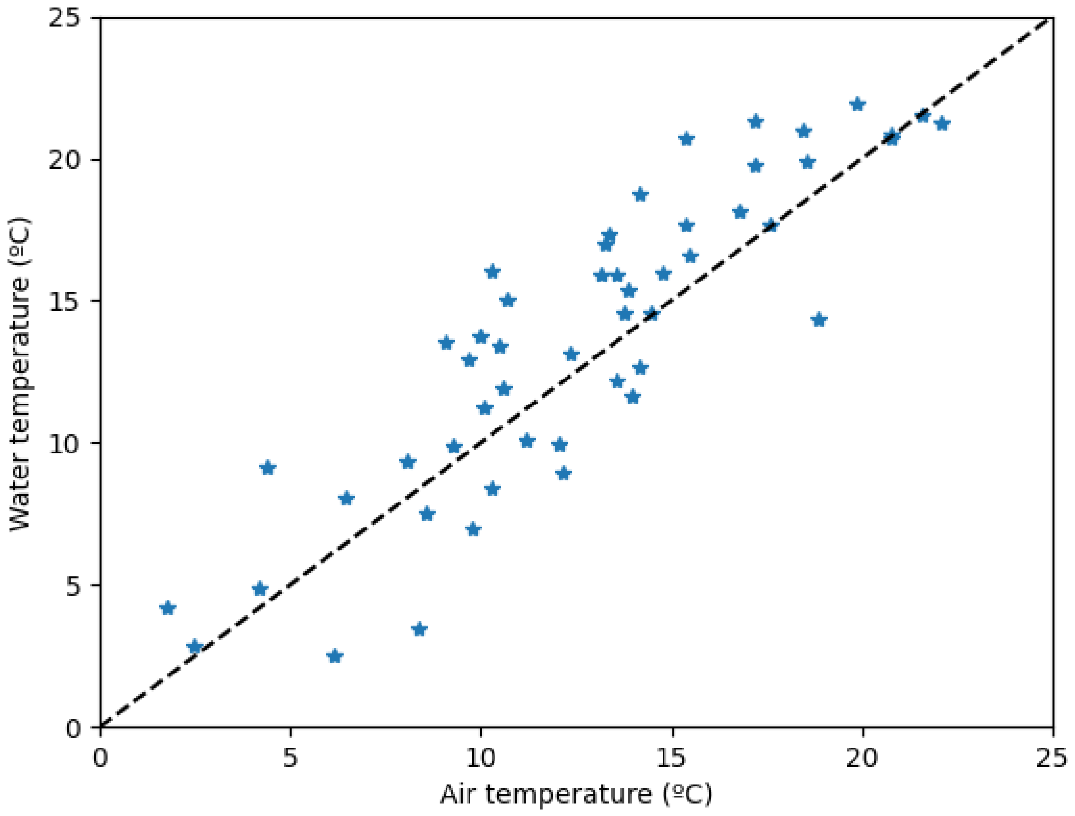
## 2.7 An example: simulation of surface water temperature

To demonstrate the concepts and techniques implemented in this package we will use the study case of the simulation of surface water temperature using one simple model depending on air temperature.

The relationship between air and water temperatures can be described by a non-linear equation. Mohseni et al. (1998) proposed the following sigmoid equation:

$$T_w = \mu + \frac{\alpha - \mu}{1 + e^{\gamma(\beta - T_a)}} \tag{2.1}$$

In this example, we will apply this model to the case of surface temperature in the reservoir of Naussac. We will use satellite measurements of surface temperature from the LakeSST dataset (Prats et al. 2018) and air temperature from the SAFRAN reanalysis (Quintana-Seguí et al., 2008), shown in the following figure. The data covers the period 1999-2015.



The objective of the exercise will be to estimate water temperature and its uncertainty on the 1st and 15th of each month in 2015.

### 2.7.1 Prior uncertainty

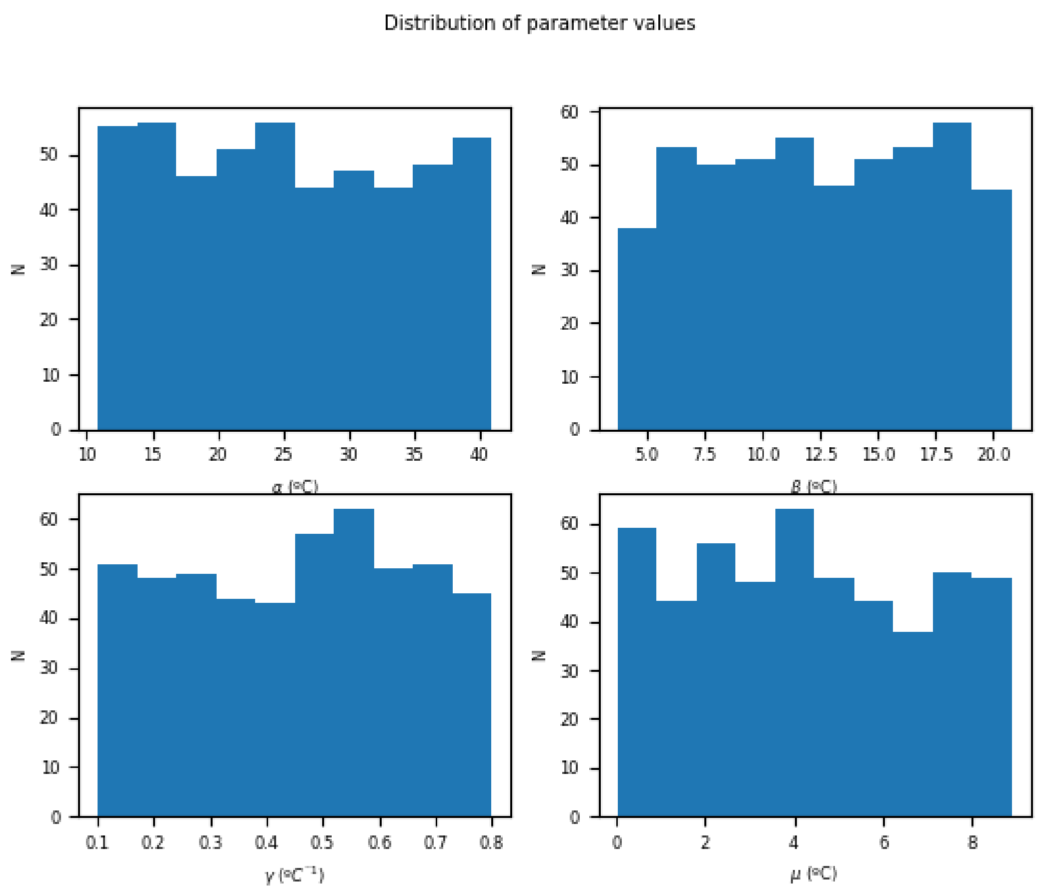
Assume we did not have any available data on surface temperature in the reservoir. And assume we need to estimate it. How confidently could we do it? We can answer this question with Monte Carlo simulations.

Even in the absence of field data, we could still make an estimation based on previous scientific knowledge. We know water temperature is highly correlated to air temperature and that it can be estimated quite accurately using Eq. (2.1). Also, many studies have calibrated that model in several instances, so that we can use expert judgement to provide an estimation of the expected value and range of the parameters of the model. The values on the following table are based on the results obtained by Mohseni et al. (1998).

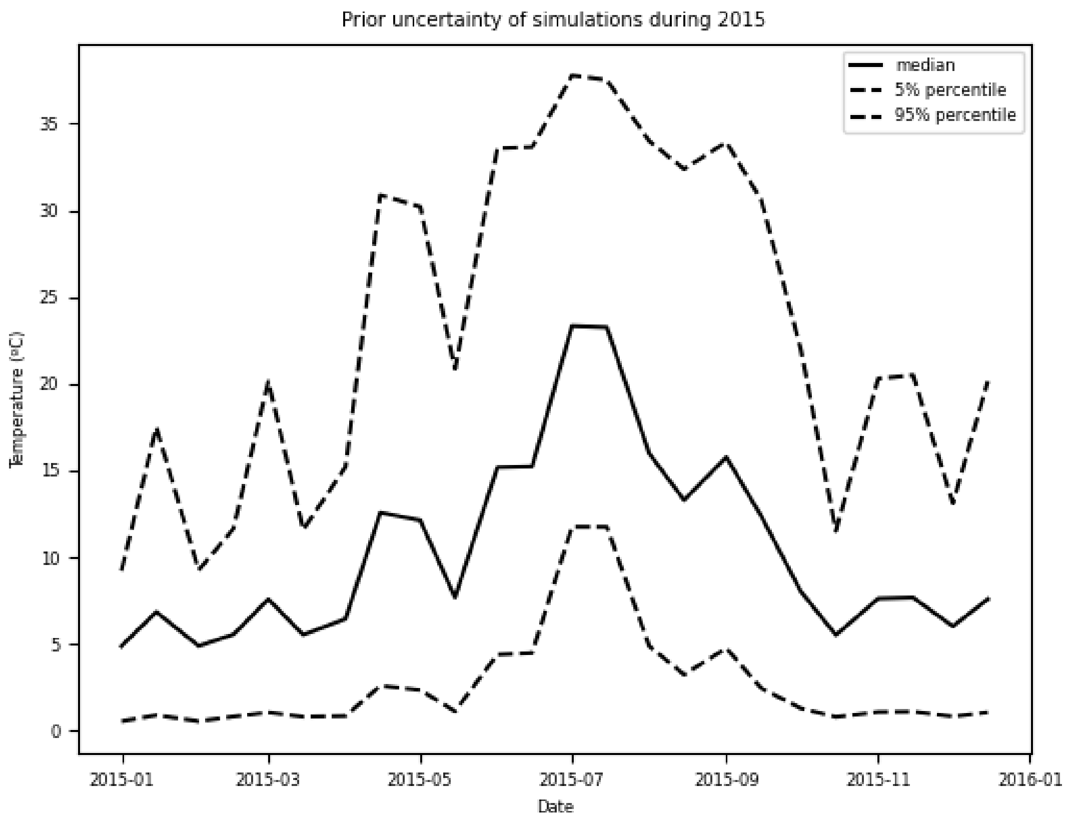
Table 2.3: A first (expert-based) estimation of model parameters

| Parameter | Mean | Min. | Max. |
|---|---|---|---|
| $\alpha$ (ºC) | 26.2 | 10.8 | 40.9 |
| $\beta$ (ºC) | 13.3 | 3.7 | 20.8 |
| $\gamma$ (ºC$^{-1}$) | 0.18 | 0.10 | 0.80 |
| $\mu$ (ºC) | 0.8 | 0.0 | 8.9 |

From these parameters we can draw several random samples (e.g., 500 samples) and carry simulations with each set of parameter values. In this case, since we suppose we do not have any additional information. When there is no information on the shape of the distribution of the parameter values, it is often assumed that they follow a uniform distribution, which can be considered a non-informative distribution.



The simulated values for the different parameter sets provide the prior uncertainty of the temperature estimations based on prior knowledge. We can see that, as expected, uncertainty is quite large at this point.



## 2.7.2 Sensitivity analysis

To reduce simulation uncertainty, we need to calibrate the model choosing appropriate values for the model parameters. However, before passing to calibration it is helpful to make a sensitivity analysis to find which are the parameters that have a greater influence in the simulation results. Given the large uncertainty at this point, a global sensitivity analysis method should be used. We will use here the method of Sobol.

The method of Sobol is a Monte Carlo-based method of decomposition of the variance. The objective of the method is to assess the proportion of the output variance that is due to each parameter. This method can reveal the existence of nonlinearity and interactions between parameters. When using this method usually two indexes are calculated: first-order sensitivity index ($S_i$) and total effect index ($S_{T_i}$).

Table 2.4: Sensitivity of model parameters (Sobol method, 400 samples)

| **Parameter** | $S_i$ | $S_{T_i}$ |
|---|---|---|
| $\alpha$ | 0.29 | 0.52 |
| $\beta$ | 0.37 | 0.55 |
| $\gamma$ | 0.03 | 0.08 |
| $\mu$ | 0.01 | 0.03 |

The first order sensitivity index $S_i$ measures the proportion of the variance of the model output that is explained by each parameter (without interactions). The index $S_i$ theoretically varies between 0 (unsensitive) and 1 (maximum sensitivity). The sum of all $S_i$ is equal to 1 for additive models (the effect of the parameters is independent), and it is less than 1 for nonadditive models (there are interactions between the parameters). In this case, the sum of all $S_i$ is 0.7.

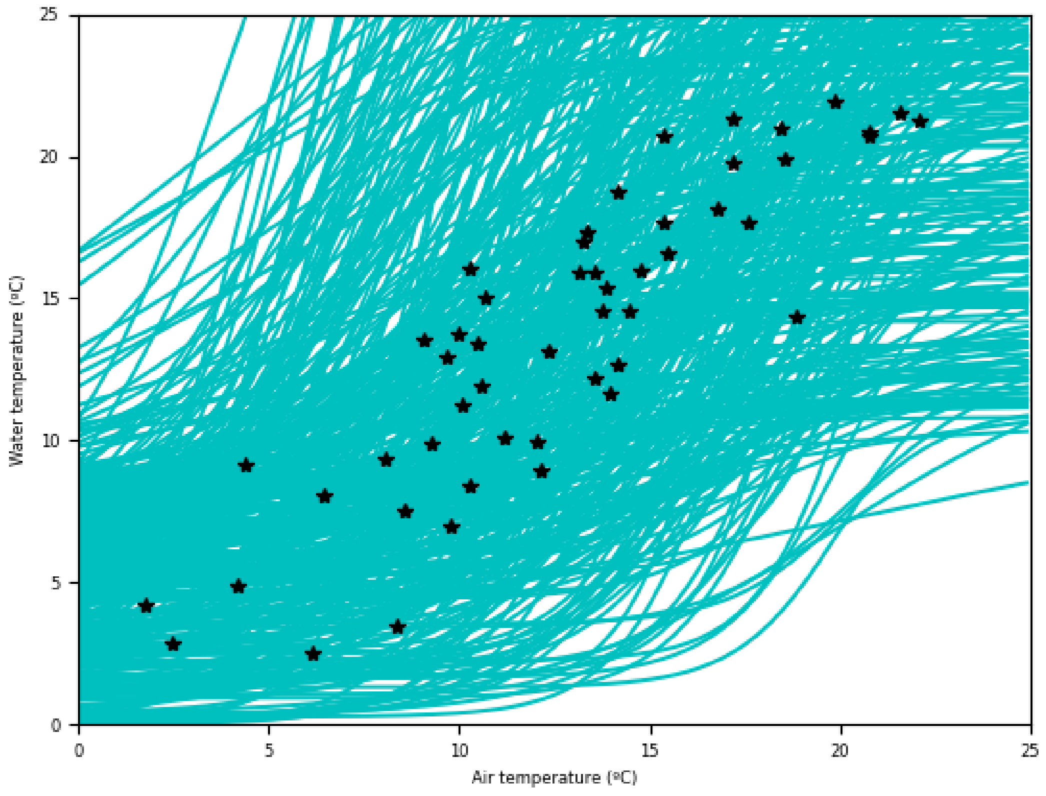
Thus, the variation in individual parameters explains 70% of the output variance and the model may be considered to have a largely linear behaviour, notwithstanding the rather large parameter space explored here.

**Note:** When $S_i$ is calculated using the Sobol's method, it may take negative values because of numeric inaccuracies when $S_i$ is near zero and the sample size is small.

The total effect index $S_{T_i}$ measures the contribution to the variance of the model output made by a parameter including interactions with other parameters. The index $S_{T_i}$ is equal to $S_i$ if there are no interactions between parameter $X_i$ and other parameters, and greater than $S_i$ if there are interactions with other parameters. If $S_{T_i} = 0$, the parameter $X_i$ is noninfluential. The sum of all $S_{T_i}$ is usually greater than 1; it is 1 only if the model is perfectly additive (no interactions between parameters).

In our example, the parameters $\alpha$ and $\beta$ are the most influential parameters: fixing the value of one of them would reduce the output variance by more than 50%. The paramater $\gamma$ is much less influential. And the parameter $\mu$ is noninfluential since its total effect index is nearly zero. Also, since the sum of $S_i$ is less than 1, the sum of $S_{T_i}$ is greater than 1, and $S_{T_i} > S_i$ for the parameters $\alpha$, $\beta$ and $\gamma$, there are interactions between these parameters.

The following figure shows the 400 random realizations used to calculate Sobol indexes superimposed on measurements. The figure shows visually that there is a large uncertainty in the value of the parameters $\alpha$ and $\beta$, which determine, respectively, the upper asymptote and the inflexion point of the logistic curve.

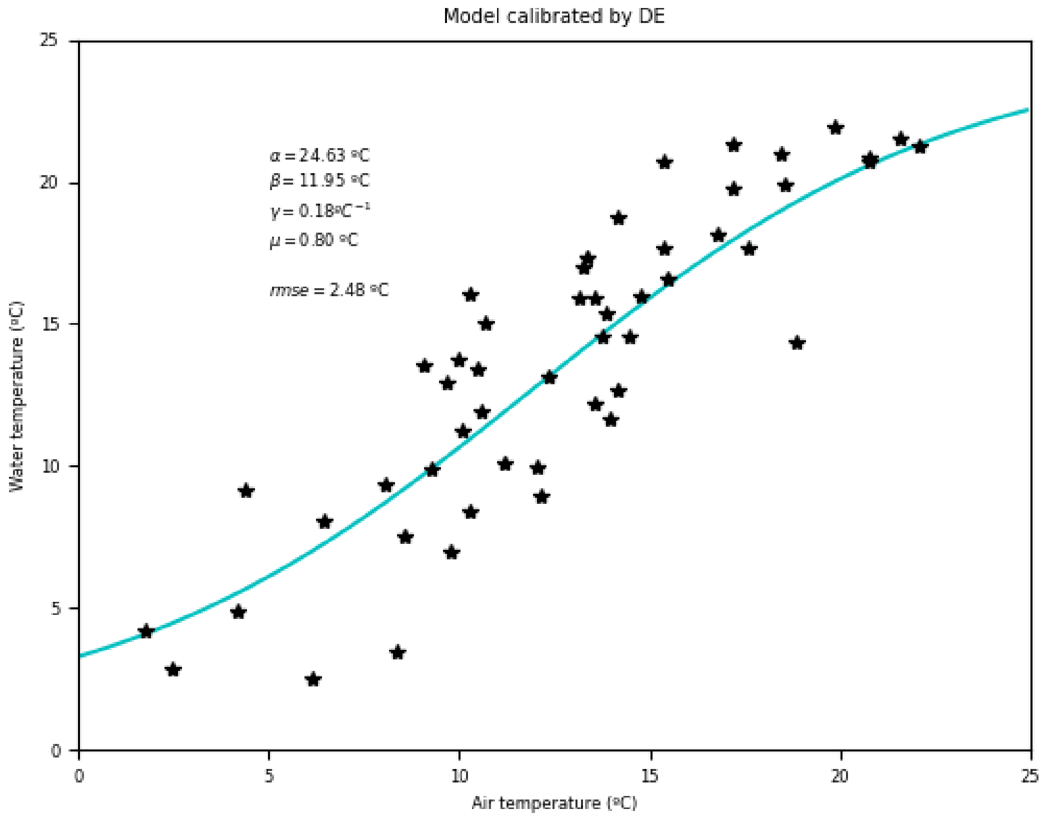


### 2.7.3 Calibration

In the previous section, we have seen that the parameters $\gamma$ and $\mu$ are mostly noninfluential. This means that multiple combinations of the values of these parameters results in simulations of similar quality, thus increasing the uncertainty of parameter values. To reduce this uncertainty, we may fix the value of the parameters $\gamma$ and $\mu$ to their average values, and calibrate the value of the parameters $\alpha$ and $\beta$.

The previous process of fixing the value of some parameters is an example of (manual) regularisation, a process by which the complexity of a model is reduced so that the remaining parameters can be estimated uniquely (i.e., with low uncertainty). Other more sophisticated regularisation methods include Tikhonov regularisation and subspace regularisation.

Since there is a great uncertainty on the value of the parameters, we will use Differential Evolution (DE) to make a first calibration. DE is a global calibration method that explores the whole parameter space, rather than a local range of parameter values. Thus this method provides a global optimum.



After applying a global calibration method it is advisable to refine the calibration with a local method. Thus, we repeat the calibration with the GLM algorithm, but now without fixing the parameters $\gamma$ and $\mu$. We can see there is just a slight improvement in RMSE.

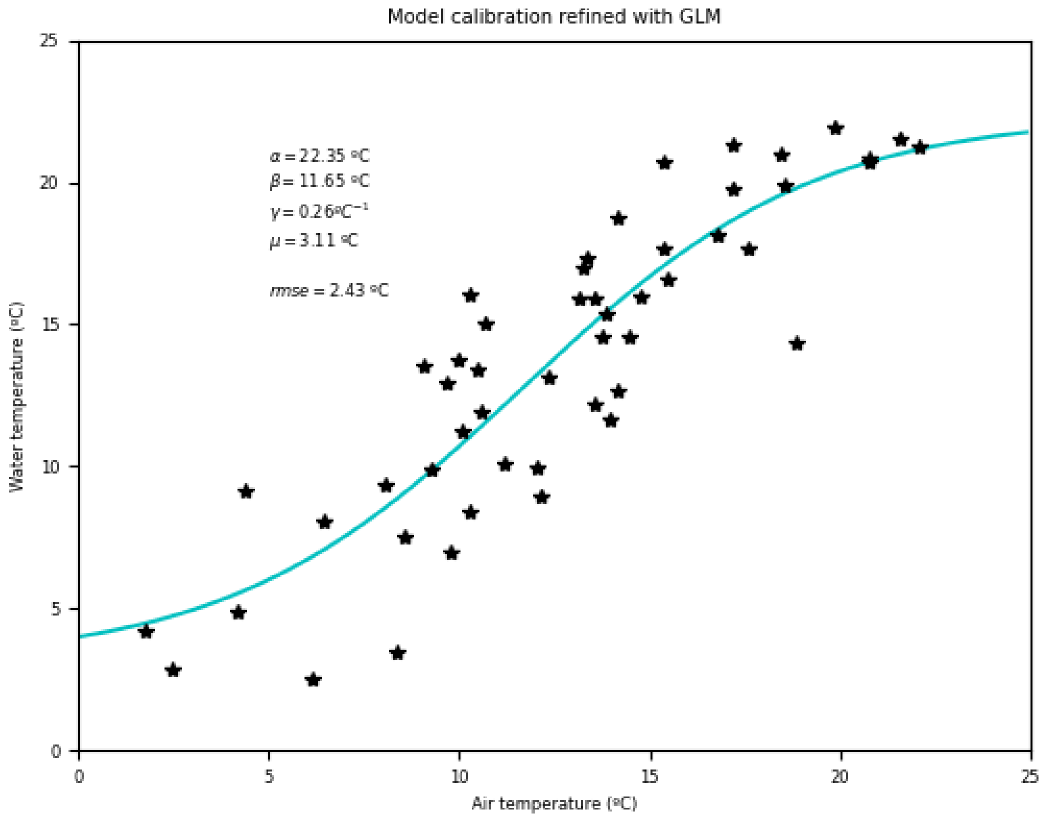


Table 2.5: Calibrated values of model parameters

| Parameter | Mean | 2.5% confidence value | 97.5% confidence value |
|---|---|---|---|
| $\alpha$ (ºC) | 22.3 | 20.2 | 24.4 |
| $\beta$ (ºC) | 11.7 | 10.6 | 12.8 |
| $\gamma$ (ºC$^{-1}$) | 0.26 | 0.19 | 0.34 |
| $\mu$ (ºC) | 3.2 | 0.6 | 5.7 |

### 2.7.4 Posterior uncertainty (Schur's complement analysis)

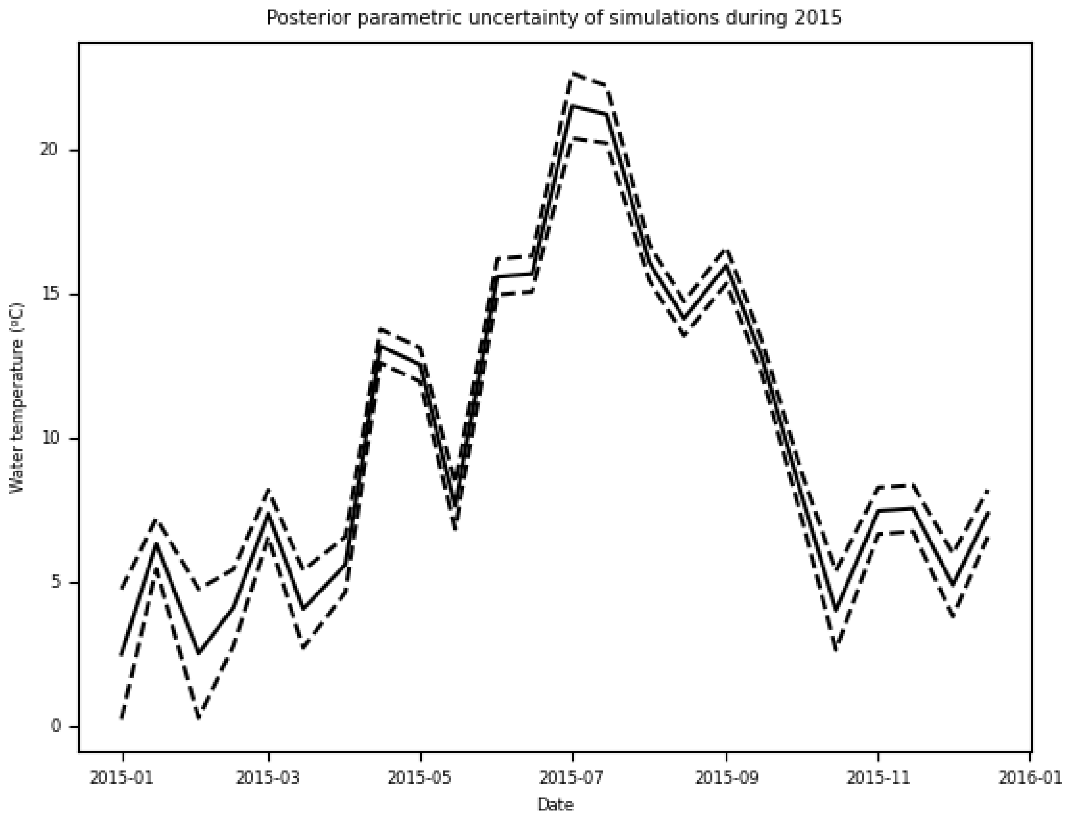
Schur's complement analysis is valid when the model has linear (or almost linear) response to variations in the value of the parameters. We already showed above that the model has a nearly linear behaviour in the prior parameter space using Sobol's uncertainty method of decomposition of variance. The following table shows the reduction in parameter uncertainty through calibration according to Schur's complement analysis.

Table 2.6: Reduction of parameter uncertainty by calibration

| Parameter | Prior variance | Posterior variance | Reduction in par. uncertainty (%) |
|---|---|---|---|
| $\alpha$ | 56.6 | 1.0 | 98 |
| $\beta$ | 18.3 | 0.4 | 98 |
| $\gamma$ | 0.031 | 0.002 | 93 |
| $\mu$ | 4.95 | 1.56 | 69 |

After calibration the uncertainty of the parameters has been much reduced. We note that the reduction in uncertainty for $\mu$ is less important than for the other parameters. In fact, this parameter corresponds to the lower asymptote of the logistic curve, but there are few low temperature measurements to inform this parameter.

The reduction in the uncertainty of the parameter values results in a decrease in the uncertainty of the simulations. Please note, though, that in this case the uncertainty is the uncertainty due to parametric uncertainty only and does not take into account measurement error or model inadequacies.



We note that uncertainty is not equal along the year. Uncertainty is greater for periods of high and, especially, for low temperatures.

### 2.7.5 Data worth

The following figure shows the percent reduction in uncertainty by parameter and forecast date. It can be seen that forecasts on days with lower temperatures are influenced mostly by $\mu$ and forecasts on days with the highest temperatures are influenced mostly by $\alpha$. This is quite logical since $\alpha$ determines the higher asymptote of the logistic curve and $\mu$ determines the lower asymptote.

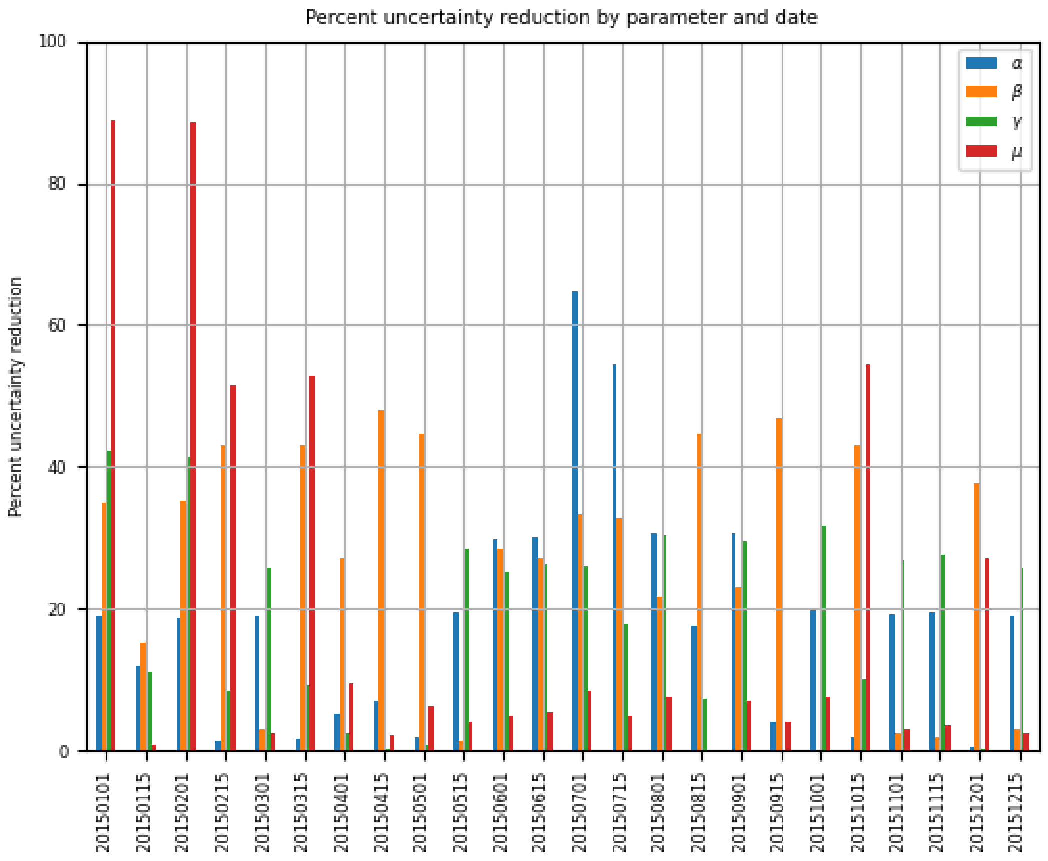
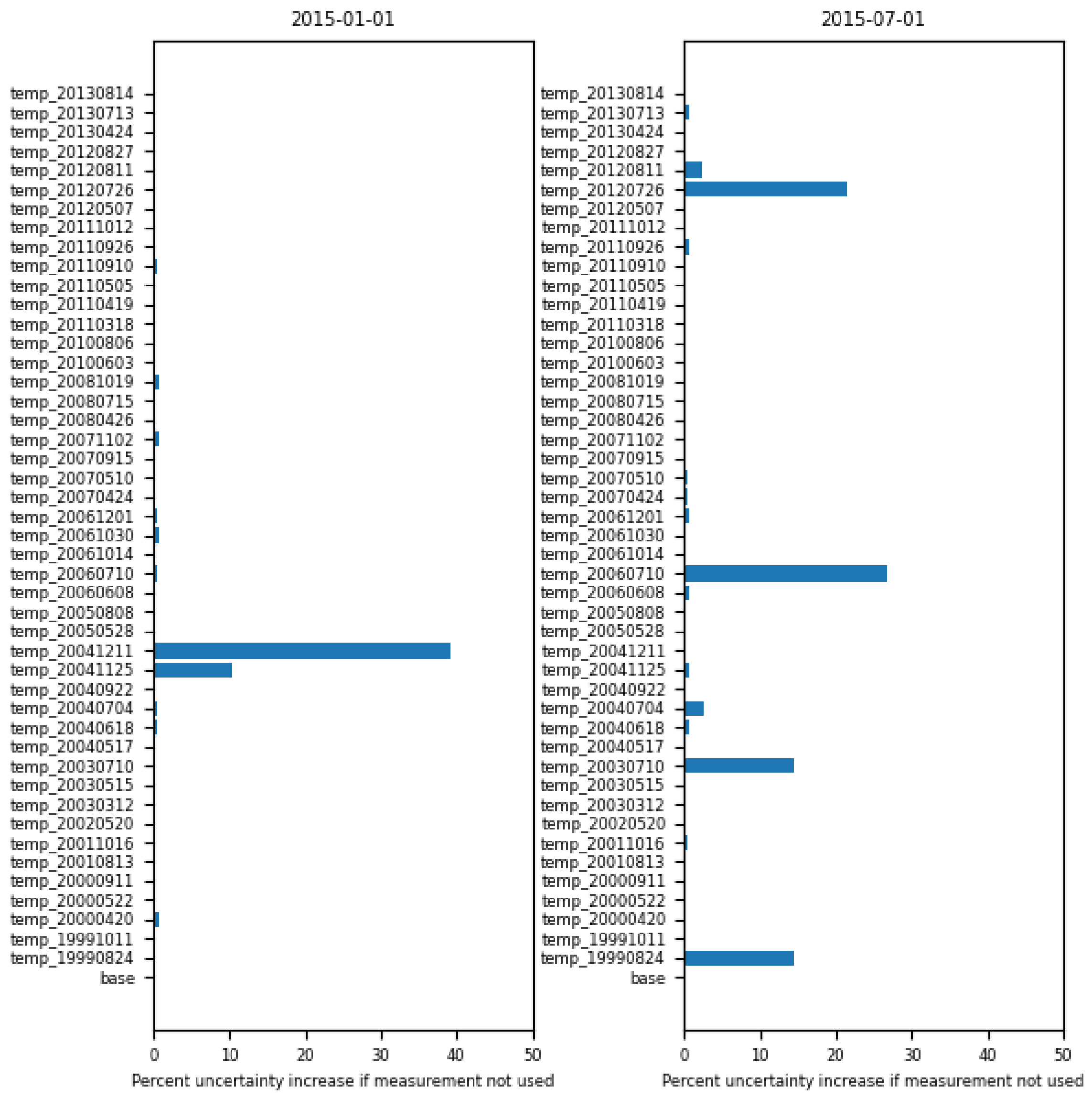


Fig. 2.1: This figure shows the percentage reduction in uncertainty that we would obtain in the estimation of the value of the parameters by acquiring a measurement on the given days.

If we wanted to reduce the uncertainty of winter and summer forecasts, which measurements would we need to take? We can investigate this question with the concept of data worth. In the next figure we show data worth as the percent uncertainty increase when removing an observation for two forecasts, for 1 January 2015 and for 1 July 2015. We see the most valuable data to reduce forecast uncertainty is that which corresponds to the same period of the year. We would thus need to concentrate our measurement efforts in the summer and winter.



## 2.7.6 Posterior predictive error (error variance analysis)

Error variance analysis is an alternative linear uncertainty analysis method that decomposes the variance of predictive error into three components:

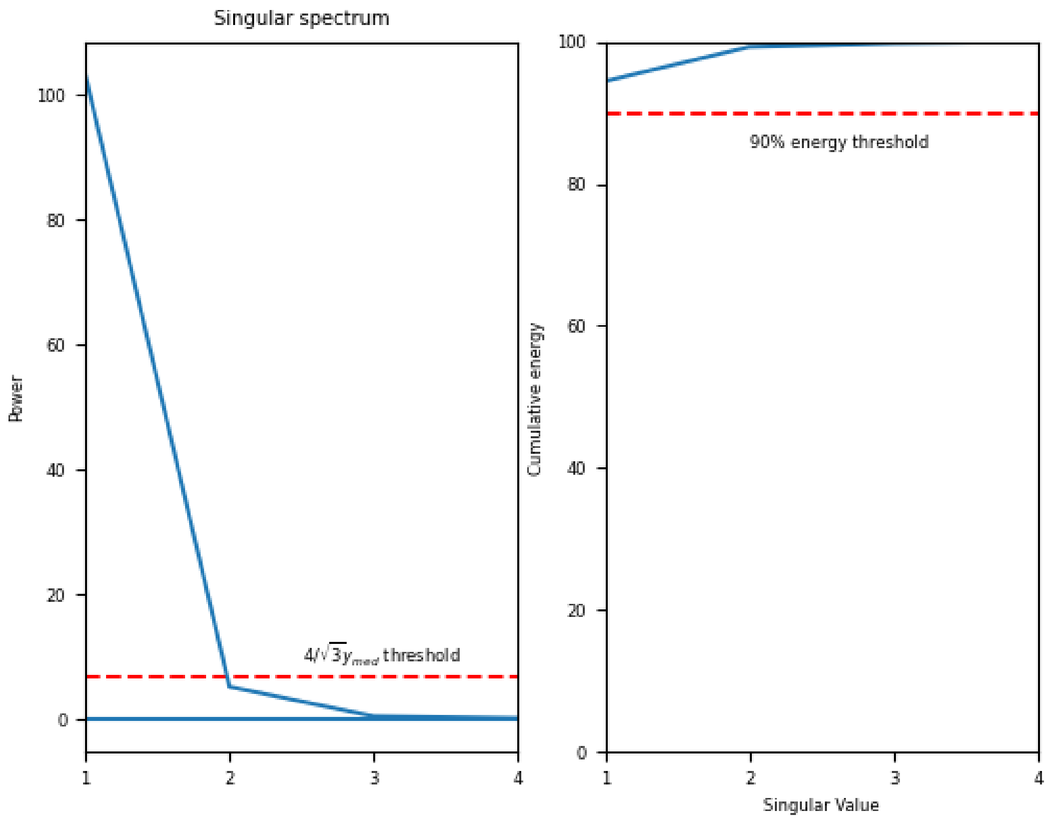
- Potential forecast error arising from parameter uncertainty.
- Potential forecast error arising from model-to-measurement misfit (including measurement noise and structural noise).
- Potential forecast error arising from not calibrating all model parameters.

Thus, posterior predictive error is expected to be greater than posterior uncertainty.

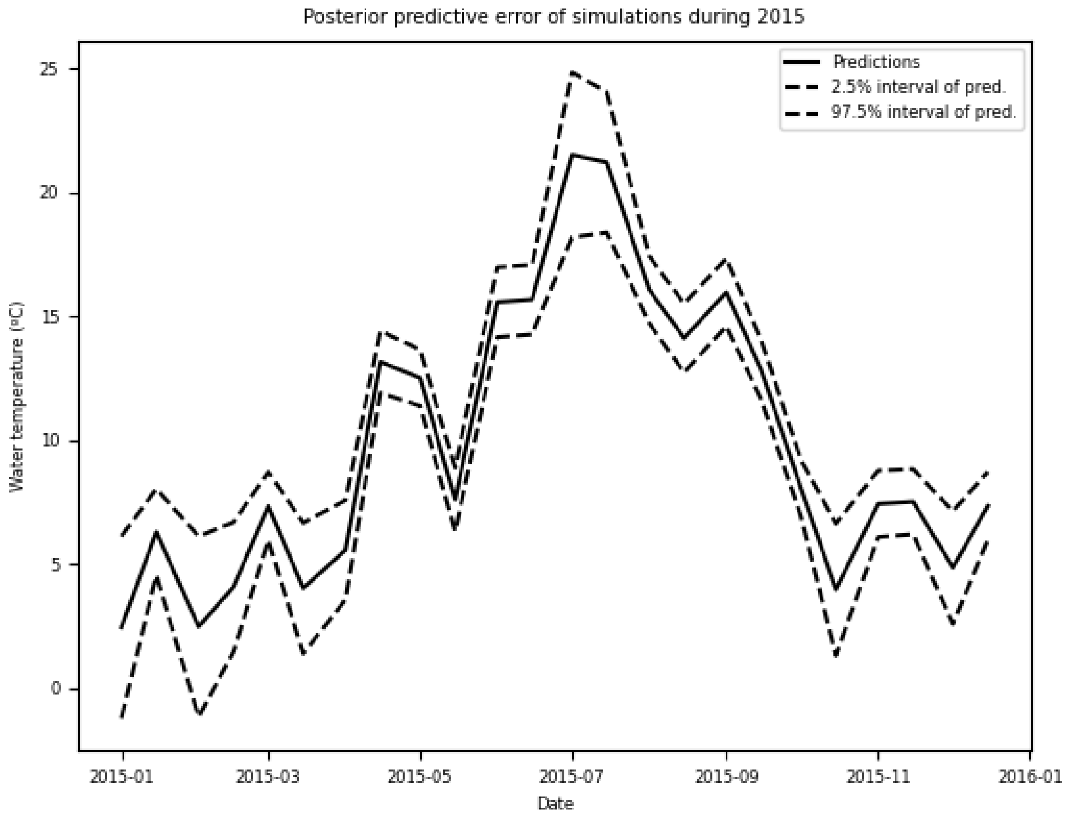
However, one of the difficulties in using error variance analysis lies in estimating the number of singular values to use, which may introduce a certain degree of subjectivity. The error variance analysis uses singular value decomposition (SVD) to estimate the effective dimensionality of the system and different criteria have been proposed to choose how many singular values should be used:

- Identifying the place with a sharp elbow.
- Use as many singular values as necessary to account for at least 90 % (or 99%) of total energy.
- When residuals have mean zero and are uncorrelated (white noise), Gavish & Donoho (2014) proposed an optimal threshold of $2.858 y_{med}$, where $y_{med}$ is the median singular value.

The rational is that the few singular values kept are enough to characterise most of the information, while the discarded singular values represent mostly noise.



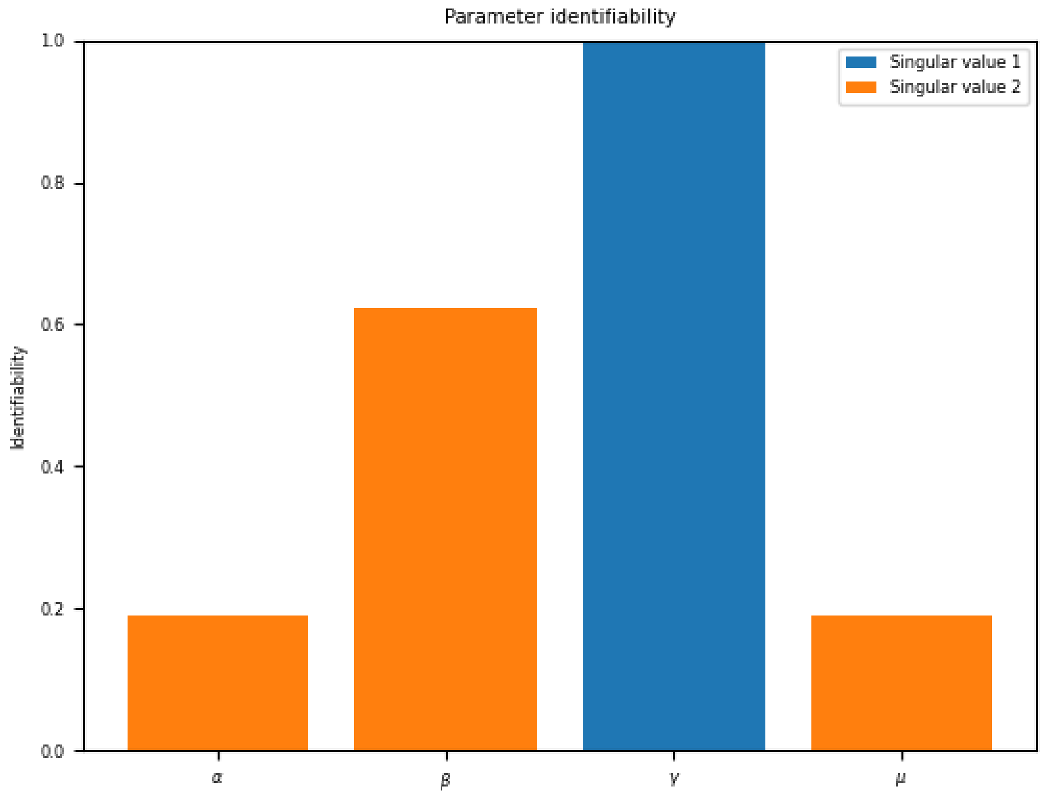
In our study case, we would need 2 singular values according to the 90% energy criterion. But only 1, according to Gavish & Donoho's hard threshold. However, this threshold might not be appropriate, since we expect the residuals to be correlated. Besides, the power of the second singular value is quite near to this threshold. Thus, we will use 2 singular values in this case.



According to these results, predictive error is greater in the summer and in the winter, in coherence with the results for posterior uncertainty.

### 2.7.7 Identifiability

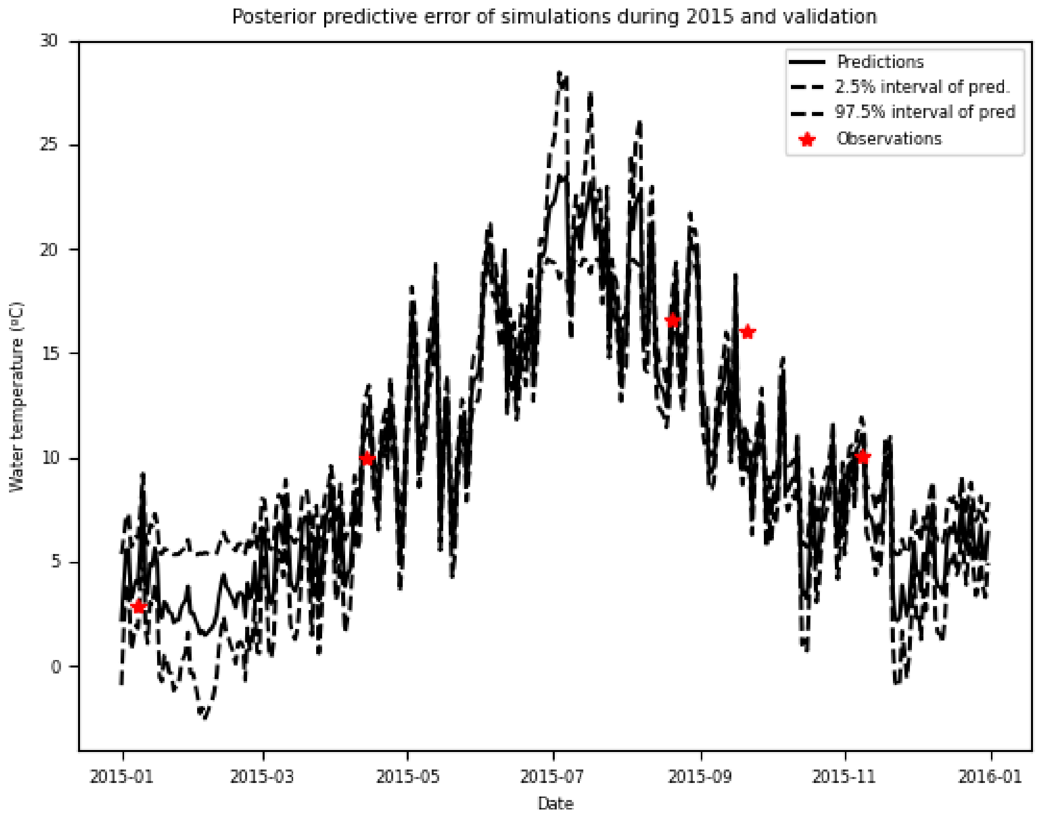
Error variance analysis also provides an estimation of parameter identifiability. According to parameter identifiability results (again, using 2 singular values), the calibration data allow to determine uniquely the value of $\gamma$ (identifiability near to one). However, for the other parameters the information is shared between them, and the calibration data only allows to estimate a combination of these parameters. This means that the solution obtained for the parameters $\alpha$, $\beta$ and $\mu$ is not unique, but one of several possibilities with a similar error level. Also, parameters are correlated among themselves.



The parameters with highest identifiability in the study case are $\gamma$ and $\beta$, which determine the slope and location of the curve around its inflexion point. But the available data does not allow to determine as well the location of the S-curve asymptotes indicated by $\alpha$ and $\beta$.

### 2.7.8 Validation

Validation may be considered as a test to assess the degree of coherence of simulations with the observed behaviour. The following figure shows temperature measurements taken in 2015, superimposed on simulations and predictive uncertainty for all days in the same year.

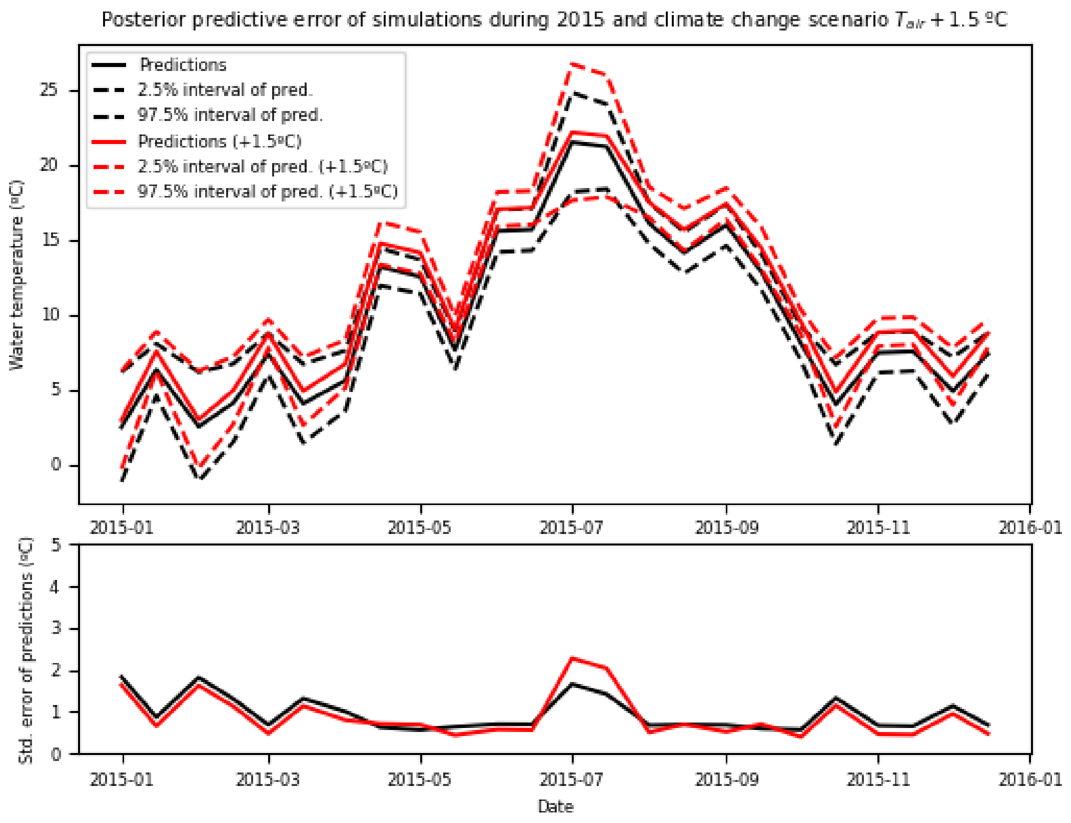


We see that the predictions are quite consistent with observations: most measurements are within the estimated ranges of predictive error. However, the fact that the measurement for 21 Sept. 2015 is outside the predictive error range may indicate a model inadequacy.

In fact, the variability of estimated water temperature in spring and autumn seems too large. Water temperature varies more slowly than air temperature, and daily water temperature is best correlated to the average of air temperature in the previous days. This is the rational for the modification proposed by Koch & Grünewald (2010) to Eq. (2.1).

### 2.7.9 Using the results in a management setting: effect of climate change

We can use the results obtained here to analyse a management question: which would be the effect of climate change? To do this we will use the same air temperatures for 2015 but increased by 1.5 ºC to simulate water temperature using eq. (2.1).



The model predicts an average increase of 1.1 ºC in water temperature with a 1.5 ºC increase in water temperature. During most of the year predictive uncertainty in the $T_{air} + 1.5$ ºC scenario is equal or lower to that during the base scenario, with the exception of July when it increases. It is also the month when uncertainty was already highest for the base scenario. This is related to the difficulty in obtaining a precise value for the parameter $\alpha$ and the fact that the warming makes summer temperatures leave the range of values for which there is information. This points to a critical issue: the difficulty in predicting water temperature is greatest when there is a higher risk of high temperatures.

### 2.7.10 A note on the use of linear an nonlinear uncertainty methods

Some of the previous analyses are based on the use of linear uncertainty methods. The application of such methods to nonlinear models provides approximate results only. However, although these methods do not allow an exact estimation of uncertainty, they provide semi-quantitative information sufficiently robust to analyse the relative contributions of different sources of uncertainty or parameters (Moore & Doherty, 2005).

Nonlinear methods (such as Monte Carlo methods) could also be used to obtain more appropriate quantitative results. However, they require a much higher number of simulations, which may be prohibitive if the model run time is large. Such methods allow analysing the effect of parametric uncertainty as shown above (see *Prior uncertainty*). Using such methods, the effect of noise (measurement noise of model structural noise) can also be taken into account by appropriately characterising it, for example, as white noise (if noise is uncorrelated) or as a stochastic process. But to determine the characteristics of noise may require a large dataset.

Thus, to choose between a linear or nonlinear method it is necessary to take into account the model run time and computational cost of the method and the available data.

## 2.8 References

- Bennett, N.D.; Croke, B.F.W.; Guariso, G.; Guillaume, J.H.A.; Hamilton, S.H.; Jakeman, A.J.; Marsili-Libelli, S.; Newham, L.T.H.; Norton, J.P.; Perrin, C.; Pierce, S.A.; Robson, B.; Seppelt, R.; Voinov, A.A.; Fath, B.D.; Andreassian, V. (2013) Characterising performance of environmental models. *Environmental Modelling & Practice* 40, 1-20.
- Beck M.B. (1987) Water Quality Modeling: A Review of the Analysis of Uncertainty. *Water Resources Research* 23(8), 1393-1442.
- Beven, K.; Binley, A. (1992) The future of distributed models: model calibration and uncertainty prediction. *Hydrological Processes* 6, 279-298.
- Beven, K.; Freer, J. (2001) Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology. *Journal of Hydrology* 249, 11-29.
- Beven, K.; Young, P. (2013) A guide to good practice in modeling semantics for authors and referees. *Water Resources Research* 49, 5092-5098.
- Boschetti, F.; Grigg, N.J.; Enting, I. (2011) Modelling = conditional prediction. *Ecological Complexity* 8, 86-91.
- Doherty, J.; Hunt, R.J. (2009) Two statistics for evaluating parameter identifiability and error reduction. *Journal of Hydrology* 366, 119-127.
- Gavish, M.; Donoho, D.L. (2014) The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory* 60(8), 5040-5053.
- Guillaume, J.H.A.; Jakeman, J.D.; Marsili-Libelli, S.; Asher, M.; Brunner, P.; Croke, B.; Hill, M.C.; Jakeman, A.J.; Keesman, K.J.; Razavi, S.; Stigter, J.D. (2019) Introductory overview of identifiability analysis: A guide to evaluating whether you have the right type of data for your modeling purpose. *Environmental Modelling & Software* 119, 418-432.
- Hall, J.W.; Boyce, S.A.; Wang, Y.; Dawson, R.J.; Tarantola, S.; Saltelli, A. (2009) Sensitivity analysis for hydraulic models. *Journal of Hydraulic Engineering* 135(11), 959-969.
- Jakeman, A.J.; Letcher, R.A.; Norton, J.P. (2006) Ten iterative steps in development and evaluation of environmental models. *Environmental Modelling & Software* 21, 602-614.
- Koch, H.; Grünewald, U. (2010) Regression models for daily stream temperature simulation: case studies for the river Elbe, Germany. *Hydrological Processes* 24, 3826-3836.
- Moore, C.; Doherty, J. (2005) Role of the calibration process in reducing model predictive error. *Water Resources Research* 41, W05020, 14 p.
- Oreskes, N.; Shrader-Frechette, K.; Belitz, K. (1994) Verification, validation, and confirmation of numerical models in the Earth sciences. *Science* 263, 641-646.
- Prats, J.; Reynaud, N.; Rebière, D.; Peroux, T.; Tormos, T.; Danis, P.-A. (2018) LakeSST: Lake Skin Surface Temperature in French inland water bodies for 1999-2016 from Landsat archives. *Earth System Science Data* 10, 727-743.
- Quintana-Seguí, P.; Le Migno, P.; Durand, Y.; Martin, E.; Habets, F.; Baillon, M.; Canellas, C.; Franchisteguy, L.; Morel, S. (2008) Analysis of near-surface atmospheric variables: validation of the SAFRAN analysis over France. *Journal of Applied Meteorology* 47, 92-107.
- Saltelli, A.; Tarantola, S.; Campolongo, F.; Ratto, M. (2004) *Sensitivity Analysis in Practice. A Guide to Assessing Scientific Models.* John Wiley & Sons, Ltd, Chichester. 219 p.
- Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F.; Cariboni, J.; Gatelli, D.; Saisana, M.; Tarantola, S. (2008) *Global Sensitivity analysis. The Primer.* John Wiley & Sons, Ltd, Chichester. 292 p.

- Saltelli, A.; Aleksankina, K.; Becker, W.; Fennell, P.; Ferretti, F.; Holst, N.; Li, S.; Wu, Q. (2019) Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices. *Environmental Modelling & Software* 114, 29-39.

# CHAPTER THREE

# INSTALLATION

## 3.1 Requirements and dependencies

You need Python 3.6 or later to run the `cuspy` package. You can have multiple Python versions (2.x and 3.x) installed on the same system without problems using, for example, virtual environments.

You may use `pip` to install the required packages.

**Note:** In Windows, to use `pip` to install the required packages, please verify that the path to `pip`'s executable file has been added correctly to your environment variable `PATH`. The address may vary depending on whether you install it for one or all users (e.g., `%USERPROFILE%/AppData/roaming/Python/Python37/scripts` or `C:/Users/MyUserName/AppData/Local/Programs/Python/Python37/Scripts`).

The application `cuspy` depends on the Python packages `numpy`, `pandas` and `pyemu`. The package `pyemu` also depends on `matplotlib` for plotting. Make sure they are installed before using `cuspy`.

In addition, `cuspy` relies on the use of the PEST++ executables. PEST++ executables for Windows are available at https://github.com/usgs/pestpp/bin/win, but you will need to compile them for Linux (see the instructions below, *Compiling PEST++ executables*).

If you do not have to compile the documentation, `sphinx` is not necessary to use `cuspy`, since a precompiled pdf copy of the documentation is included in the folder `docs`.

If you need to compile the documentation from source (e.g., to modify it), you will need the package `sphinx`. To install `sphinx` just make:

```
pip install sphinx
```

The package `sphinx` works by default with reStructuredText documents. However, it can also recognise Markdown by installing the Markdown parser `recommonmark`.

```
pip install --upgrade recommonmark
```

To create multilingual documentation, you will need the package `sphinx-intl`. The installation process is as above:

```
pip install sphinx-intl
```

To compile pdf documents (only in Linux), you will also need to have installed latex and the Python package `latexmk`. To install latex, type:

```
sudo apt-get install texlive-full
```

And to install `latexmk`, type:

```
pip install latexmk.py
```

On Ubuntu you may use:

```
sudo apt install latexmk
```

## 3.2 Cloning repositories

You need to clone the `cuspy` package and `pestpp` source with git. For this go to an appropriate directory (e.g., `pathtoprojectsfolder`) where to copy the project's code in a subfolder and clone the project. For example, for `cuspy`:

```
cd pathtoprojectsfolder
git clone https://github.com/inrae/ALAMODE-cuspy
```

This command creates the `cuspy` repertory in the folder `pathtoprojectsfolder`.

To install the development branch of the project, after cloning the `cuspy` package, change to the `dev` branch:

```
cd cuspy
git checkout dev
```

To clone the `pestpp` source, do:

```
cd pathtoprojectsfolder
git clone https://github.com/usgs/pestpp
```

## 3.3 Compiling PEST++ executables

If you work on Linux, you need to compile the PEST++ executables. For this you will need to have `gcc`, `gfortran` and the libraries `lapack` and `blas` installed. To install them you can use:

```
sudo apt-get install gcc gfortran
sudo apt-get install liblapack-dev libblas-dev
```

To compile the PEST++ executables, go to the `pestpp/src` folder and do:

```
make clean
STATIC=no make install
```

The compiled executables can then be found in the folder `pestpp/bin/linux`.

## 3.4 Installing cuspy

To install cuspy, go to the repertory created during the cloning of the package cuspy (e.g., pathtorepertorycuspy) and install it using pip:

```
cd pathtorepertorycuspy
pip install -U .
```

## 3.5 Compilation of the project documentation

The source files for the project user manual are stored in the folder pathtorepertorycuspy/sphinx-doc/source. Sphinx also extracts data from the project modules docstrings.

### 3.5.1 Documentation in English

To compile the user manual in English as html files go to the folder pathtorepertorycuspy/sphinx-doc and type:

```
make html
```

The output html files are saved in the folder pathtorepertorycuspy/sphinx-doc/build/html.

You can also compile the user manual as a pdf file making:

```
make latexpdf
```

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder pathtorepertorycuspy/sphinx-doc/build/latex.

### 3.5.2 Documentation in French

To compile the user manual in French as html files go to the folder pathtorepertorycuspy/sphinx-doc and type:

```
sphinx-build -b html -aE -D language='fr' -c source/locale/fr source build_fr/html
```

The output html files are saved in the folder pathtorepertorycuspy/sphinx-doc/build_fr/html.

To compile the pdf documentation, type the following commands:

```
sphinx-build -b latex -aE -D language='fr' -c source/locale/fr source build_fr/latex
cd build_fr/latex
make
```

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder pathtorepertorycuspy/sphinx-doc/build_fr/latex.

CHAPTER

# FOUR

# CODING STYLE GUIDE

## 4.1 Python code

We have followed the PEP 08 – Style Guide for Python Code.

## 4.2 Docstrings

We have followed the PEP 257 – Docstring Conventions and the recommendations of the *Google Python Style Guide* for docstrings in functions (point 3.8.1).

According to the GNU General Public License, a copyright notice is included in every module.

## 4.3 Documentation

Some help documents have been written using markdown (README.md, DISCLAIMER.md).

The user manual has been generated using `sphinx` from files using reStructuredText markup language. The documentation translated into French has been created using `sphinx-intl`.

CHAPTER

FIVE

# TRANSLATION OF THE PROJECT'S DOCUMENTATION

The `cuspy` package is originally written in English and translated into French. This section describes the process to translate the *User and Developer Guide* by using pot and po files and the package `sphinx-intl`. Using this method only the modified sections of text need to be translated again when modifications are made to the original documentation. It also facilitates the translation of docstrings and module documentation. The translation instructions are based on https://sphinx-doc.org/en/master/usage/advanced/intl.html.

## 5.1 Preliminary steps

Before starting the translation it is advisable to check the source files thoroughly to eliminate any language or format issues.

A glossary of technical terms has been created to improve the consistency of the translation. It is located in the folder `sphinx-doc/glossaries`.

## 5.2 Creation of POT and PO files

The translation of the documentation files is based on the extraction of the translatable text into pot (portable object template) files and the creation of translated text in po (portable object) files. For this, the package `sphinx-intl` is used.

To extract the document's translatable messages into pot files, change to the `sphinx-doc` directory in the terminal and do:

```
make gettext
```

The resulting pot files are saved in the folder `build/gettext`. They contain the translatable text broken down into segments.

Then you need to generate the po files for each target language. For example, for French you need to do

```
sphinx-intl update -p build/gettetxt/ -l fr
```

The resulting po files are stored in the folder `source/locale/fr/LC_MESSAGES`.

The po files contain pairs of segments of text in the source (msgid) and target language (msgstr).

Before starting the translation, the value of msgstr is empty:

```
#: ../../source/style.rst:24
msgid "The user manual has been generated using ``sphinx`` from files using␣
↪`reStructuredText <http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.
↪html>`_ markup language."
msgstr ""
```

## 5.3 Translation

Thus to translate the text you need to edit the po files and type de translated text beside msgstr:

```
#: ../../source/style.rst:24
msgid "The user manual has been generated using ``sphinx`` from files using␣
↪`reStructuredText <http://www.sphinx-doc.org/en/master/usage/restructuredtext/index.
↪html>`_ markup language."
msgstr "Le guide utilisateur a été créé avec ``sphinx`` depuis fichiers utilisant le␣
↪language markup `reStructuredText <http://www.sphinx-doc.org/en/master/usage/
↪restructuredtext/index.html>`_."
```

**Attention:** Be careful to maintain the reST formatting in the translated version.

You can do the translation by hand by modifying the po files. However, it is more efficient to use a PO editor (e.g., GNOME Translation Editor or PO edit) or computer assisted translation (CAT) software such as OmegaT.

## 5.4 Compilation of the translated documentation

The documentation in French is saved in the folder `build_fr`. To compile the user manual in French as html files go to the folder `pathtorepertorycuspy/sphinx-doc` and type:

```
sphinx-build -b html -aE -D language='fr' -c source/locale/fr source build_fr/html
```

The output html files are saved in the folder `pathtorepertorycuspy/sphinx-doc/build_fr/html`.

To compile the pdf documentation, type the following commands:

```
sphinx-build -b latex -aE -D language='fr' -c source/locale/fr source build_fr/latex
cd build_fr/latex
make
```

The source documentation files are converted to latex and then to pdf. The output latex and pdf files are saved in the folder `pathtorepertorycuspy/sphinx-doc/build_fr/latex`.

## 5.5 Update of the translated documentation

If the project's documentation is updated, it is necessary to create new pot files following the procedure described above. To apply the changes to the po files, do

```
sphinx-intl update -p build/gettext -l fr
```

Then, you only need to translate the modified segments.

CHAPTER

SIX

# MODULES

## 6.1 Module `input_output`

Functions to read and write data.

The functions in this module are used to manage reading and writing of input and output files.

This module contains the following functions:

- *`get_obs_data()`*: Get observation data.
- *`process_sweep_out()`*: Process file `sweep_out.csv`.
- *`write_dict()`*: Write dictionary to file.
- *`write_dummy_pred_file()`*: Write a predictions file with dummy values.
- *`write_ins_file()`*: Write PEST instruction file.
- *`write_pest_files()`*: Writes PEST files.
- *`write_par_data_file()`*: Write parameter data file.
- *`write_tpl_file()`*: Write PEST template parameter file.

`input_output.`**`get_obs_data`**(*obs_file*, *start_date*, *end_date*, *weights=1*, *groups='obs'*, *obsnames='default'*, *delimiter=' '*)

Get observation data

This function is used to read data (observations or predictions) from a file. It is also used to group data and to assign names and weights to data. In PEST parlance "observations" correspond to measurements already made (usually with weight > 0), and "predictions" (or "forecasts") correspond to data we intend to forecast (with weight = 0).

**Parameters**

- **`obs_file`** – path of the observations file (it should have the same format as the model output file). Missing values may be indicated with NA, na or NaN. The first column contains dates in the format "YYYY-mm-dd".
- **`start_date`** – date of start of simulation (in the format "YYYY-mm-dd").
- **`end_date`** – date of end of simulation (in the format "YYYY-mm-dd").
- **`weights`** – str or float. If str, it indicates the path to a file with the same format and column names as obs_file containing weights for each measurement. If float, it indicates a weight to apply to all measurements.

- **groups** – it can be a path, a string or 'default'. If it is a path, group names are read from a file with the same format and column names as obs_file. If 'default', observations are grouped by variable name. If another str, all observations are assigned to the same group.
- **obsnames** – it can be a path, a string or 'default'. If it is a path, observation names are read from a file with the same format and column names as obs_file. If 'default', observations names have the format xxxx_YYYYmmdd (xxxx is the corresponding variable name, and YYYYmmdd is a date string). If another str, the observation names have the format obsnamesNN, where NN is the index of the observation in the resulting dataframe.
- **delimiter** – column delimiter used in the files read by the function.

**Returns** A pandas dataframe containing the row and column indexes of observations in the output file, the observation names and groups, and their weights.

`input_output.`**`process_sweep_out`**`(`*fname_in*, *var_names*, *folder_out*, *ptl_avg=None*`)`

Process file `sweep_out.csv`.

**Parameters**

- **fname_in** – Path of the `sweep_out.csv` file, created when making Monte Carlo simulations.
- **var_names** – List of variable names.
- **folder_out** – Folder where the processed data will be written.
- **ptl_avg** – Dictionary of variable names for which partial averages will be calculated. The keys indicate the variables used to group data into percentiles; the values are lists of variables names for which partial averages will be calculated for each of the percentile groups of the key variable.

**Returns** A series of text files containing the base simulation, the minimum, maximum, and the percentiles 5%, 25%, 50%, 75% and 95% of the simulations in the `sweep_out.csv` file. The names of the output files follow the pattern "mc_uncert_<v>.txt", where *v* is the name of the output variable as used in `sweep_out.csv`. In addition, if *ptl_avg* is specified, another series of files is created, which contains the partial averages of one variable (*vpa*) by percentile group of another variable (*v*). In this case, the output files follow the pattern "mc_pavg_<vpa>_by_<v>.txt".

`input_output.`**`write_dict`**`(`*x_dict*, *path*`)`

Write dictionary to file.

This function writes a Python dictionary to a file. The file is organised in two columns, the first containing keys, and the second one containing values.

**Parameters**

- **x_dict** – a Python dictionary.
- **path** – path of the text file to write the data.

**Returns** A file located at *path* where the dictionary data is written.

`input_output.`**`write_dummy_pred_file`**`(`*fname*, *time_stamps*, *colnames*, *time_colname='date'*, *dummy_value=0*, *sep=' '*`)`

Write predictions file filled with dummy values.

**Parameters**

- **fname** – File name.
- **time_stamps** – List or array of time stamps corresponding to the data in the predictions file.
- **colnames** – Nomes of the columns in the predictions file.

- **time_colname** – Name of the column containing time data (provided through the argument *time_stamps*.
- **dummy_value** – Value used to fill the predictions file (except time data).
- **sep** – Column separator of the output file.

**Returns** A file with the same format as the output file, filled with dummy values.

**Note:** A predictions file filled with dummy values may be useful when it is necessary to obtain simulation results for a certain period and there are no observations available.

`input_output.`**`write_ins_file`**`(`*fname*, *obs_inds*, *pred_inds*, *ncols*, *nl_header=1*, *field_wd=10*, *delimiter=' '*`)`

Write PEST instruction file.

This function is used to write a PEST(++) instruction file, a text file used to specify how PEST should read a model's output file. Detailed instructions on how to format an instructions file can be found in White et al. (2019).

**Parameters**

- **fname** – a file name. It is recommended to use the extension .ins.
- **obs_inds** – list of tuples (x, y, obsname) indicating the row index (x), column index (y) and name (obsname) corresponding to observations in the model output file.
- **pred_inds** – list of tuples (x, y, obsname) indicating the row index (x), column index (y) and name (obsname) corresponding to predictions in the model output file.
- **ncols** – number of columns of the output file.
- **nl_header** – number of lines of the file header.
- **field_wd** – width of the field in characters. It must be at least as long as the observation/prediction name.
- **delimiter** – delimiter character used in the output file.

**Returns** A PEST(++) instruction file (a text file specifying how to process a model output file to read the simulation results corresponding to the observations and predictions of interest).

## References

- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`input_output.`**`write_par_data_file`**`(`*parnme*, *parval1*, *parlbnd*, *parubnd*, *partrans='none'*, *parchglim='relative'*, *pargp=None*, *scale=1.0*, *offset=0.0*, *dercom=1*, *file_path='par_data.csv'*`)`

Write parameter data file.

**Parameters**

- **parnme** – List of names of the parameters.
- **parval1** – List of initial values of the parameters.
- **parlbnd** – List of lower bounds of the parameters.
- **parubnd** – List of upper bounds of the parameters.
- **partrans** – Transformation applied to all the parameters or list of transformations applied to each parameter. The possible values are “none”, “log”, “fixed” or “tied”.

- `parchglim` – Type of parameter change limit (value to apply to all the parameters or list of values). The possible values are: "relative" and "factor".
- `pargp` – List of parameter group names. If None, the parameter group names are the same as the parameter names (one parameter group for each parameter).
- `scale` – Multiplier of list of multipliers of the parameter values.
- `offset` – Offset or list of offsets to aplly to the parameter values.
- `dercom` – Number (or list of numbers) of the command line used to calculate the derivatives of the parameters. It is usually equal to 1.
- `file_path` – Path of the parameter data file where the data will be written.

input_output.**write_pest_files**(*start_date*, *end_date*, *model_command*, *par_file='par.txt'*, *output_file='output.txt'*, *par_data_file='par_data.txt'*, *obs_file=None*, *pred_file=None*, *obs_weights=1*, *obs_groups='default'*, *obs_names='default'*, *pred_groups='default'*, *pred_names='default'*, *tpl_file='par.tpl'*, *ins_file='res_file.ins'*, *pst_file='pest.pst'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *delimiter=' '*)

Write PEST files

The function `write_pest_files()` is used to write the files necessary to run programs of the PEST(++) suite. This includes the control file, instructions file and template parameter file. For more information on the files used by PEST++, please see White et al. (2019).

**Parameters**

- `start_date` – date of start of simulations in str format ('YYYY-mm-dd') or datetime-like format.
- `end_date` – date of end of simulations in str format ('YYYY-mm-dd') or datetime-like format.
- `model_command` – command used to run the model.
- `par_file` – path of the model's parameter file. It is written by the function using the parameter values in *par_data_file*.
- `output_file` – path of the model's output file.
- `par_data_file` – path of a file containing parameter data to configure the parameter section of the PEST control file.
- `obs_file` – path of the observations file. It has the same format as *output_file*.
- `pred_file` – path of the predictions file. It has the same format as *output_file*.
- `obs_weights` – weights assigned to observations. It can be str or float. If str, it indicates the path to a file with the same format and column names as *obs_file* containing weights for each measurement. If float, it indicates a weight to apply to all measurements.
- `obs_groups` – groups to which each observation is assigned. It can be a path, a string or 'default'. If it is a path, group names are read from a file with the same format and column names as *obs_file*. If 'default', observations are grouped by variable name. If another str, all observations are assigned to the same group.
- `obs_names` – observation names. It can be a path, a string or 'default'. If it is a path, observation names are read from a file with the same format and column names as *obs_file*. If 'default', observation names have the format "xxxx_YYYYmmdd" (xxxx is the corresponding variable name, and YYYYmmdd is a date string). If another str, the observation names

have the format "obs_namesNN", where NN is the index of the observation in the resulting dataframe.

- **pred_groups** – groups to which each prediction is assigned. It can be a path, a string or 'default'. If it is a path, group names are read from a file with the same format and column names as *obs_file*. If 'default', predictions are grouped by variable name. If another str, all predictions are assigned to the same group.
- **pred_names** – prediction names. It can be a path, a string or 'default'. If it is a path, prediction names are read from a file with the same format and column names as *pred_file*. If 'default', prediction names have the format "xxxx_YYYYmmdd" (xxxx is the corresponding variable name, and YYYYmmdd is a date string). If another str, the prediction names have the format "pred_namesNN", where NN is the index of the prediction in the resulting dataframe.
- **tpl_file** – path of the template parameter file. It is written by the function using the parameter names in *par_data_file*.
- **ins_file** – path of the pest instruction file. It is written by the function by using the data in *obs_file* and *pred_file*.
- **pst_file** – path of the PEST control file. It is written by the function using the data it has been passed to it.
- **control_data** – a dictionary defining options in the control data section of the PEST control file.
- **svd_data** – a dictionary used to pass options to the SVD section of the PEST control file.
- **reg_data** – a dictionary used to pass options to the regularization section of the PEST control file.
- **pestpp_opts** – a dictionary used to pass PEST++ options to the PEST control file.
- **delimiter** – delimiter character used in the *output_file*, *obs_file* and *pred_file*.

**Returns** A series of PEST files (control file, instructions file and template parameter file) and a parameter file.

## References

- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

input_output.**write_tpl_file**(*fname*, *par_names*)

Write PEST template parameter file.

This function writes a PEST template parameter file, a text file used to specify the format of the parameter file. The parameter file is a text file where the values of the parameters used by the model are written.

**Parameters**

- **fname** – a file name. It is recommended to use the extension .tpl.
- **par_names** – list of parameter names in the order they appear in the parameter file.

**Returns** A PEST(++) template parameter file (a text file specifying the format of the parameter file).

### References

- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

## 6.2 Module `functions`

Useful functions.

The function in this module is used to launch the PEST++ executable.

This module contains the following function:

- *`launch_pestpp()`*: launch PEST++ executable

`functions.`**`launch_pestpp`**`(`*pst_file*, *pestpp_folder*, *pestpp_cmd='pestpp-glm'*, *parallel=False*`)`

Launch PEST++ executable.

This function launches the requested PEST++ executable, either making calculations on line or in parallel. The PEST++ executables and uses are:

- PESTPP-GLM: highly parameterized inversion and global optimization (i.e., parameter optimization, calibration).
- PESTPP-SEN: global sensitivity analysis.
- PESTPP-OPT: constrained optimization under uncertainty (White et al. 2018).
- PESTPP-IES: interative ensemble smoothing (White 2018).
- PESTPP-SWP: Monte Carlo simulations.

For more details on the PEST++ executables, please see White et al. (2019).

**Parameters**

- **`pst_file`** – path of pest control file (.pst).
- **`pestpp_folder`** – folder containing the PEST++ executables.
- **`pestpp_cmd`** – PEST++ executable. The PEST++ executables are: “pestpp-glm”, “pestpp-sen”, “pestpp-opt”, “pestpp-ies” and “pestpp-swp”.
- **`parallel`** – parallelize calculations.

**Returns** The output of the PEST++ command is shown on screen. In addition, the *pestpp_cmd* output files are written in the folder containing the *pst_file*.

### References

- White, J.T. (2018) A model-independent iterative ensemble smoother for efficient history matchingh and uncertainty quantification in very high dimensions. *Environmental Modelling and Software* 109, 191-201.
- White, J.T.; Fienen, M.N.; Barlow, P.M.; Welter, D.E. (2018) A tool for efficient, model-independent management optimization under uncertainty. *Environmental Modelling and Software* 100, 2013-221.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

## 6.3 Module `analyses`

Functions to make analyses based on PEST++.

The functions in this module carry the different type of analyses implemented in the package.

This module contains the following functions:

- *`calibration()`*: calibrate model.
- *`gsa()`*: Global Sensitivity Analysis.
- *`ies()`*: Iterative Ensemble Smoother.
- *`linear_uncertainty()`*: predictive uncertainty.
- *`monte_carlo()`*: Monte Carlo simulations.

`analyses.`**`calibration`**`(`*method='glm'*, *reg=False*, *pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *pestpp_folder='..'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=False*`)`

Calibrate model (GLM or DE methods).

**Parameters**

- **`method`** – method used to calibrate the model. It is 'glm' for the Gauss-Levenberg-Marquardt algorithm, 'de' for differential evolution.
- **`reg`** – use Tikhonov regularisation.
- **`pst_file0`** – path of original pest file.
- **`pst_file1`** – path of modified pest file.
- **`pestpp_folder`** – folder containing the PEST++ executables.
- **`control_data`** – a dictionary defining options in the control data section of the pest file.
- **`svd_data`** – a dictionary used to pass options to the svd section of the pest file.
- **`reg_data`** – a dictionary used to pass options to the regularization section of the pest file.
- **`pestpp_opts`** – a dictionary used to pass options to configure the DE method.
- **`parallel`** – parallelize model runs.

**Returns** A pest instance where the parameter values correspond to the calibrated values. A pest file (*pst_file1*) and associated files containing the results of the calibration process are also created.

**Note:** This function calls the PEST++ executable *pestpp-glm*.

### References

- Doherty, J. (2010) *Methodologies and Software for PEST-Based Model Predictive Uncertainty Analysis.* Watermark Numerical Computing. 157 p.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`analyses.`**`gsa`**`(`*method='morris'*, *pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *pestpp_folder='..'*, *parallel=True*`)`

Carry global sensitivity analysis (Morris or Sobol methods).

**Parameters**

- `method` – 'morris' for Morris's method (one parameter at a time) or 'sobol' for Sobol's method.
- `pst_file0` – path of original pest file.
- `pst_file1` – path of modified pest file.
- `control_data` – a dictionary defining options in the control data section of the pest file.
- `svd_data` – a dictionary used to pass options to the svd section of the pest file.
- `reg_data` – a dictionary used to pass options to the regularization section of the pest file.
- `pestpp_opts` – a dictionary used to pass options to configure the GSA method.
- `pestpp_folder` – folder containing the PEST++ executables.
- `parallel` – parallelize model runs.

**Returns** A modified pst file and associated files containing results.

**Note:** This function calls the PEST++ executable *pestpp-sen*.

## References

- Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F.; Cariboni, J.; Gatelli, D.; Saisana, M.; Tarantola, S. (2008) *Global Sensitivity Analysis. The Primer.* John Wiley & Sons, Ltd. 292 p.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`analyses.ies`(*pst_file0*, *pst_file1*, *pestpp_folder*, *n_reals=50*, *parcov=None*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=True*)

Iterative Ensemble Smoother.

**Parameters**

- `pst_file0` – path of original pest file.
- `pst_file1` – path of modified pest file.
- `pestpp_folder` – folder containing the PEST++ executables.
- `n_reals` – number of realizations.
- `parcov` – name of prior parameter covariance file. If None, parameter covariance is estimated from parameter bounds.
- `control_data` – a dictionary defining options in the control data section of the pest file.
- `svd_data` – a dictionary used to pass options to the svd section of the pest file.
- `reg_data` – a dictionary used to pass options to the regularization section of the pest file.
- `pestpp_opts` – a dictionary containing additional options to pass to *pestpp-ies*.
- `parallel` – parallelize calculations.

**Returns** A modified pest control file and the associated output files.

**Note:** This function calls the PEST++ executable *pestpp-ies*.

### References

- White, J. T. (2018) A model-independent iterative ensemble smoother for efficient history-matching and uncertainty quantification in very high dimensions. *Environmental Modelling & Software*, 109, 191-201.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

`analyses.`**`linear_uncertainty`**(*analysis*, *pst_file0*, *pst_file1*, *pestpp_folder*, *predictions=None*)

Carry linear uncertainty calculations.

**Parameters**

- **`analysis`** – Type of predictive uncertainty analysis. It can take the values 'schur', for Schur's complement analysis for conditional uncertainty propagation; 'err_var', for error variance analysis; or 'prior', prior uncertainty estimation.
- **`pst_file0`** – Path of the original pest file.
- **`pst_file1`** – Path of the modified pest file.
- **`pestpp_folder`** – folder containing the PEST++ executables.
- **`predictions`** – list of predictions names. If None, predictions are read from the *pst_file0* as the observations with null weight.

**Returns** A `LinearAnalysis` object (`analysis='prior'`), a `Schur` object (`analysis='schur'`) or an `ErrVar` object (`analysis='err_var'`). A modified PEST control file (*pst_file1*) and a Jacobian matrix file are also created file in the process.

**Note:** It is supposed that the model has already been calibrated. Thus, the original PEST control file *pst_file0* should be in the same folder as the files created during the calibration process.

### References

- Doherty, J. (2010) *Methodologies and Software for PEST-Based Model Predictive Uncertainty Analysis.* Watermark Numerical Computing. 157 p.
- White, J.T.; Fienen, M.N.; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling & Software*, 85, 217-228.

`analyses.`**`monte_carlo`**(*pst_file0='pest.pst'*, *pst_file1='pest.pst'*, *dist_type='post'*, *distribution='gaussian'*, *n_samples=100*, *how_dict=None*, *csv_in='sweep_in.csv'*, *pestpp_folder='..'*, *add_base=False*, *control_data=None*, *svd_data=None*, *reg_data=None*, *pestpp_opts=None*, *parallel=True*, *process_swp_out=False*)

Monte Carlo simulations.

**Parameters**

- **`pst_file0`** – path of original pest file.
- **`pst_file1`** – path of modified pest file.
- **`dist_type`** – type of distribution; 'prior' for prior distribution or 'post' for posterior distribution.
- **`distribution`** – 'uniform', 'triangular', 'gaussian', 'mixed'. Ignored if `type = post` (gaussian distribution used).
- **`n_samples`** – number of samples to draw.

- **`how_dict`** – dictionary of parameter names (keys) and distributions (values). Only used if `distribution = mixed`.
- **`csv_in`** – name of the file to save the parameter samples.
- **`pestpp_folder`** – folder containing the PEST++ executables.
- **`add_base`** – add the *pst_file0* parameter values as a realization.
- **`control_data`** – a dictionary defining options in the control data section of the pest file.
- **`svd_data`** – a dictionary used to pass options to the svd section of the pest file.
- **`reg_data`** – a dictionary used to pass options to the regularization section of the pest file.
- **`pestpp_opts`** – a dictionary used to pass options to configure the *pestpp-swp* options.
- **`parallel`** – parallelize calculations.
- **`process_swp_out`** – option to process the sweep_out.csv results file. Observation group names should correspond to the names of the variables to which the observations belong.

**Returns** A modified pest control file and the associated output files.

**Note:** This function calls the PEST++ executable *pestpp-swp*.

## References

- White, J.T.; Fienen, M.N.; Doherty, J.E. (2016) A python framework for environmental model uncertainty analysis. *Environmental Modelling & Software*, 85, 217-228.
- White, J.; Welter, D.; Doherty, J. (2019) *PEST++ Version 4.2.16*. PEST++ Development Team. 175 p.

CHAPTER

SEVEN

# USAGE

The cuspy package can be applied to a large class of models, on the condition that:

- the model can be run using the command line (e.g., through an executable that is available in the path or a call to a script).
- the communication between the model and the user (output data, parameter values) is made by means of text files (or the script calling the model reads data in binary form and creates output in text form).

In addition, the package has been specially designed to work with dynamic models the output of which is in several columns, the first of which is date. Working with models with other type of output is possible, but it requires more implication of the user in the preparation of the PEST instruction files.

## 7.1 Command line application

## 7.2 Python module

To use cuspy as a Python module you can simply import it and use the functions within:

```python
import cuspy
```

You can include the previous commands in a Python script (see the scripts in the folder tests). To run a python script from the command line, type:

```
python path_to_script
```

## 7.3 Analysis process

### 7.3.1 Preparation of the model

Before starting the analyses, you need to prepare all the input files and scripts required by the model.

In particular, it must be possible to execute the model using a command line instruction of the type:

```
run_model -x arg_x -y arg_y
```

Alternatively, if the model has binary output or several output files, you may need to create a script to process the model output so that it conforms with the recommended format (text file separated by spaces with the date in the first column). For example:

```
python script.py
```

Plus, the model must read the values of its parameters from a text file organised in two columns separated by spaces containing parameter names and values. For example:

```
A 6.2
B 1.007
C -0.007
D 0.51
E 0.24
ALPHA 0.07
BETA 0.13
mat -0.4
at_factor 1.0
sw_factor 1.0
```

## 7.3.2 Creation of PEST files

The first stage in the use of the `cuspy` package is the creation of the PEST files. You can do this manually, or you can use the function `write_pest_files()`. The following is an example of the instruction to create the PEST files for the OKP model.

```
cuspy.write_pest_files(start_date=start_date, end_date=end_date,
                       par_file='par.txt', output_file='output.txt',
                       par_data_file='par_data.csv',
                       obs_file='obs.txt', pred_file='pred.txt',
                       model_command='run_okp',
                       control_data={'noptmax': 0, 'numlam': 10},
                       svd_data={'maxsing': len(par_names)},
                       pestpp_opts={'parcov': uncertainty_file},
                       tpl_file='par.tpl', ins_file='res_file.ins',
                       pst_file='test0.pst')
```

In the previous example, we can notice the following facts:

- `start_date` and `end_date` correspond to the initial and final dates of the simulation.
- `par_file` and `output_file` are the names of the model's parameter and output files.
- `par_data_file`, `obs_file` and `pred_file` are space-separated text files used to pass parameter and observation data to the function. These must be prepared by the user.
- `model_command` is the command line instruction used to run the model. In this case, the model OKP is run using the instruction `run_okp` of the `okplm` package.
- `control_data`, `svd_data` and `pestpp_opts` are dictionaries used to set the values of variables in the control data, singular value decomposition and PEST++ control data sections of the pest control file.
- `tpl_file`, `ins_file` and `pst_file` are the names of PEST files created by the function. The `tpl_file` is a template of the parameter file `par_file` used by PEST++ to modify it. The `ins_file` is a file indicating PEST++ which values in the `output_file` correspond to observations and predictions. The `pst_file` controls the execution of the PEST++ executables.

We describe next some of the files used by `write_pest_files()`. For more information on other arguments (including some not used above), please see the function's description in the Modules section.

### The PEST control file (`pst_file`)

The PEST control file contains all the information necessary to control the execution of the PEST++ executables. It is automatically written by the function `write_pest_files()` and it is divided in the following sections:

- **control data** (mandatory)
- automatic user intervention (optional)
- **singular value decomposition** (optional)
- lsqr (optional)
- sensitivity reuse (optional)
- svd assist (optional)
- **parameter groups** (mandatory)
- **parameter data** (mandatory)
- **observation groups** (mandatory)
- **observation data** (mandatory)
- derivatives command line (optional)
- **model command line** (mandatory)
- **model input** (mandatory)
- **model output** (mandatory)
- prior information (optional)
- predictive analysis (optional)
- **regularization** (optional)
- pareto (optional)
- **PEST++ variables** (optional)[1]

At the moment `cuspy` functions can be used to configure the sections in bold, which include all mandatory sections and some optional ones. If necessary, the other sections can be configured manually. For details on the file structure and variables, please see PEST++ user manual.

### The parameter data file (`par_data_file`)

In the example above, `par_data_file` is a text file separated by spaces containing the data of the parameter section of the PEST control file. This file must be provided by the users, based on their previous knowledge of the optimum parameter values and acceptable ranges of variation.

An example is shown below.

```
parnme partrans parchglim parval1 parlbnd parubnd pargp scale offset dercom
a none relative 6.2 4.7 7.7 a 1.0 0.0 1
b none relative 1.007 0.847 1.167 b 1.0 0.0 1
c none relative -0.0069 -0.015 0.001 c 1.0 0.0 1
d none relative 0.51 0.0 1.0 d 1.0 0.0 1
e none relative 0.24 0.0 1.0 e 1.0 0.0 1
```

(continues on next page)

[1] Declarations of PEST++ variables (the string "++" followed by the variable name and value) can be located anywhere in the PEST control file, but they are placed at the end of the file by `write_pest_files()`.

(continued from previous page)

```
alpha none relative 0.07 0.0 0.23 alpha 1.0 0.0 1
beta none relative 0.13 0.0 1.0 beta 1.0 0.0 1
mat none relative 2.4 1.4 3.4 mat 1.0 0.0 1
at_factor fixed relative 1.0 0.9 1.1 at_factor 1.0 0.0 1
sw_factor fixed relative 1.0 0.9 1.1 sw_factor 1.0 0.0 1
```

The first line of the file contains the column names, described below:

- `parnme`: parameter name (up to 200 characters, case insensitive).
- `partrans`: parameter transformation. It takes one of four possible values:
  - `'none'`: no transformation applied to the parameter.
  - `'log'`: the parameter is log-transformed.
  - `'fixed'`: constant or non-adjustable parameter.
  - `'tied'`: the parameter is tied to another parameter (not implemented in `cuspy`).
- `parchglim`: parameter change limit. It is used by the PEST++ executables *pestpp-glm* and (optionally) *pestpp-ies* to limit the change a parameter can suffer at each iteration of the optimization process. It can take the values:
  - `'relative'`: a relative limit is used, i.e., the value of the parameter $b$ must fulfill the condition $|b - b_0|/|b_0| \leq r$, where $b_0$ is the parameter value at the start of the iteration and $r$ is the value of the PEST control variable `relparmax` (=10 by default).
  - `'factor'`: a factor limit is used,, i.e., the value of the parameter $b$ must be within the limits $|b_0/f| \leq b \leq |fb_0|$, where $b_0$ is the parameter value at the start of the iteration and $f$ is the value of the PEST control variable `facparmax` (=10 by default).
- `parval1`: initial parameter value. It should be the best estimate of the parameter value before calibration based on expert knowledge.
- `parlbnd` and `parubnd`: lower and upper bounds of the parameter value. If an uncertainty file is provided[2], they represent the minimum and maximum values that the parameter can take. If an uncertainty file is not provided, they are understood to represent the 95% confidence interval (a width of 4 standard deviations)[3] of the parameter value.
- `pargp`: parameter group name to which the parameter has been assigned.
- `scale` and `offset`: multiplier (1.0 by default) and offset (0.0 by default) to apply to a parameter before being written to a model input file. It can be used to modify the parameter range of a parameter to a more convenient one (e.g., if the parameters takes negative values and a log transformation is envisaged).
- `dercom`: integer indicating the line of the "model command line" section in the PEST control file that is used to calculate derivatives for a given parameter. Usually there is only one command (`dercom=1`).

[2] An uncertainty file can be provided with the PEST++ control variable `parcov`.
[3] The width of the interval can be modified with the PEST++ control variable `par_sigma_range`.

### The observations

The files `obs.txt` and `pred.txt` contain observations or measurements. They have the same format as the model output file, but include only the available observations. It is necessary to provide at least one of these files.

One way of looking at the data in `obs.txt` is as the data used in the calibration or history matching process. The observations in `obs.txt` receive a positive weight (1 by default) and if there is missing data (coded as NA, NaN or nan), the corresponding rows will be ignored. The observation weights can be modified with the argument `obs_weights`. This is useful when some measurements are more uncertain than others.

Instead, the observations in the `pred.txt` file or predictions, may be seen as validation data or dates for which a forecast is desired. They receive a null weight and their value is not used in calculations.

Observations can be assigned to different groups using the arguments `obs_groups` and `pred_groups`. By assigning observations to different groups, we obtain a multiple-component objective function. For example, in the OKP model, that estimates epilimnion and hypolimnion temperatures, measurements for each of these two compartments may be placed in different groups.

For more complex models this is still more useful. For example, in the GLM model, the user could have temperature measurements taken within the lake, water temperatures at the outlet, satellite temperatures at the surface, water level measurements, salinity measured within the lake and salinity at the outlet. All these different types of measurement could be assigned to different groups. Or, if the study is interested in only one aspect, use only one of those sets of data (and only one group) to calibrate the model.

Still, Doherty & Welter (2010) recommend using multiple-component objective functions as a way of reducing the influence of model structural error on the calibration.

### The uncertainty file

Information on prior parameter uncertainty can be provided to PEST++ through an uncertainty file using the PEST++ control variable `parcov`. This file can take several forms:

- A parameter uncertainty file (.unc). It contains standard deviations for individual parameters that have no statistical correlation with other parameters and/or indicates the name of covariance matrix files for other parameters.
- A single prior covariance matrix file (.cov). It contains a covariance matrix of all the model's parameters. The file is written according to the PEST matrix file specifications.
- A binary covariance matrix file (.jco or .jcb). It is a file used by PEST++ to write a Jacobian matrix.

For more information on the specification of these types of file see the PEST++ user guide.

If an uncertainty file is not provided, parameter uncertainty is estimated from the parameter bounds by assuming the interval covers 4 standard deviations.

## 7.3.3 Carry the analyses

This package allows you to carry several types of analysis:

- Model calibration (aka parameter estimation, history matching). There are four possible options:
  - Gauss-Levenberg-Marquardt (GLM) algorithm, using the function `calibration()`. It is a gradient algorithm for overdetermined cases.
  - GLM algorithm with Tikhonov regularisation, using the function `calibration()`. For under-determined cases.
  - Differential Evolution (DE), using the function `calibration()`. It is a global algorithm with longer time of computation than the previous two options.

  - Iterative ensemble smoother, using the function `ies()`. It is also a global algorithm, especially useful for nonlinear models with a large number of parameters, that obtains several sets of parameter values that can be considered to be calibrated.

- Prior (before calibration) and posterior (after calibration) uncertainty analysis. There are the following options:
  - Linear uncertainty analysis, using the function `linear_uncertainty()`. Appropriate for linear or quasi-linear models. It may be applicable to a nonlinear model if the behaviour of the model in the parameter space is approximately linear ($R^2 > 0.7$). It requires a low computation time, once the model is calibrated. The following options are available:
    - Prior linear uncertainty analysis.
    - Schur's complement analysis for conditional uncertainty propagation.
    - Error variance analysis. It also includes identifiability estimation.
  - Nonlinear uncertainty analysis. Preferable when the model is not linear. However, calculation times are longer.
    - Monte Carlo simulations, using the function `monte_carlo()`. It usually requires a large number of simulations (>1000).
    - Iterative ensemble smoother, using the function `ies()`. It requires a much lower number of simulations than Monte Carlo methods.
- Sensitivity analysis. There are two options:
  - Local sensitivity analysis. Low computation time. It is calculated during calibration using the function `calibration()`.
  - Global sensitivity analysis (GSA). Applied using the function `gsa()`. Two implemented methods:
    - Morris's method (one parameter at a time). Low computation time.
    - Sobol's method (all parameters vary at the same time). High computation time.

In all cases calculations can be parallelized using the argument `parallel=True`.

The following table shows the computational cost of several `cuspy` functions in terms of number of model runs and run time. Run time calculations are based on the cases presented in the `tests` folder and were made using an Ubuntu virtual machine with 4 processors of 3067 MHz. These tests consist in the application of different `cuspy` functions to the OKP water temperature model with 8 adjustable parameters.

Table 7.1: Computational cost of different types of analyses

| **Function** | **Details** | **Parallelized** | **N. runs** | **Run time (s)** |
|---|---|---|---|---|
| `calibration()` | `method='glm'`, `noptmax=10` | Yes | 95 | 101.3 |
| `calibration()` | `method='glm'`, `noptmax=10` | No | 95 | 192.1 |
| `calibration()` | `method='glm'`, `noptmax=10`, `reg=True` | Yes | 145 | 146.6 |
| `calibration()` | `method='glm'`, `noptmax=10`, `reg=True` | No | 145 | 292.6 |
| `calibration()` | `method='de'`, `de_max_gen=20` | Yes | 801 | 707.3 |
| `calibration()` | `method='de'`, `de_max_gen=20` | No | 801 | 1694.7 |
| `gsa()` | `method='morris'` | Yes | 36 | 37.0 |
| `gsa()` | `method='morris'` | No | 36 | 72.8 |
| `gsa()` | `method='sobol'`, `gsa_sobol_samples=50` | Yes | 500 | 413.4 |
| `gsa()` | `method='sobol'`, `gsa_sobol_samples=50` | No | 500 | 1009.1 |
| `monte_carlo()` | `n_samples=50`[4] | Yes | 50 | 50.9 |
| `monte_carlo()` | `n_samples=50`[7] | No | 50 | 103.5 |
| `ies()` | `noptmax=10`, `n_reals=50` | Yes | 788 | 672.2 |
| `ies()` | `noptmax=10`, `n_reals=50` | No | 788 | 1590.0 |

### 7.3.4 Calibration with GLM or DE

To calibrate a model you can use the function `calibration()`. For example:

```
pst = cuspy.calibration(method='glm', reg=False,
                        pst_file0='test0.pst',
                        pst_file1='test1.pst',
                        control_data={'noptmax': 10})
```

The pest instance returned by the function can be used to access the results of the calibration. To obtain the calibrated parameter values, type:

```
pst.write_par_summary_table()
```

To obtain the simulated values and residuals, type:

```
pst.res
```

And to obtain a summary of these, type:

```
pst.write_obs_summary_table()
```

You may plot the results using:

```
pst.plot()
```

In addition, a certain number of useful files is written by PEST++. Please, see the PEST++ user guide to obtain information on these files.

### 7.3.5 Global sensitivity analysis

Two methods can be used to carry a global sensitivity analysis: Morris's method and Sobol's method. You need to use the function `gsa()`, by choosing the desired method ('sobol' or 'morris') and configuring the method using the *pestpp_opts* argument. For example:

```
cuspy.gsa(method='sobol', pst_file0='pest.pst', pst_file1='pest3.pst',
          pestpp_folder=pestpp_folder,
          pestpp_opts={'gsa_sobol_samples': 50, 'gsa_sobol_par_dist': 'unif'})
```

### 7.3.6 Linear uncertainty analysis

Once the function is calibrated, you can make a linear uncertainty analysis using the function `linear_uncertainty()`. For example:

```
la = cuspy.linear_uncertainty(analysis='prior', pst_file0='test6.pst',
                              pst_file1='test6b.pst',
                              pestpp_folder=pestpp_folder)
```

[4] The number of simulations (`n_reals`) used in Monte Carlo experiences is usually of the order of thousands and greater. Thus calculation time is usually much greater than shown here.

### Prior uncertainty

The `LinearAnalysis` object returned by the function can be used to access the results of the uncertainty analysis. To obtain the prior uncertainty of predictions, type:

```
la.prior_forecast
```

### Posterior uncertainty

To carry a Schur's complement analysis, you need to set `analysis='schur'`. For example:

```
la_sc = cuspy.linear_uncertainty(analysis='schur', pst_file0='test6.pst',
                                 pst_file1='test6b.pst',
                                 pestpp_folder=pestpp_folder)
```

With Schur's complement analysis you can obtain the posterior uncertainty typing:

```
la_sc.posterior_forecast
```

To obtain prior and posterior uncertainties for the predictions of interest and the reduction in uncertainty due to calibration ('schur'), type:

```
forecast_sum = la_sc.get_forecast_summary()
```

Similarly, you can obtain information on the reduction of parameter uncertainty through calibration by typing:

```
parameter_sum = la_sc.get_parameter_summary()
```

### Data worth

It is possible to analyse data worth with the `Schur` object returned by the function `linear_uncertainty()`. The following instructions return the variance of the predictions after adding a new observation or group of observations:

```
la_sc.get_added_obs_importance()
la_sc.get_added_obs_group_importance()
```

A similar analysis can be made by removing observations or groups of observations with the methods `get_removed_obs_importance()` and `get_removed_obs_group_importance()`.

To obtain the contribution of each parameter to prediction uncertainty, type:

```
la_sc.get_par_contribution()
```

#### Predictive error

To analyse predictive error with an error variance analysis you need to set `analysis='err_var'`. For example:

```
la_ev = cuspy.linear_uncertainty(analysis='err_var', pst_file0='test6.pst',
                                 pst_file1='test6b.pst',
                                 pestpp_folder=pestpp_folder)
```

The `ErrVar` object returned by the function `linear_uncertainty()` allows to access the results of the error variance analysis:

```
la_ev.get_errvar_dataframe()
```

#### Identifiability

With the `ErrVar` object, it is also possible to access the results of the identifiability analysis with:

```
la_ev.get_identifiability_dataframe()
```

### 7.3.7 Monte Carlo simulations

To carry Monte Carlo simulations with the `cuspy` package you can use the function `monte_carlo()`. For example:

```
cuspy.monte_carlo(pst_file0='test1.pst', pst_file1='test4.pst',
                  dist_type='post', distribution='uniform',
                  n_samples=5000, pestpp_folder=pestpp_folder,
                  csv_in='sweep_in.csv', parallel=parallel)
```

The sets of parameter values drawn from the specified distribution are saved in the *csv_in* file (`sweep_in.csv` by default). The simulation results are also saved to a csv file, named `sweep_out.csv` by default. The name of the output csv file can be modified using the *pestpp_opts* argument:

```
cuspy.monte_carlo(pst_file0='test1.pst', pst_file1='test4.pst',
                  dist_type='post', distribution='uniform',
                  n_samples=5000, pestpp_folder=pestpp_folder,
                  csv_in='sweep_in.csv', parallel=parallel,
                  pestpp_opts={'sweep_output_csv_file': 'sweep_out.csv'})
```

Monte Carlo simulations can have several uses, including calibration, global sensitivity analyses and uncertainty analyses, depending on how the simulation results are analysed. No particular type of analysis is implemented in this package, but they can be easily implemented based on the treatment of the input and output csv files (*csv_in* and *sweep_output_csv_file*).

### 7.3.8 Iterative Ensemble Smoothing

To apply the IES method, you need to use the function `ies()`. For example:

```
cuspy.ies(pst_file0='test0.pst', pst_file1='test5.pst',
          pestpp_folder=pestpp_folder, n_reals=50,
          control_data={'noptmax': 10})
```

The implementation of the IES method uses the GLM algorithm (called with *pestpp-glm*), which can be configured using the *pst_file0* and *control_data* arguments. The IES method can be configured with the *pestpp_opts* argument.

The output of the `ies()` function can be found in a series of csv files. The optimized parameter value sets can be found in the file `case.N.par.csv`, where "case" is the name of the pest file *pst_file1* and "N" is the number of the latest iteration. The simulation results for each parameter set can be found in the file `case.N.obs.csv`. The objective function results during all iterations can be found in the files `case.phi.actual.csv` (objective function calculated from differences between forecasts and observations), `case.phi.meas.csv` (objective function calculated from differences between forecasts and noisy observations), `case.phi.group.csv` (objective function calculated by parameter and observation groups) and `case.phi.regul.csv` (regularization objective functions).

## 7.4 References

- Doherty, J.; Welter, D. (2010) A short exploration of structural noise. *Water Resources Research*, 46, W05525.

CHAPTER

EIGHT

# EXAMPLE DATA TO TEST THE PACKAGE CUSPY

You can test the application with example data provided in the folder `example_data` and the OKP model implemented in the `okplm` package. Several scripts that use these data and model are also available in the folder `tests`.

## 8.1 The data

The lake data in the `lake.txt` file corresponds to the Lake Allos (lake code = ALL04).

The meteorological data (`meteo.txt`) is synthetic data based on meteorological data. Air temperature has been created using a seasonal component and an ARMA model, while solar radiation data corresponds to the seasonal component only.

The observations in the files `obs.txt` and `pred.txt` are fictitious and their only aim is to verify the package is working correctly and to help learn its usage.

The folder also contains three files containing observation names (`obs_names.txt`), observation group names (`obs_groups.txt`) and observation weights (`obs_weights.txt`).

## 8.2 The scripts

The folder `tests` contains several scripts that can be used to test the package functionalities. Each script illustrates how to make different types of analysis.

The included scripts are:

- `test_0_input.py`: it illustrates input file management.
- `test_1_overdet.py`: it provides examples of calibration in the overdetermined case, with GLM or DE algorithms.
- `test_2_calib_reg.py`: it provides an example of calibration in the under-determined case, using Tikhonov regularisation.
- `test_3_sensitivity.py`: it exemplifies the realisation of a global sensitivity analysis.
- `test_4_monte_carlo.py`: it shows how to carry Monte Carlo simulations.
- `test_5_ies.py`: it shows an application of Iterative Ensemble Smoothing.
- `test_6_uncertainty.py`: it shows an example of calibration and uncertainty analysis.

# PYTHON MODULE INDEX

# INDEX