Technische Universität München

Fakultät für Informatik
Lehrstuhl für Wissenschaftliches Rechnen

Efficient non-intrusive uncertainty quantification for large-scale simulation scenarios

Florian Künzner

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des Akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Helmut Seidl
Prüfer der Dissertation:	1. Prof. Dr. Hans-Joachim Bungartz
	2. Prof. Dr. Ernst Rank

Die Dissertation wurde am 15.09.2020 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 14.12.2020 angenommen.

Uncertainty is everywhere!

Abstract

In Computational Science & Engineering, the model parameters influence the model characteristics and therefore the results of the simulation. Often, these parameters are not exactly known or uncertain, and one computer simulation with specific model parameter values provides only very few information about the general behaviour of such a modelled process. To deal with that type of uncertain parameters, the wide field of uncertainty quantification (UQ) comes into play: Several different methods exist with their own advantages and disadvantages. The generalised polynomial chaos expansion (gPCE) is a prominent and efficient UQ method that is intensively used in this thesis. Typically many forward simulations are necessary to quantify the uncertainty of a process. Compared to a single simulation run with specific, deterministic parameter values, this increases the computational effort significantly and requires efficient solutions.

In the last years, several software frameworks for uncertainty quantification have been developed. All have their specific focus and usually concentrate on providing the latest UQ methods. But the users of the frameworks still need the possibility to rapidly prototype UQ scenarios, support for parameter studies, scaling options from development to production environments, knowledge on the total runtime of the UQ analysis, and automatic and efficient utilisation of the computing systems when quantifying the uncertainty.

This thesis investigates how the uncertainty of a model can be quantified efficiently with the help of high-performance computing. Selected methods are applied in the context of pedestrian dynamics simulation scenarios, e.g. the evacuation of a building, and the utilisation of a campus with several buildings. This includes the choice of a suitable UQ method and its parameters, the prototyping and development of the simulation on a desktop PC up to the execution of the simulation on a high-performance computing system, and the computation and visualisation of the results. With the combination of different existing methods and techniques, as well as new ideas to reduce the simulation time developed in this thesis, it is now possible to quantify the uncertainty of a computer simulation even more efficiently: Speed-ups by a factor of two can be achieved with the developed optimised scheduling strategies using the novel runtime prediction mechanism. The ideas are not limited to the presented software and applications, but can directly be extended to other fields and implementations.

Kurzfassung

Im Wissenschaftlichen Rechnen beeinflussen die Parameter eines Modells dessen Charakteristik und daher auch die Simulationsergebnisse. Häufig sind diese Parameter nicht genau bekannt oder unsicher und eine Computersimulation mit bestimmten Werten der Modellparameter liefert nur sehr wenige Informationen über das allgemeine Verhalten eines solchen modellierten Prozesses. Um mit dieser Art von unsicheren Parametern umzugehen, kommt das weite Feld der Uncertainty Quantification (UQ) ins Spiel: Es gibt mehrere verschiedene Methoden mit ihren eigenen Vor- und Nachteilen. Die "Generalised Polynomial Chaos Expansion" ist eine prominente und effiziente UQ-Methode die in dieser Arbeit intensiv genutzt wird. Typischerweise sind viele Vorwärtssimulationen notwendig um die Unsicherheit eines Prozesses zu quantifizieren. Im Vergleich zu einem einzelnen Simulationslauf mit spezifischen, deterministischen Parameterwerten erhöht dies den Rechenaufwand erheblich und erfordert effiziente Lösungen.

In den letzten Jahren wurden mehrere Software-Frameworks für Uncertainty Quantification entwickelt. Alle haben ihren spezifischen Fokus und konzentrieren sich in der Regel auf die Bereitstellung der neuesten UQ-Methoden. Die Anwender der Frameworks benötigen jedoch nach wie vor die Möglichkeit, schnell Prototypen von UQ-Szenarien zu erstellen, Unterstützung für Parameterstudien, Skalierungsmöglichkeiten von der Entwicklungs- bis zur Produktionsumgebung, Kenntnisse über die Gesamtlaufzeit der UQ-Analyse und die automatische und effiziente Nutzung der Rechensysteme bei der Quantifizierung der Unsicherheit.

In dieser Arbeit wird untersucht, wie die Unsicherheit eines Modells mit Methoden des Hochleistungsrechnens effizient quantifiziert werden kann. Ausgewählte Methoden werden im Rahmen von Simulationsszenarien im Bereich der Fußgängerdynamik angewendet, z.B. bei der Evakuierung eines Gebäudes und der Auslastung eines Campus mit mehreren Gebäuden. Dies umfasst die Auswahl einer geeigneten UQ-Methode und seiner Parameter, die prototypische Entwicklung der Simulation auf einem Desktop-PC bis hin zur Ausführung der Simulation auf einem Hochleistungsrechner sowie die Berechnung und Visualisierung der Ergebnisse. Mit der in dieser Arbeit entwickelten Kombination verschiedener bestehender Methoden und Techniken sowie neuen Ideen zur Reduzierung der Simulationszeit ist es nun möglich, die Unsicherheit einer Computersimulation noch effizienter zu quantifizieren: Mit den entwickelten optimierten Scheduling-Strategien können mit dem neuartigen Laufzeitvorhersage-Mechanismus Beschleunigungen um den Faktor zwei erreicht werden. Die Ideen sind nicht auf die vorgestellte Software und Anwendungen beschränkt, sondern können direkt auf andere Bereiche und Implementierungen übertragen werden.

Acknowledgement

My first thank goes to Prof. Dr Hans-Joachim Bungartz for giving me the opportunity to join the chair as an external researcher. It was very nice to be part of the chair and the excellent people. I enjoyed having the freedom to decide the research topics and directions by myself. I would also like to highlight the fast response and helpful feedback whenever I needed something during the last years. Thank you, Hans, for supervising me.

I do also thank the examination committee: the chairman, Prof. Dr Martin Schulz, and my second reviewer, Prof. Dr Ernst Rank.

A huge thank goes to Dr Tobias Neckel. You helped me on my first scientific computing algorithm steps and to get into the broad field of uncertainty quantification. Your excellent feedback and the valuable discussions helped me so much. It was a pleasure to work with you. During the years, you are now also a friend, and I enjoyed the private chats with you. Thank you, Tobi, for your help.

I would like to thank my PhD candidate collogues at the chair of scientific computing at the technical university of Munich for the fruitful conversations on UQ, mathematics and the (sometimes) hard life as a computer scientist. I want to highlight namely Ionut-Gabriel Farcas, Moritz August, Friedrich Menhorn, Michael Obersteiner, and Ivana Jovanovic, because of the nice discussions, the collaborative work, and the common interests.

Thank you, Prof. Dr Gerta Köster and your team for your interest in applying uncertainty quantification on pedestrian dynamics: Especially Dr Isabelly von Sivers for the cooperation with the pedestrian dynamics scenario 1 and 2, and Dr Felix Dietrich (which now joined the SCCS chair) for the collaboration on the COSM in the pedestrian dynamics scenario 3.

Thank you, Jonathan Feinberg, for developing and maintaining Chaospy. It is fun to work with it and to collaborate with you.

Maria, my love, thank you for having patience with me during my PhD time. I know, my time for you was always very limited, and I had to follow a very consequent schedule to get everything managed: business, private life as a couple, building and renovating our house, our little Theodor, and the big PhD project.

Contents

1 Opening

The real world and its behaviour seem to be uncertain for humans. The traditional mathematical equations and computer models are usually deterministic and cannot handle such uncertainties. For real-world applications and large-scale application scenarios, the input parameters of the models are often not exactly known or uncertain, which makes it hard to produce accurate results. The mathematical equations and computer models could produce much more accurate results and predictions when they are taking these uncertainties into account.

Since many decades, stochastic and statistical approaches such as classical Monte Carlo methods [128, 12] or stochastic (partial) differential equations [92, 179] are applied to investigate random behaviour in real-world applications.

Uncertainty quantification (UQ) also tries to tackle this by applying uncertainty to mathematical equations and models, and use it to simulate and predict the behaviour under the given uncertainties. Youssef Marzouk and Karen Willcox define UQ in [121] as follows:

> *"Uncertainty quantification (UQ) involves the quantitative characterization and management of uncertainty in a broad range of applications. It employs both computational models and observational data, together with theoretical analysis. UQ encompasses many different tasks, including uncertainty propagation, sensitivity analysis, statistical inference and model calibration, decision making under uncertainty, experimentaldesign, and model validation. UQ therefore draws upon many foundational ideas and techniques in applied mathematics and statistics (e.g., approximation theory, error estimation, stochastic modeling, and Monte Carlo methods) but focuses these techniques on complex models (e.g., of physical or sociotechnical systems) that are primarily accessible through computational simulation. UQ has become an essential aspect of the development and use of predictive computational simulation tools."*

As stated in the definition above, UQ allows to understand the influence of the uncertainty to the models' output and can help to reduce uncertainties. In this thesis, modern, state-of-the-art UQ methods are applied to large-scale simulation scenarios.

1.1 Motivation

In recent years, the wide field of UQ has been greatly developed by many new mathematical concepts and frameworks. The growing computational power of today's computers and computing clusters additionally accelerates the application of UQ.

Xiu introduced in [213] the general polynomial chaos expansion (gPCE) approach, which reduces the computational effort by keeping an excellent accuracy. Since then, many improvements [210, 200, 201, 69, 211, 34] at moderate cost have been developed and current research further improve the methods which are nicely covered in [53]. The gPCE approach is a category with many sub-methods. In this thesis, the stochastic collocation with the pseudo-spectral approach [210]—which is a very prominent gPCE based method—is intensively used to quantify the uncertainty of large-scale simulation scenarios efficiently.

The goal of this work is to bring the UQ methodology into new fields with existing large-scale legacy models without changing the existing model's equations and software, i.e. applying non-intrusive UQ methods only. In this thesis, two research fields have been taken into account, which may greatly benefit from the use of UQ.

The first one is the field of pedestrian dynamics, which tries to simulate pedestrians movements and behaviour. Three concrete scenarios are considered: *evacuation of a train station*, *evacuation of a building with separated families*, and *utilisation of a campus.* The second field covers hydrological modelling with the scenario of *large runoffs due to snow melting.*

The research questions for both fields are:

- Which uncertainty exists in the input parameters for the specific scenarios?
- Does the uncertainty in the input parameters have any influence on the results, and is it possible to improve the prediction quality.
- Which parameters have the most influence on the models' output under uncertainty?
- What are the challenges when quantifying the uncertainty?
- Can the uncertainty be visualised such that a proper investigation of the influence of the uncertainties is possible?

Another goal in this work is to perform and develop the UQ simulations efficiently, which covers: fast prototyping, short simulation time, high utilisation of computational resources, and the support for a fast interpretation of the results.

Besides the improvements and emerging UQ methods in recent years, many UQ software toolkits and libraries have been published and are available for academic and commercial use. Each UQ software toolkit has its advantages and disadvantages and typically focuses on the implementation of the latest UQ methods of a certain category.

For rapid prototyping, the UQ software toolkits have to have a clear application programming interface which easily allows to prototype new ideas and to perform parameter studies to find a proper setup for the UQ simulations.

Definition 1.1. UQ simulation: The term *UQ simulation* is used for a simulation run to quantify the uncertainty of a model, which requires to run the model many times, depending on the chosen UQ method.

Another important software requirement for UQ toolkits is the support for many computing environments, which allows to run the UQ simulation software on a PC for development purposes and scale up to large compute clusters (like high-performance computing (HPC) systems) for production runs. On each computing platform, an automatic and efficient utilisation of the computing resources is desired, to reduce the dependency on platform-specific features. Furthermore, it is also important to estimate how long a UQ simulation will take; this is rarely tackled in existing software. All these aspects require to choose a proper, lightweight UQ software toolkit. In this thesis, Chaospy [43] is chosen as a basis because of its lightweight solution and its flexibility. Nevertheless, it is still missing some required support to propagate the uncertainties efficiently: This will be completed and improved in this work to enable an all-in-one UQ simulation workflow that covers all phases of a UQ simulation on all relevant computing platforms.

The development aspects which have to be considered when quantifying the uncertainty on these chosen large-scale simulation scenarios should be analysed and documented. For efficient propagation, common scheduling techniques that are widely used in computing systems should be taken into account with an analysis of their idling behaviour when using in non-intrusive forward UQ simulations.

1.2 Contributions

The four main contributions in this thesis are driven by the idea of

Bring together: Modern non-intrusive UQ methods with large-scale simulation scenarios and investigate methods to quantify the uncertainty efficiently.

1. The first main contribution is the use of modern, state-of-the-art UQ methods (stochastic collocation with the pseudo-spectral approach (Section 2.2.2)) to quantify the uncertainty in the field of pedestrian dynamics (Chapter 6) with the concrete scenarios of *evacuation of a train station* (pedestrian dynamics scenario 1: Section 6.3), *evacuation of a building with separated families* (pedestrian dynamics scenario 2: Section 6.4), and *utilisation of a campus* (pedestrian dynamics scenario 3: Section 6.5). For the UQ simulations, the already existing pedestrian dynamics simulator Vadere [17, 91] (Section 6.1) is used. The second field is hydrological modelling (Chapter 7) with the scenario of *large runoffs due to snow melting* (hydrological modelling scenario: Section 7.3). To simulate the runoffs, the existing water balance model simulator LARSIM [105] (Section 7.1) is used.

   For both fields, it can be shown that the use of modern UQ methods to quantify the uncertainty works well and is very important to obtain results with statistical moments (such as mean or variance) compared to the results of a single run of the simulators, which gives an impression of the influence of the uncertainty. For each field, an analysis of the specific challenges when quantifying the uncertainty, as well as detailed information about the scenario, the UQ simulation setup and the interpretation of the results, is included. On top of the UQ simulation, a global sensitivity analysis is performed to determine which of the uncertain input parameters contributes most to the quantity of interest.

   Because the quantification of the uncertainties should be efficient, a definition of efficiency aspects and the resulting requirements of the UQ simulations are included (Section 2.5), which covers the aspects of development time to solution, simulation time, computational resources, and interpretation of results with some hints for visualisation techniques for the quantity of interest. All these aspects are considered in the proposed solutions in this thesis.

2. This leads to the second main contribution, the investigation of the propagation phase of a non-intrusive forward UQ simulation. The individual black-box model runs are usually independent, which allows to embarrassingly parallelise them. By analysing standard scheduling strategies (Section 4.4), it has been shown that due to idling, some of the computing resources do not fully participate the whole simulation time, while the others do intensively work. Because of that and the observation that the runtime of some individual black-box model runs does significantly vary depending on the values of the uncertain input parameters, an estimation of the overall runtime for the UQ simulation is very hard or even impossible. To solve this, the novel idea of using the runtime of a simulator as the quantity of interest and constructing a runtime estimator (Section 4.5) based on the general polynomial chaos expansion with the stochastic collocation with the pseudo-spectral approach is presented. The constructed runtime estimator is used to estimate the runtime depending on the input parameters. Depending on the predicted runtime, the individual black-box model runs are reordered by their estimated runtime to optimise the scheduling with the goal of reducing the idling time.

3. For some scenarios, it is a requirement to obtain the results of a UQ simulation very fast. This is often somehow problematic with large-scale scenario simulators, because of their long runtime (minutes to hours for a single black-box model simulation run) combined with the usually large number of required single black-box model runs for a whole UQ

simulation. For that, a data-driven surrogate model (the closed observables surrogate model (COSM) [28, 27, 24]), is used in a UQ simulation (pedestrian dynamics scenario 3: Section 6.5), which leads to the concept of the closed observables surrogate model in Section 4.7.1 that allows reducing the runtime for a UQ simulation by some orders of magnitude. This is the third main contribution.

4. The fourth main contribution is the implemented software "uncertainty quantification execution framework (UQEF)" (Chapter 5) on top of Chaospy, which includes the runtime estimation with the standard and the optimised scheduling strategies. UQEF offers the users predefined interfaces to include their custom models and statistical evaluations of the quantity of interest. All the propagation and the scheduling is done automatically by UQEF, which allows the users to seamless develop on a PC and to efficiently execute their UQ simulations on a computing environment such as HPC systems.

Further contributions are the structured collection and description of various aspects that have to be considered when implementing UQ simulations (Chapter 3). A highlight is the implemented *SampleDist* function which allows to automatically generate a probability distribution based on a kernel density estimator which can be used to describe an uncertain input parameter (Section 3.4), and gives the possibility to obtain the probability distribution of the quantity of interest (Section 3.5).

Table 1.1 gives an overview of the contributions and the used features within the investigated large-scale simulation scenarios.

	Stochastic collocation with the pseudo-spectral approach (Section 2.2.2)	*Sensitivity indices* (Section 2.3)	*Parameter study* (Section 3.3)	*QoI function visualisation* (Section 3.5)	*All-in-one simulation* (Section 4.1)	*Standard scheduling strategies* (Section 4.4)	*Optimised scheduling strategies* (Section 4.6)	*COSM* (Section 4.7)	*UQEF* (Chapter 5)
Pedestrian dynamics scenario 1 (Section 6.3)	✓	✓	✓	✓	✓	✓			✓
Pedestrian dynamics scenario 2 (Section 6.4)	✓	✓		✓	✓	✓	✓		✓
Pedestrian dynamics scenario 3 (Section 6.5)	✓	✓			✓	✓		✓	✓
Hydrological modelling scenario (Section 7.3)	✓	✓			✓	✓			✓

Table 1.1: List of features and strategies that are investigated in this thesis and are used in the case study scenarios to efficiently quantify the uncertainty.

1.3 Outline

The general outline of this thesis is visualised in Figure 1.1. Starting from this opening chapter, the required background information with a gentle introduction into uncertainty quantification including the used methods, an overview of the existing UQ software toolkits as well as the introduction into the used toolkit Chaospy, and the definition of the efficiency aspects is given in Chapter 2.

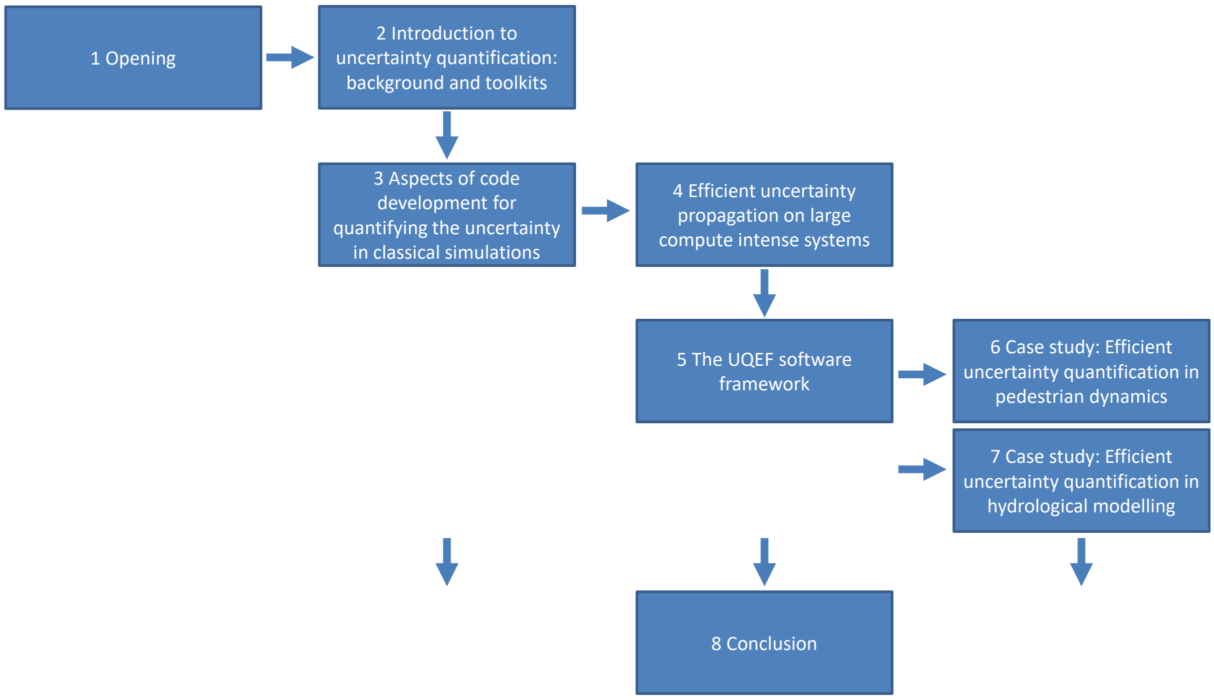

1 Opening
2 Introduction to uncertainty quantification: background and toolkits
3 Aspects of code development for quantifying the uncertainty in classical simulations
4 Efficient uncertainty propagation on large compute intense systems
5 The UQEF software framework
6 Case study: Efficient uncertainty quantification in pedestrian dynamics
7 Case study: Efficient uncertainty quantification in hydrological modelling
8 Conclusion

Figure 1.1: Overview about the thesis chapters. The arrows give hints about the possible reading orders.

Chapter 3 discusses different aspects when developing UQ simulations: The chapter starts with the introduction of some academic test functions and models that are used to show and explain various development aspects. It continues by describing how to select a suitable UQ method and how to select proper parameters for the chosen UQ method. The chapter also includes a brief survey on the modelling of uncertain parameters and gives an introduction to the developed *SampleDist* function, which is based on a kernel density estimator. For the interpretation of the results, various aspects are discussed, including the use of time series data for the QoI, the analysis of the uncertain parameters via sensitivity indices, the reconstruction of the QoI distribution, as well as the QoI function visualisation technique.

Chapter 4 describes the developed ideas to perform the propagation phase on large compute intense systems efficiently. It contains the analysis of the idling behaviour in the propagation phase for three standard scheduling strategies. With the novel idea of using the runtime of a model as the quantity of interest, a runtime predictor is constructed using the stochastic collocation with the pseudo-spectral approach. By using the runtime predictor to predict the runtime of each individual black-box model run, the individual runs are reordered with the goal of minimising the idling, which results in the optimised scheduling strategies. Additionally, the use of the closed observables to construct a data-driven surrogate model (the so-called COSM) that can be used to quantify the uncertainty in decision-making problems efficiently is presented.

In Chapter 5, the developed UQEF software framework is described in detail. The framework is designed to support the defined efficiency aspects as well as the ideas from Chapters 3 and 4.

First of all, a general overview of the software architecture with an illustrative source code example is given; followed by the description of the implementation of custom models and statistics. Then, it is described how a UQ simulation is parametrised. It also includes the description of the supported UQ methods, the scheduling and solver strategies, the support for parameter studies and finally, the automatic runtime measurements and predictions, and the optimised scheduling strategies.

Chapter 6 contains a case study for the field of pedestrian dynamics. In the beginning, the used pedestrian dynamics simulator Vadere is introduced with its specific challenges when quantifying the uncertainties. Then, the three scenarios: *evacuation of a train station*, *evacuation of a building with separated families*, and *utilisation of a campus* are presented. For each scenario, the general research question is discussed, followed by the UQ simulation setup, and the numerical results.

Chapter 7 contains the case study for the hydrological modelling field. It starts with the description of the used water balance model simulator LARSIM and its specific challenges when using in UQ simulations. After that, the specific scenario *large runoffs due to snow melting* is described with the used UQ simulation setup and the obtained numerical results.

The thesis is concluded in Chapter 8 with a summary and an outlook including a discussion about the achievements and the results, as well as possible future research topics on how to further use and improve the developed ideas.

2 Introduction to uncertainty quantification: background and toolkits

Uncertainty quantification (UQ) is a wide field of methods to determine the influence of uncertainties (unknowns) in applications and has become a prominent tool in research for many years. The information about the uncertain influence helps to understand the behaviour of applications under these conditions, which can support decision-making processes and help to reduce uncertainties by knowing the problematic uncertain sources. The increasing computational power of today's computers combined with new advances in the methods makes UQ even more accessible for many different applications, in academia as well as in real-world applications. General introductions for UQ are available by McClarren [123], Smith [178], Sullivan [186], or Iaccarino [72], for instance.

The uncertainties can be classified into two categories [178]: the aleatoric and the epistemic uncertainties. Aleatoric uncertainty, which is also known as statistical uncertainty, categorises the unknowns in the investigated applications that are different each time the experiment is run. These uncertainties are inherently present. The epistemic or systematic uncertainties on the other side contain uncertainties which are caused by errors, e.g. due to not exactly measured values, or some particular data that are not taken into account into the application (e.g. simplifying model assumptions), but causing effects which seem to vary the results.

Another categorisation of the sources of uncertainties is described in [86] as: parameter uncertainty, model inadequacy, residual variability, parametric variability, observation error, and code uncertainty. In this thesis, the uncertainties that are considered as input parameters of the application's models are taken into account. These can be caused by the lack of knowledge about the true values of some input parameters or measurement (observation) errors.

In UQ, forward and inverse problems are distinguished, as illustrated in Figure 2.1.

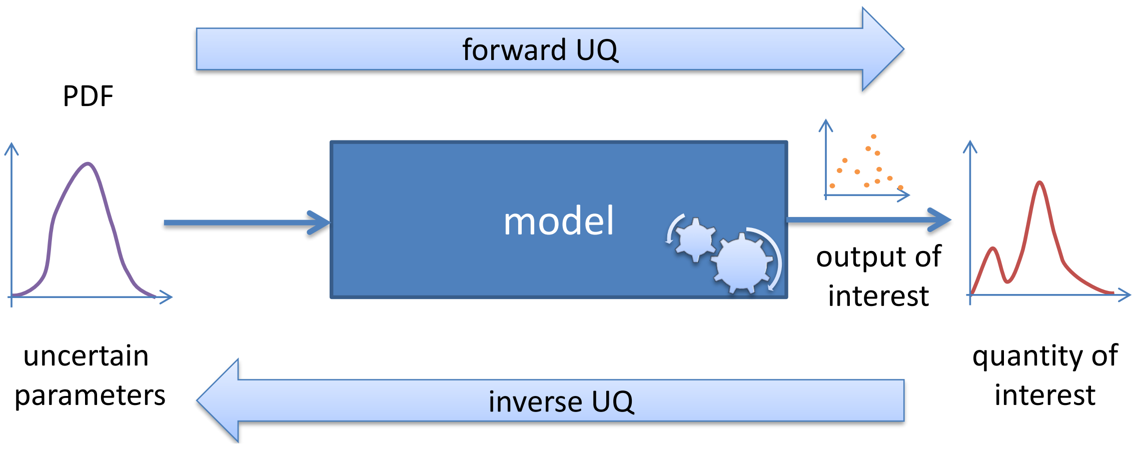


Figure 2.1: Illustration of forward and inverse uncertainty quantification.

In forward UQ, the impact of the uncertain parameters, which are usually modelled with known or assumed probability distributions, to some values of interest (VoI) of the resulting model outputs (also called output of interests (OoI)) are quantified with statistical moments: with the so-called quantities of interest (QoI). The inverse methods try to find the unknown parameter values by using some experimental available measurements of the model outputs. It is possible to gather the probability distributions for the unknown parameter values during the simulations. Inverse methods are known to be much more complex than forward UQ and require therefore much more computational effort.

Has an application to be modified to quantify its uncertainty? This is the difference between intrusive and non-intrusive UQ methods. In intrusive methods, the models' equations or the source code of a model has to be changed to be able to quantify the uncertainties. A prominent and efficient method for that is stochastic Galerkin [54], because if its proved accuracy. But changing the model formulation is often very complicated and can take a lot of time. Especially for large-scale scenarios where huge source code bases exist—which are grown over many years—changes are not easy. Additionally, some applications are closed source or only commercially available, whose sources are then not possible to change. The non-intrusive methods for UQ combine all approaches where the model is used as it is, and mostly considered as a black-box. No reformulation of the equations and the source code is necessary to quantify the uncertainties.

In this thesis, the focus is on *efficient non-intrusive forward uncertainty quantification for large-scale simulation scenarios*. All uncertain input parameters are assumed to be stochastically independent.

The remainder for this chapter is as follows: Sections 2.1 and 2.2 give an overview and introduction into the uncertainty quantification methods used in this thesis. To understand the influence of each uncertain parameter, a short introduction is given to the field of global sensitivity analysis in Section 2.3. An overview of existing toolkits for UQ is given in Section 2.4, with an introduction into the Chaospy toolkit, which is extensively used in this thesis. Section 2.5 lists various requirements that are required to quantify the uncertainty efficiently.

2.1 Phases of non-intrusive forward UQ simulations

A non-intrusive forward UQ simulation consists of three major phases, the assimilation, the propagation, and the certification, which are visualised in Figure 2.2. The phases are explained in more detail in the following.

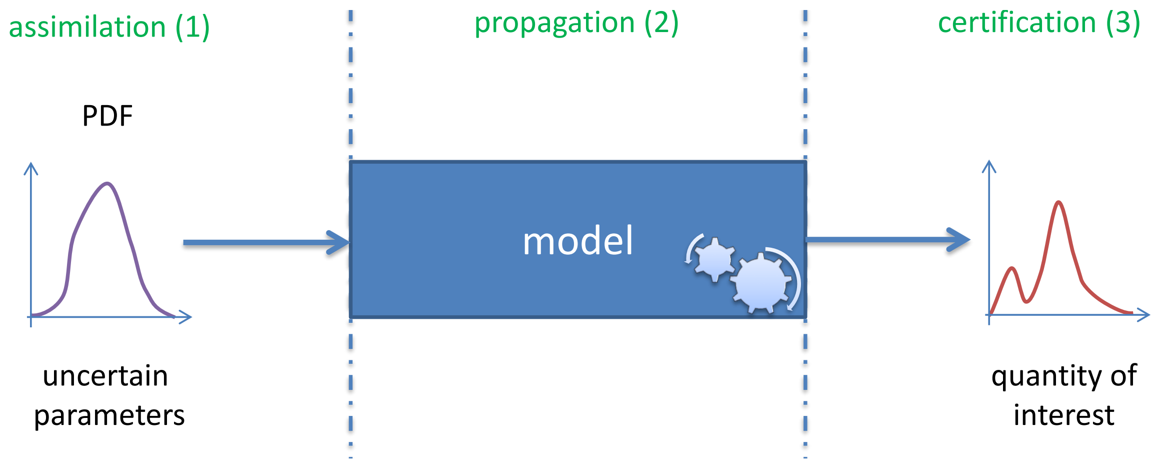


Figure 2.2: Illustration of a non-intrusive forward UQ simulation with its three phases: assimilation, propagation, and certification.

Assimilation

The assimilation is the first of the three phases. In this thesis, it is divided into a theoretical and a practical part. The theoretical part consists of all investigations on the uncertain parameters and its values. The goal is to find a suitable stochastic representation, mostly a probability distribution for each uncertain parameter. The choice of the applied UQ method and its general setup is also mostly part of this early phase. The actual usage of the theoretically gathered knowledge is included in the practical part. Here, the values for the unknown parameters are—depending on the UQ method—generated and prepared for the second phase, the propagation.

Propagation

In the propagation phase, the model is called several times with the values representing the unknown parameters, which are generated in the assimilation phase. This phase is usually very similar for different non-intrusive forward UQ methods—but the number of required black-box runs differ significantly on the chosen UQ method. This requires to perform the propagation efficiently to obtain the results in a reasonable time. After each individual black-box model run, its values of interest are extracted and somehow stored for the certification phase.

Certification

The certification phase is the last step. Once all black-box model runs have finished in the previous propagation phase and all outputs of interest are present, the practical part of the certification can start. The values of interest are statistically evaluated, and the quantities of interest are determined. This is again very specific to the chosen UQ method. In this thesis, the practical part also comprises the generation of the resulting numbers, tables, and plots. The theoretical part of the certification phase is the actual interpretation of the quantities of interest, for which the numbers, tables, and the plots are useful. This is very application-specific and usually requires a lot of domain knowledge.

2.2 Non-intrusive forward UQ methods

In uncertainty quantification, there exist several non-intrusive methods for forward problems (see [178, 186] for an overview). This work uses the classical sampling-based Monte Carlo method (Section 2.2.1), because of its ease of use for comparison purposes. As a state-of-the-art method, stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is mainly used, to benefit of its proven high accuracy and the comparatively less computational effort for low (1–3) to mid (4–12) dimensional problems. Additionally, the point collocation method (Section 2.2.3) is considered, because it is a powerful tool that can be used together with Monte Carlo based samples to generate more QoI information that is not possible with pure Monte Carlo.

2.2.1 Monte Carlo method

The Monte Carlo method [128] is a well known method in statistics for several decades. With the increasing computational power of today's computers, it also becomes usable for small to mid-sized application scenarios. For a general introduction and the variations of the Monte Carlo method, the reader is referred to [134, 100].

The basic idea in uncertainty quantification is to model the uncertain input parameters of a model as random variables $\boldsymbol{\zeta}$. The general form of the expectation value $\mathbb{E}$ respectively, the mean μ is defined as

$$\mu = \mathbb{E}[f(x,t,\boldsymbol{\zeta})] = \int_{\Omega} f(x,t,\boldsymbol{\zeta}) d\boldsymbol{\zeta}\,, \tag{2.1}$$

with $f(x,t,\boldsymbol{\zeta})$ being the model function whose expectation value should be estimated under uncertainty. $\boldsymbol{\zeta}$ is a set of M independent random variables $\boldsymbol{\zeta} = (\zeta_1, \ldots, \zeta_M)$. Typically, the model function f depends additionally on space x and time t.

To numerically evaluate the integral of Equation (2.1) via the Monte Carlo method, $\mathcal{M}$ sample evaluations of f are performed and equally weighted by dividing the sum by $\mathcal{M}$. This results in

$$\mu \approx \hat{\mu} = \mathbb{E}[f(x,t,\zeta)] = \frac{1}{\mathcal{M}} \sum_{i=1}^{\mathcal{M}} f(x,t,n_i) \tag{2.2}$$

for the numerical estimation of μ. For each independent random variable in $\boldsymbol{\zeta}$, a suitable probability distribution is applied, and a sample value that follows the distribution is drawn. To generate sample values that follow a certain probability distribution for an uncertain parameter, pseudo-random number generators [94] are preferably used. The vector n has a length of M nodes, and each node n_i is a set of values and contains for each uncertain parameter one value with which f is called. The return value of the model function f is the OoI, and the resulting μ is a statistical moment that describes the QoI. The variance σ^2 or Var is estimated by

$$\sigma^2 = \text{Var} \approx \frac{1}{\mathcal{M}-1} \sum_{i=1}^{\mathcal{M}} [f(x,t,n_i) - \hat{\mu}]^2 \,, \tag{2.3}$$

and the standard deviation σ or StdDev by

$$\sigma = \text{StdDev} = \sqrt{\sigma^2} \approx \sqrt{\frac{1}{\mathcal{M}-1} \sum_{i=1}^{\mathcal{M}} [f(x,t,n_i) - \hat{\mu}]^2} \,. \tag{2.4}$$

The Pth percentile (p_P) is defined as P percent of the values are lower or equal of this value. A prediction interval that represents 80% of the values can be spanned by $p_{90} - p_{10}$. A prominent method to determine a percentile is the nearest-rank-method: All values that are collected for an OoI are stored in ascending order (by their values) in a list with $\mathcal{M}$ elements. The index i is determined with

$$i = \left\lceil \frac{P}{100} \times \mathcal{M} \right\rceil \,, \tag{2.5}$$

and the value at the position of the index i can than be interpreted as P percent of the values are lower or equal that value.

The Monte Carlo method is widely used in many different disciplines because it is easy to understand and to implement. Its convergence rate is relatively independent of the input dimension (the number of uncertain parameters M) which makes these methods usable for large dimensional problems. Due to the independence of the individual samples, each evaluation of the model f is also independent, which allows evaluating f in an embarrassingly parallel manner.

A drawback of Monte Carlo is its slow convergence rate of $O(\frac{1}{\sqrt{\mathcal{M}}})$ which requires many sample evaluations; hence, it is often not feasible to use it for large-scale scenarios.

To improve the convergence rate of the classical Monte Carlo, various improvements exist, e.g. variance-reduction techniques [90], quasi Monte Carlo techniques [12] with a convergence rate of $O(\frac{\log(\mathcal{M})^M}{\sqrt{\mathcal{M}}})$, and multi-fidelity [140, 145] and multi-level [55] Monte Carlo techniques.

2.2.2 Stochastic collocation with the pseudo-spectral approach

The stochastic collocation with the pseudo-spectral approach is a very prominent UQ method since Xiu publishes it in [210]. The method is based on the generalised polynomial chaos expansion (gPCE) [214], which is a generalisation of the Wiener chaos expansion or Wiener-chaos introduced by Wiener in [206].

The general form of the generalised polynomial chaos expansion for the solution of $U(x,t,\boldsymbol{\zeta})$ reads

$$U(x,t,\boldsymbol{\zeta}) = \sum_{j=0}^{\infty} \underbrace{c_j(x,t)}_{\substack{\text{spatio-}\\\text{temporal}}} \cdot \underbrace{\Phi_j(\boldsymbol{\zeta})}_{\text{random}} . \tag{2.6}$$

In gPCE, the coefficients c_j with the spatio-temporal part are separated from the random part with the $\Phi_j(\boldsymbol{\zeta})$ base functions. The base functions $\Phi_j(\boldsymbol{\zeta})$ are orthogonal polynomials that fit to the probability distributions [213] of the uncertain input parameters in $\boldsymbol{\zeta}$. To numerically evaluate Equation (2.6), the infinite sum is truncated after $N+1$ terms and results in

$$U(x,t,\boldsymbol{\zeta}) \approx u_N(x,t,\boldsymbol{\zeta}) := \sum_{j=0}^{N} c_j(x,t) \cdot \Phi_j(\boldsymbol{\zeta}) \tag{2.7}$$

for the solution of $U(x,t,\boldsymbol{\zeta})$. The number N of the $N+1$ terms of the expansion in the multivariate case can be determined by

$$(N+1) = \frac{(M+P)!}{M!P!}, \tag{2.8}$$

with M being the number of independent uncertain parameters, and P is the highest order of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$.

A scalar product (also called inner product) in the form of

$$\langle \Phi_i(\boldsymbol{\zeta}), \Phi_j(\boldsymbol{\zeta}) \rangle := \int_{\Omega} \Phi_i(\boldsymbol{\zeta}) \Phi_j(\boldsymbol{\zeta}) W(\boldsymbol{\zeta}) d\boldsymbol{\zeta} = \mathbb{E}[\Phi_i(\boldsymbol{\zeta}) \Phi_j(\boldsymbol{\zeta})] \tag{2.9}$$

is defined that represents the expectation value $\mathbb{E}$ of the product of two orthogonal polynomials. $W(\boldsymbol{\zeta})$ is called weighting function that represents the multivariate probability density function (PDF) of the random variables in $\boldsymbol{\zeta}$. The orthogonal polynomials satisfy

$$\langle \Phi_i(\boldsymbol{\zeta}), \Phi_j(\boldsymbol{\zeta}) \rangle = \gamma_j \delta_{ij} , \tag{2.10}$$

with δ_{ij} being the Kronecker delta that has the form

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j , \end{cases} \tag{2.11}$$

where γ_i is the normalisation factor

$$\gamma_j = \langle \Phi_j(\boldsymbol{\zeta}), \Phi_j(\boldsymbol{\zeta}) \rangle = \mathbb{E}[\Phi_j^2(\boldsymbol{\zeta})] \tag{2.12}$$

because $\Phi_j(\boldsymbol{\zeta})$ are orthogonal but not necessarily orthonormal.

How the coefficients c_j are determined is very specific to the used UQ method. In stochastic collocation with the pseudo-spectral approach, the coefficients c_j are determined by an integral of the form

$$c_j(x,t) := \frac{1}{\gamma_j} \int f(x,t,\boldsymbol{\zeta}) \Phi_j(\boldsymbol{\zeta}) W(\boldsymbol{\zeta}) d\boldsymbol{\zeta} . \tag{2.13}$$

The OoI of f is projected into the direction of $\Phi_j(\boldsymbol{\zeta})$ and weighted with $W(\boldsymbol{\zeta})$. The integral is typically approximated via a quadrature rule. In this thesis, the Gaussian quadrature is used for that approximation, which reads as follows

$$c_j(x,t) \approx \frac{1}{\gamma_j} \sum_{i=1}^{Q} f(x,t,z_i) \Phi_j(z_i) w_i . \tag{2.14}$$

It uses Q number of collocation points z_i with a corresponding weight w_i. The collocation points z_i are the roots of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$, and the weight is determined according to the probability density functions of the random variables in $\boldsymbol{\zeta}$. The model function f is exactly evaluated at the collocation points z_i, which results—due to the tensor product [194] of all number of collocation points q for each parameter in the Gaussian quadrature—in $Q = q^M$ evaluations of f. Each z_i contains exactly M number of values, for each uncertain parameter one value.

An important part of the accuracy of this UQ method comes from the proper choice of the basis functions (orthogonal polynomials) for the gPCE and corresponding probability distributions for the random variables in $\boldsymbol{\zeta}$. Xiu describes according to the Wiener–Askey scheme in [213] the correspondence between common probability distributions and orthogonal polynomial basis functions for the gPCE. Table 2.2 contains an overview of common combinations for continuous and discrete probability distributions.

	Distribution of ζ	gPCE basis polynomials	Support
Continuous	Gaussian	Hermite	$(-\infty, \infty)$
	Gamma	Laguerre	$[0, \infty)$
	Beta	Jacobi	$[a, b]$
	Uniform	Legendre	$[a, b]$
Discrete	Poisson	Charlier	$\{0, 1, 2, \dots\}$
	Binomial	Krawtchouk	$\{0, 1, \dots, N\}$
	Negative binomial	Meixner	$\{0, 1, 2, \dots\}$
	Hypergeometric	Hahn	$\{0, 1, \dots, N\}$

Table 2.2: Common combinations of probability distributions and polynomial basis functions in the general polynomial chaos expansion according to the Wiener–Askey scheme (compare [213]).

If for a 1D problem an uncertain parameter is Gaussian (normal) $\zeta \sim \mathcal{N}(0, 1)$ ($\mu_{u_N} = 0$ and $\sigma^2 = 1$) distributed, and a model f within a continuous space should be evaluated, then the Hermite basis polynomials are chosen. Typically the probabilistic Hermite polynomials in the form

$$\Phi_0(\boldsymbol{\zeta}) = 1, \quad \Phi_1(\boldsymbol{\zeta}) = \boldsymbol{\zeta}, \quad \Phi_2(\boldsymbol{\zeta}) = \boldsymbol{\zeta}^2 - 1, \quad \Phi_3(\boldsymbol{\zeta}) = \boldsymbol{\zeta}^3 - 3\boldsymbol{\zeta}, \quad \dots \tag{2.15}$$

are used. The weight function $W(\zeta)$ is then the probability density function (PDF) for the $\mathcal{N}(0, 1)$ distributed parameter

$$W(\zeta) = \frac{1}{\sqrt{2\pi}} e^{-\zeta^2/2} \, . \tag{2.16}$$

Figure 2.3 visualises the resulting values of the PDF on the collocation points z_i (Figure 2.3(a)) and the generated weights w_i (Figure 2.3(b)) for the combination of the $\zeta \sim \mathcal{N}(0, 1)$ random variable and the orthogonal Hermite polynomials. Some more visualisations can be found in Appendix A.1.

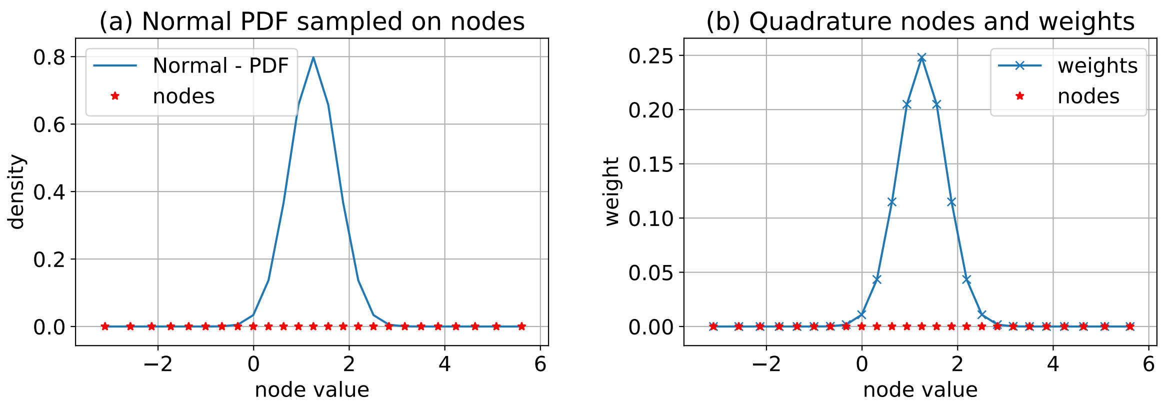


Figure 2.3: Visualisation of the probability density function of a $\zeta \sim \mathcal{N}(0,1)$ distributed random variable ζ in (a) and the corresponding generated collocation points z_i with its weights w_i in (b). For this visualisation, $q = 25$ collocation points for ζ are used.

The first statistical moment, the mean μ_{u_N} or the so-called expectation value $\mathbb{E}$, is evaluated as follows

$$\begin{aligned}\mu \approx \mu_{u_N} = \mathbb{E}[u_N(x,t,\boldsymbol{\zeta})] &= \mathbb{E}[\sum_{j=0}^{N} c_j \Phi_j(\boldsymbol{\zeta})] \underset{(N<\infty)}{=} \sum_{j=0}^{N} c_j \mathbb{E}[\Phi_j(\boldsymbol{\zeta})] \\ &= \sum_{j=0}^{N} c_j \mathbb{E}[\Phi_j(\boldsymbol{\zeta}) \cdot 1] = \sum_{j=0}^{N} c_j \mathbb{E}[\Phi_j(\boldsymbol{\zeta}) \underbrace{\Phi_0(\boldsymbol{\zeta})}_{=1}] \\ &= \sum_{j=0}^{N} c_j \underbrace{\langle \Phi_j(\boldsymbol{\zeta}), \Phi_0(\boldsymbol{\zeta}) \rangle}_{=0,\text{ if } j \neq 0} = c_0 \underbrace{\langle \Phi_0(\boldsymbol{\zeta}), \Phi_0(\boldsymbol{\zeta}) \rangle}_{\gamma_0 = 1} \\ &= c_0 \cdot 1 = c_0 \,. \end{aligned} \tag{2.17}$$

Due to the nature of orthogonal polynomials, $\Phi_0(\boldsymbol{\zeta})$ is always 1 and $\langle \Phi_j(\boldsymbol{\zeta}), \Phi_0(\boldsymbol{\zeta}) \rangle = 0$ if $j \neq 0$, the expectation value $\mathbb{E}$ is reduced to c_0.

The second statistical moment, the variance σ^2 or Var can be approximated in the simplified form of

$$\begin{aligned}\sigma^2 \approx \sigma^2_{u_N} \triangleq \mathrm{Var}(u_N(x,t,\boldsymbol{\zeta})) &= \mathbb{E}[(u_N(x,t,\boldsymbol{\zeta}) - \underbrace{\mathbb{E}[u_N(x,t,\boldsymbol{\zeta})]}_{c_0 = c_0\Phi_0(\boldsymbol{\zeta})})^2] \\ &= \mathbb{E}[(\sum_{j=0}^{N} c_j \Phi_j(\boldsymbol{\zeta}) - c_0 \Phi_0(\boldsymbol{\zeta}))^2] = \mathbb{E}[(\sum_{j=1}^{N} c_j \Phi_j(\boldsymbol{\zeta}))^2] \\ &= \langle \sum_{j=1}^{N} c_j \Phi_j(\boldsymbol{\zeta}), \sum_{k=1}^{N} c_k \Phi_k(\boldsymbol{\zeta}) \rangle = \sum_{j=1}^{N} c_j \langle \Phi_j(\boldsymbol{\zeta}), \sum_{k=1}^{N} c_k \Phi_k(\boldsymbol{\zeta}) \rangle \\ &= \sum_{j=1}^{N} \sum_{k=1}^{N} c_j c_k \underbrace{\langle \Phi_j(\boldsymbol{\zeta}), \Phi_k(\boldsymbol{\zeta}) \rangle}_{=0,\text{ if } j \neq k} = \sum_{j=1}^{N} c_j^2 \underbrace{\langle \Phi_j(\boldsymbol{\zeta}), \Phi_j(\boldsymbol{\zeta}) \rangle}_{=\gamma_j} \\ &= \sum_{j=1}^{N} c_j^2 \gamma_j \,. \end{aligned} \tag{2.18}$$

Because it is known from Equation (2.17) that $\mathbb{E}[u_N(x,t,\boldsymbol{\zeta})] = c_0$ and $\langle \Phi_j(\boldsymbol{\zeta}), \Phi_k(\boldsymbol{\zeta}) \rangle = 0$ if $j \neq k$, the double sum can be reduced to a single sum by only considering the coefficients c_j.

The standard deviation can, again, be determined with

$$\sigma = \sqrt{\sigma^2}\,. \tag{2.19}$$

According to [210, 212], the convergence of the pseudo-spectral approach for smooth model functions shows fast convergence rates depending on three error contributions: First, the projection error of the expansion due to the finite number of terms N for Equation (2.7); Second, the numerical aliasing error introduced by the integration rule and the used number of collocation points Q for Equation (2.14); And finally, the error introduced by numerically solving the model function f at the collocation points z_i. Hence, higher N and higher Q usually results in more accurate results. In Section 3.3, there is a discussion about how to choose Q and N in such a way that they fit properly together. The method works well for low (1–3) to mid (4-10) dimensional (number of uncertain parameters M) problems. Higher-dimensional problems become more feasible in combination with sparse grids [58, 46, 45, 208] and adaptive sparse grid methods [38, 40]. In [200, 201], it is shown that polynomial chaos methods do not work well for non-smooth models. For that, improvements exist, e.g. in [138].

In traditional publications, usually, the mean μ_{u_N} and the variance σ^2 are of interest. But the whole gPCE of $u_N(x,t,\boldsymbol{\zeta})$ (Equation (2.7)) is also very useful when evaluated: It can be used as a surrogate which returns the most likely OoI which represents the model function f under the given uncertain conditions. This feature is intensively used in this thesis to formulate a runtime surrogate of a model function in Section 4.5.

2.2.3 Point collocation

The point collocation method [69] is also a very prominent and widely used gPCE method in uncertainty quantification based on Equation (2.6). It consists of sampling the random space of the random variables $\boldsymbol{\zeta}$, as in the Monte Carlo method to obtain the collocation points z_i, and estimating the coefficients c_j with linear regression for Equation (2.7). It is an interpolation method by using a matrix inversion approach, which uses the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ as the basis functions, and the solves of $u_N(x,t,\boldsymbol{\zeta})$ on the sampled collocation points z_i—which are the OoI values of solving the model function $f(x,t,z_i)$ on all z_i. As the coefficients c_j are not known, they are determined by solving a linear system of equations of the form

$$\begin{pmatrix} \Phi_0(z_0) & \Phi_1(z_0) & \dots & \Phi_N(z_0) \\ \Phi_0(z_1) & \Phi_1(z_1) & \dots & \Phi_N(z_1) \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_0(z_N) & \Phi_1(z_N) & \dots & \Phi_N(z_N) \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} u_N(x,t,z_0) \\ u_N(x,t,z_1) \\ \vdots \\ u_N(x,t,z_N) \end{pmatrix}. \tag{2.20}$$

The gPCE in Equation (2.7) is also truncated after $N+1$ terms. This means that a minimum of $N+1$ samples (collocation points z_i) is required. In [69], it is shown that with an oversampling of $2\cdot(N+1)$ collocation points z_i, the accuracy of the whole gPCE can be improved.

To evaluate the QoI with statistical moments, the same approaches as for the pseudo-spectral approach for the mean μ_{u_N} (Equation (2.17)) and the variance $\sigma^2_{u_N}$ (Equation (2.18)) can be used. Also, the usage of the whole $u_N(x,t,\boldsymbol{\zeta})$ (Equation (2.7)) is possible, which offers a wide range of applications.

Further improvements can be achieved by using Latin hypercube sampling (LHS) [124] or Hammersley sampling (HS) [62], which uses a more systematic way of creating the samples and ensures better coverage of the uncertain space of the random variables (see [69] for more details) compared to a random sampling of the probability distributions $\boldsymbol{\zeta}$.

Another improvement on the point collocation method exists: Instead of sampling the distributions to obtain the collocation points z_i, the roots of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$

are used, similar as in the pseudo-spectral approach in Section 2.2.2. This is known as the "probabilistic collocation" approach as it is introduced in [34].

Compared to the pseudo-spectral approach in Section 2.2.2, the point collocation has a slower convergence. However, one advantage is that at the collocation points z_i, the estimation may be more accurate compared to the pseudo-spectral approach. As a drawback, the error can be quite large between the collocation points. A detailed discussion of this aspect can be found in [34].

2.3 Global sensitivity analysis

In the context of UQ, global sensitivity analysis is used on top of uncertainty quantification to determine the relation between parameter input values and the output of interests of a model. Global sensitivity analysis allows to tackle the following topics:

- Investigate if a model's output of interest is sensitive to the variation of the values of a certain input parameter.
- Understand which parameters contribute most to the output of interest of a model.
- Determine the contribution to the variance (importance) in the output of interest for every single uncertain parameter as well as in combination with the other parameters.
- Identify the parameters that do not contribute much and fix this to deterministic values to reduce the number of uncertain parameters and therefore, the dimension which is typically related to the computational costs of a method.

For a general overview of global sensitivity analysis, [157] is recommended. Global sensitivity analysis is a variance-based method [158]. The results may be expressed as first-order, higher-order, and total-order sensitivity indices, which describes, respectively, the single contribution, the contribution of combinations, and the total contribution with its interactions of the parameters. All indices represent values between $[0, 1]$, where 0 means there is no contribution to the variance and values up to 1 means there is some considerable contribution. The indices are also called Sobol' indices[1] [180] because Sobol generalised the variance-based sensitivity indices method to a high dimensional expansion of the form

$$f(x,t,\boldsymbol{\zeta}) = f_0 + \sum_{i=1}^{M} f_i(x,t,\zeta_i) + \sum_{1\le i<j\le M} f_{ij}(x,t,\zeta_i,\zeta_j) + \ldots + f_{12\ldots M}(x,t,\boldsymbol{\zeta})\,, \tag{2.21}$$

which requires that the output of interest of a model function f can be additively expressed, where $\boldsymbol{\zeta} := \{\zeta_1, \zeta_2, \ldots, \zeta_M\}$. This implies that the variance $\operatorname{Var}(f(x,t,\boldsymbol{\zeta}))$ can also be expressed with an expansion

$$\operatorname{Var}(f(x,t,\boldsymbol{\zeta})) = \sum_{i=1}^{M} \operatorname{Var}(f_i(x,t,\zeta_i)) + \sum_{1\le i<j\le M} \operatorname{Var}(f_{ij}(x,t,\zeta_i,\zeta_j)) + \ldots + \operatorname{Var}(f_{12\ldots M}(x,t,\boldsymbol{\zeta}))\,. \tag{2.22}$$

$\operatorname{Var}(f(x,t,\boldsymbol{\zeta}))$ is also called unconditional variance, because it contains the full information on the variance on the output of interest of $f(x,t,\boldsymbol{\zeta})$.

Consider the conditional variance $\operatorname{Var}(f_i(x,t,\zeta_i))$ of ζ_i given as

$$\operatorname{Var}(f_i(x,t,\zeta_i)) = \operatorname{Var}(\mathbb{E}[f(x,t,\boldsymbol{\zeta})|\zeta_i])\,. \tag{2.23}$$

The first-order sensitivity indices S_i can be determined by

$$S_i = \frac{\operatorname{Var}(\mathbb{E}[f(x,t,\boldsymbol{\zeta})|\zeta_i])}{\operatorname{Var}(f(x,t,\boldsymbol{\zeta}))} = \frac{\operatorname{Var}(f_i(x,t,\zeta_i))}{\operatorname{Var}(f(x,t,\boldsymbol{\zeta}))}\,, \tag{2.24}$$

[1]In this thesis, the more general term sensitivity indices is used.

and $\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))$ comprises the unconditional variance estimators for σ^2 of Equation (2.3) or Equation (2.18).

With $\mathrm{Var}(f_{ij}(x,t,\zeta_i,\zeta_j))$, the joint effect off (ζ_i,ζ_j) on $f(x,t,\boldsymbol{\zeta})$ can be measured

$$\begin{aligned}\mathrm{Var}(f_{ij}(x,t,\zeta_i,\zeta_j)) &= \mathrm{Var}[\mathbb{E}(f(x,t,\boldsymbol{\zeta})|\zeta_i,\zeta_j)] - \mathrm{Var}[\mathbb{E}(f(x,t,\boldsymbol{\zeta})|\zeta_i)] - \mathrm{Var}[f(x,t,\boldsymbol{\zeta})(Y|\zeta_j)] \\ &= \mathrm{Var}[\mathbb{E}(f(x,t,\boldsymbol{\zeta})|\zeta_i,\zeta_j)] - \mathrm{Var}(f_i(x,t,\zeta_i)) - \mathrm{Var}(f_j(x,t,\zeta_j))\,,\end{aligned} \tag{2.25}$$

which leads to the second-order sensitivity indices S_{ij} of the form

$$\begin{aligned}S_{ij} &= \frac{\mathrm{Var}[\mathbb{E}(f(x,t,\boldsymbol{\zeta})|\zeta_i,\zeta_j)] - \mathrm{Var}[\mathbb{E}(f(x,t,\boldsymbol{\zeta})|\zeta_i)] - \mathrm{Var}[f(x,t,\boldsymbol{\zeta})(Y|\zeta_j)]}{\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))} \\ &= \frac{\mathrm{Var}(f_{ij}(x,t,\zeta_i,\zeta_j))}{\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))}\,.\end{aligned} \tag{2.26}$$

Higher-order interaction can also be expressed with

$$S_{i12\ldots M} = \frac{\mathrm{Var}(f_{12\ldots M}(x,t,\boldsymbol{\zeta}))}{\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))}\,. \tag{2.27}$$

The total-order indices S_{T_i} are often of great interest, which allows determining the overall contribution of a parameter ζ_i to the output of interest of $f(x,t,\boldsymbol{\zeta})$. This can be written as

$$S_{T_i} = \frac{\mathbb{E}[\mathrm{Var}(f(x,t,\boldsymbol{\zeta})|\zeta_{\sim i})]}{\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))} = 1 - \frac{\mathrm{Var}(\mathbb{E}[f(x,t,\boldsymbol{\zeta})|\zeta_{\sim i}])}{\mathrm{Var}(f(x,t,\boldsymbol{\zeta}))}\,, \tag{2.28}$$

where $\zeta_{\sim i}$ comprises all contributions except the one of ζ_i.

The sensitivity indices can be computed with a Monte Carlo based method (see [157] for details) of evaluating the model function f many times, which is indeed an expensive computational way, especially for large-scale simulation scenarios. In the case of global sensitivity analysis combined with uncertainty quantification, the gPCE $u_N(x,t,\boldsymbol{\zeta})$ of Equation (2.7) can be used as a surrogate for f. With that, the same amount of evaluations have to be done—but once $u_N(x,t,\boldsymbol{\zeta})$ is generated, its evaluation is usually much faster than directly evaluating f, which makes this computationally feasible. This works with the pseudo-spectral approach in Section 2.2.2 as well as with the point collocation approach in Section 2.2.3.

Sudret describes in [183] how to use the coefficients c_j of Equation (2.14) for a polynomial chaos expansion to directly compute the sensitivity indices. This makes the computation of the sensitivity indices even more efficient. In this thesis, the Monte Carlo based version with the gPCE $u_N(x,t,\boldsymbol{\zeta})$ as the surrogate model is used, because this method is implemented in Chaospy.

2.4 Overview of existing toolkits for UQ

Since the very first start of the research for this thesis in December 2013, the number of toolkits for uncertainty quantification that are publicly known and available increased significantly. Now, about a dozen toolkits that support uncertainty quantification are available, and their development is spread around the globe. Many different teams, most of them from academia, develop and use the toolkits for research on the latest UQ methods.

In the beginning, it was not easy to find suitable toolkits for research. Now, there exist so many toolkits for different programming languages and purposes that it is hard to decide which one to use. In Section 2.4.1, there is a list (that may not be complete, but the listed toolkits are very prominent and where presented on the last SIAM UQ 2018 conference in the “mini-symposium on software for UQ”[2]) of UQ toolkits with some properties for a coarse comparison.

In this thesis, the Chaospy toolkit (see Section 2.4.2 for a gentle introduction) is used as a basis for the quantification of the uncertainties. Chaospy is available since 2014 on GitHub[3], it is actively developed, allows rapid prototyping, is open for changes, and has an easy-to-use application programming interface (API). That makes it a perfect candidate for efficiently quantifying the uncertainties (see Section 2.5 for efficiency requirements) for large-scale application scenarios because the time to solution, which also involves the development time is—besides others—an important factor in this thesis.

2.4.1 List of toolkits

The list in Table 2.3 is sorted in alphabetical order and shows the UQ toolkits with their programming language and language bindings, the license, and the maintaining person or institution. Since 2017, there is also a list of uncertainty quantification toolkits and related software on Wikipedia[4]. At the “mini-symposium on software for UQ” of the SIAM UQ 2018, SG++[5] [147], the sparse grid framework, has also been presented. But because it is not a software especially designed for uncertainty quantification, it is not listed in Table 2.3, but it should be taken into account when working with a higher number of uncertain parameters to reduce the number of required model evaluations using spatially adaptive sparse grids to compute the coefficients c_j of Equation (2.14).

Nr.	Toolkit	Programming language language bindings	License	Maintainer: Person(s)/ Institution	Cit.
1	**ALSVID-UQ**	C++, Python		Jonas Sukys, Siddhartha Mishra, SAM, ETH Zurich	[185]
2	**Chaospy**	Python	MIT	Jonathan Feinberg	[43]
3	**DAKOTA**	C++, Matlab, Scilab, Python	LPGL	Sandia National Laboratories	[1]
4	**EasyVVUQ**	Python	LPGL	Robin Richardson, et al., Centre for Computational Science, University College London	[153]
5	**MUQ**	C++, Python	MIT	Uncertainty Quantification Group Massachusetts Institute of Technology	[136]
6	**Mystic**	Python	BSD	The UQ Foundation	[125]
7	**OpenCOSSAN**	Matlab	LGPL v3.0	COSSAN Working Group, Institute for Risk and Uncertainty, University of Liverpool	[137]
8	**OpenTurns**	C++, Python	LGPL	OpenTurns initiative: industrial and academic partners	[6]

Table 2.3: List of known UQ toolkits as presented in the “mini-symposium on software for UQ” of the last SIAM UQ 2018 conference and some additionally selected ones appeared recently. The information are lastly updated in May 2020.

[2]https://www5.in.tum.de/wiki/index.php/SIAMUQ18_-_Slides_Minisymp_Software4UQ
[3]https://github.com/jonathf/chaospy
[4]https://en.wikipedia.org/wiki/List_of_uncertainty_propagation_software
[5]https://sgpp.sparsegrids.org

Nr.	Toolkit	Programming language language bindings	License	Maintainer: Person(s)/ Institution	Cit.
9	**Π4U**	C	GPL v2.0	CLT - Chair of Computational Science, ETH Zurich	[61]
10	**PROMETHEE**	R		Institut de Radioprotection et de Surete Nucleaire	[73]
11	**PSUADE**	C++	LGPL	UQ Method Development Team, Lawrence Livermore National Laboratory	[193]
12	**QUESO**	C++	LGPL	Oden Institute for Computational Engineering and Sciences, The University of Texas at Austin	[151]
13	**SmartUQ**	Standalone, Python, Matlab	Proprietary	SmartUQ LLC	[177]
14	**TASMANIAN**	C++, Python, Matlab, Fortran	BSD	Computational and Applied Mathematics, Oak Ridge National Laboratory	[182]
15	**Uncertainpy**	Python	GPL v3.0	Simen Tennøe	[192]
16	**UQ-PyL**	Python	GPL	Chen Wang, Qingyun Duan, Beijing Normal University	[202]
17	**UQLab**	Matlab	Academic and commercial license	Chair of Risk, Safety and Uncertainty Quantification of ETH Zurich	[120]
18	**UQTk**	C++, Python	BSD	Sandia National Laboratories	[20]
19	**Uranie**	C++, Python	LGPL	French Alternative Energies and Atomic Energy Commission (CEA)	[50]

Table 2.3 List of known UQ toolkits as presented in the “mini-symposium on software for UQ” of the last SIAM UQ 2018 conference and some additionally selected ones appeared recently. The information are lastly updated in May 2020.

2.4.2 Introduction to Chaospy

Chaospy is a framework for uncertainty quantification supporting forward propagation of uncertainty. Any model that is available within Python (Python code or available through Python bindings), or externally via system calls may be used. Chaospy is written in Python and relies on NumPy[6], numpoly[7], and SciPy[8]. Chaospy offers a very intuitive API, fast prototyping, is open-source, and open for changes and community contributions. Additionally, it is excellently supported by the authors which makes it a good choice for performing UQ simulations in research.

For the technical part of the assimilation phase (compare Figure 2.2), Chaospy contains many different configurable probability distributions[9] that can be combined into a multivariate distribution.

Chaospy provides functionality for the certification phase (see Figure 2.2) to calculate statistical moments, perform sensitivity analysis, and even generate the complete gPCE u_N (see

[6]NumPy library: `https://numpy.org/`
[7]numpoly library: `https://github.com/jonathf/numpoly`
[8]SciPy library: `https://www.scipy.org/`
[9]List of supported distributions in Chaospy: `https://chaospy.readthedocs.io/en/master/distributions/collection.html`

Equation (2.7)). Through Python pickling mechanisms, the data and the objects can be saved, which is heavily used in Chapter 4 and Chapter 5 to save and load gPCEs.

In the following, a short introduction into Chaospy coding for forward uncertainty quantification with Monte Carlo (Section 2.2.1), point collocation (Section 2.2.3), and stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is given, concluded by an analysis of the examples. Listing 2.1 shows the import of the required libraries.

```python
import chaospy as cp # import chaospy library
import numpy as np   # import numpy library
```

Listing 2.1: Import of Chaospy and NumPy libraries.

A simple `model()` function with two parameters (p_1 and p_2) is defined in Listing 2.2 that simply adds these parameters. The return value of the `model()` function is the value of interest (VoI).

```python
# model definition: simple example model with 2 parameters
def model(p1, p2):
    return p1+p2  # returns the value of interest (VoI)
```

Listing 2.2: Definition of a simple example model function.

The next step is to define probability distributions for the independent uncertain parameters of the `model()` function: For p_1 a uniform distribution with $U(-5, 1)$, and for p_2 a normal (Gaussian) distribution with $N(0, 1)$ is chosen:

```python
p1_dist = cp.Uniform(lower=-5, upper=1)  # creates a uniform distribution
p2_dist = cp.Normal(mu=0, sigma=1)       # creates a normal (Gaussian) distribution
```

Listing 2.3: Example of creating probability distributions with Chaospy using custom parameter values for the distributions: p_1 (`p1_dist`) is $U(-5, 1)$ and p_2 (`p2_dist`) is $N(0, 1)$ distributed.

With the join operator, the distributions for p_1 (`p1_dist`) and p_2 (`p2_dist`) are joined into a multivariate distribution in Listing 2.4:

```python
dists = cp.J(p1_dist, p2_dist)           # joins the dists to a multivariate dist.
```

Listing 2.4: Example of creating multivariate distributions with Chaospy: p_1 (`p1_dist`) and p_2 (`p2_dist`) are joined into a multivariate distribution.

Monte Carlo

All the steps mentioned above are independent of the specific UQ method, which now will be relevant. For Monte Carlo (Section 2.2.1), the example code is given in Listing 2.5. The Monte Carlo method relies on sampling the multivariate distribution with a high number, usually $\mathcal{M} > 100{,}000$ because of its slow convergence rate. The `sample()` method in the example creates 1,000 samples nodes (each node is a set of values for p_1 and p_2). The propagation is done in line 5: For each sample node, the `model()` function is called once. The values of interest are stored in the `solves` variable. Determining the statistical moments is then fairly easy because this can be done with basic statistical methods that are provided through NumPy (lines 8–11). Lines 14–18 print the resulting QoI values to the console. The full UQ Monte Carlo method example can be found in Appendix A.2.1.

```
1  nodes = dists.sample(1000)              # generate samples nodes according to the
2                                          # multivariate distributions
3
4  # 2. propagation
5  solves = [model(p1, p2) for p1, p2 in nodes.T]
6
7  # 3. certification: determine statistics of QoI
8  E       = np.mean(solves)               # determine E (mean)
9  Var     = np.var(solves)                # determine Var (variance)
10 StdDev  = np.std(solves)                # determine StdDev (standard deviation)
11 P5, P95 = np.percentile(solves, [5, 95]) # determine 5th and 95th percentile
12
13 # print results
14 print("E: {}".format(E))
15 print("Var: {}".format(Var))
16 print("StdDev: {}".format(StdDev))
17 print("P5: {}".format(P5))
18 print("P95: {}".format(P95))
```

Listing 2.5: Partial example of the Monte Carlo method with Chaospy: Sampling the multivariate distribution (`dists`), propagating uncertainty through the `model()` function, calculating the QoI statistics, and finally printing the results to the console.

Point collocation

The code example for the point collocation is contained in Listing 2.6. Since the point collocation method also relies on sampling the multivariate distribution and propagate the node values through the `model()` function, the code for this is the same as in lines 1–5 of Listing 2.5. The orthogonal polynomials are numerically generated on line 2, with the order 2 (which is also the highest order P of the orthogonal polynomial $\Phi_j(\boldsymbol{\zeta})$ of the gPCE (cf. Equation (2.7), Equation (2.8), and Equation (2.20)). In line 5, the gPCE is created through the regression method, and the polynomial object is stored in the `gPCE` variable, which contains all the information about the coefficients (c_j) and the orthogonality. With the help of the Chaospy functions, statistical moments are determined in lines 7–10 for the QoIs. The full UQ point collocation method example can be found in Appendix A.2.2.

```
1  # 3. certification: determine statistics of QoI
2  OP = cp.orth_ttr(2, dists) # creates orthogonal polynomials according the multivariate
3                             # distribution using three terms recursion
4
5  gPCE = cp.fit_regression(OP, nodes, solves) # generate gPCE
6
7  E       = cp.E(gPCE, dists)                 # determine E (mean)
8  Var     = cp.Var(gPCE, dists)               # determine Var (variance)
9  StdDev  = cp.Std(gPCE, dists)               # determine StdDev (standard deviation)
10 P5, P95 = cp.Perc(gPCE, [5, 95], dists)     # determine 5th and 95th percentile
```

Listing 2.6: Partial example of the point collocation method with Chaospy: Numerical generation of orthogonal polynomials, fitting the gPCE with regression, and determine the resulting QoI.

Stochastic collocation with the pseudo-spectral approach

Line 3 of Listing 2.7 generates the collocation nodes and the corresponding weights of order 2 (which is the order q with $2 + 1 = 3$ number of collocation points for each uncertain parameter) with the Gaussian integration[10] (`"G"`) scheme (full tensor product). The propagation of the nodes through the model is the same as in Listing 2.5, only the number and the values of the `nodes` is different. The generation of the orthogonal polynomials (line 9) is similar to

[10]In this thesis, the terms integration and quadrature are used as synonyms.

Listing 2.6. In line 12, the gPCE (Equation (2.7)) is generated via the quadrature rule. To determine the statistical moments of the QoI, the same code (lines 7–10) as in Listing 2.6 can be used. The full UQ stochastic collocation with the pseudo-spectral approach example can be found in Appendix A.2.3.

```python
 1 # creates 2+1 nodes per parameter and corresponding weights according to
 2 # the multivariate distribution and the Gaussian integration scheme
 3 nodes, weights = cp.generate_quadrature(2, dists, rule="G")
 4 
 5 # 2. propagation
 6 solves = [model(p1, p2) for p1, p2 in nodes.T]
 7 
 8 # 3. certification: determine statistics of QoI
 9 OP = cp.orth_ttr(2, dists) # creates orthogonal polynomials according the multivariate
10                            # distribution using three terms recursion
11 
12 gPCE = cp.fit_quadrature(OP, nodes, weights, solves) # generate gPCE
```

Listing 2.7: Partial example of stochastic collocation with the pseudo-spectral approach with Chaospy: Generation of nodes and weights according to the quadrature rule and finally generating the gPCE with the chosen quadrature rule.

Analysis of the three Chaospy code examples

The listings above for the three UQ methods show the basic usage of Chaospy. As already mentioned, the technical phase of the assimilation and the certification is well supported.

But the propagation is very much up to the developer. The propagation is usually done with a simple loop over all `nodes` or with the list comprehension feature of Python. This works well for academic test functions with low computational costs. However, if the model is more complex, then it results in long runtimes on a single CPU core. This can be improved by using threads on a single PC, but if the propagation should be run on a cluster, different techniques are required. How to generally improve the propagation is part of this thesis and is analysed and improved in more detail in Chapter 4, and implemented in Chapter 5.

Note that there are similarities between the three UQ methods, but they are also considerably different. If one wants to change the UQ method, the source code has to be changed. This is addressed in more detail in Section 5.5, where a software solution is proposed, which is designed to switch between UQ methods with fewer code changes.

2.5 Efficiency aspects

Usually, when talking about efficiency, the understanding is that something is fast, the used resources are well used, and less material or time is wasted. However, there exists also the term effectiveness, which is of importance. [74] contains a more fine-grained definition of efficiency vs effectiveness:

Efficiency:

- No wasting of materials, energy, efforts, money, and time
- Ability to do things right: well, successfully, and without waste
- Optimise doing: achieve goals fast and with less effort
- Minimise required resources: computational resources, human resources, time

Effectiveness:

- Doing the right things
- Getting things done
- Accuracy and completeness
- Using the correct methods

This means that the effectiveness—doing the right thing—should be considered before doing it efficiently (doing it fast). Or in other words, the goal is to maximise efficiency by minimising the required resources and fully utilise them.

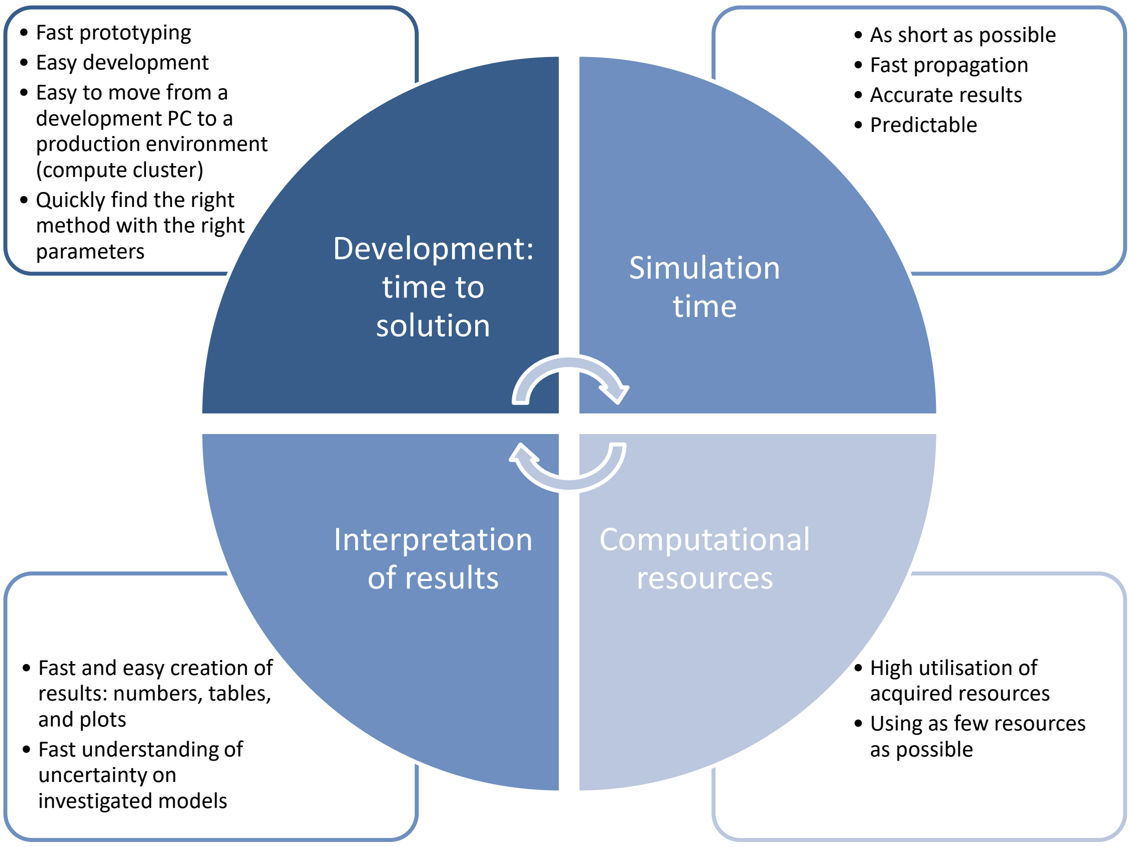

• Fast prototyping
• Easy development
• Easy to move from a development PC to a production environment (compute cluster)
• Quickly find the right method with the right parameters
Development: time to solution
Simulation time
• As short as possible
• Fast propagation
• Accurate results
• Predictable
Interpretation of results
Computational resources
• Fast and easy creation of results: numbers, tables, and plots
• Fast understanding of uncertainty on investigated models
• High utilisation of acquired resources
• Using as few resources as possible

Figure 2.4: Illustration of defined efficiency aspects (containing effectiveness and efficiency) in UQ simulations.

In uncertainty quantification, many aspects of effectiveness and efficiency have to be considered. In the context of this thesis, the efficiency aspects (which contain effectiveness and efficiency) are specified for the following four categories: *development: time to solution*, *simulation time*, *computational resources*, and *interpretation of results*. Figure 2.4 illustrates these efficiency aspects.

1. *Development: time to solution* defines that one wants to prototype and implement the UQ simulation source codes fast and with minimal (easy) development effort. Additionally, it should be easy to move with the UQ simulation from a development PC to a production environment (computer cluster), which usually provides different and much more computing resources. Finding the right UQ method with the right parameters can also be very difficult. Therefore, a change between different methods and their parameters should be easy and with minimal effort.

2. The *simulation time* should be as short as possible and the propagation therefore very fast and with accurate numerical results. Because the UQ simulation can take a lot of time, the runtime should be predictable such that the user knows when the results are available for further investigations.

3. Whether on the development PC or on the production (compute) system, the *computational resources* (CPUs, memory, nodes, ...) should be highly utilised for the simulation

time. Performing a UQ simulation should require as few resources as possible.

4. For the users of UQ simulations, the *interpretation of results* should be easy, i.e. visually seeing the results with numbers, tables, and plots. They should have easy access to this data and be able to repeat the certification phase without the need to repeat the whole propagation phase. A further requirement is that the effects of the uncertainty should be quickly apparent, which means that a broad spectrum of the statistical data of the QoI is provided.

3 Aspects of code development for quantifying the uncertainty in classical simulations

This chapter gives an overview of selected aspects that have to be considered when developing uncertainty quantification simulations for classical simulation scenarios. Some of the information is common knowledge from selected UQ books and research articles, and some information is gathered as practical experience during the research and development in the context of this thesis. These development aspects are additionally considered from the efficiency point of view whose aspects and requirements are defined in Section 2.5.

First, some academic test functions and models are defined in Section 3.1 for presenting and discussing various aspects. Then, Section 3.2 discusses how to select a suitable UQ method and how to find a proper setup of the UQ-method-specific parameters in Section 3.3. This is followed by the uncertain parameter modelling in Section 3.4 and the interpretation of simulation results in Section 3.5 including the representation of uncertainty, the uncertain model parameter analysis via sensitivity indices, the reconstruction of the QoI distribution and the model function via the constructed gPCE.

3.1 Academic test functions and models

Two academic test functions and a simple ordinary differential equation (ODE) are defined and described in the following sections. These functions and the ODE are used for the discussion of the selected development aspects. The two academic test functions will be additionally relevant in Chapter 4.

Example 1: Simple test function

As a simple test function, f_{ex1} is defined as follows

$$f_{ex1} = 2(e^{5\cdot|x|} + \max(y, 0) + 0.2 \cdot |z|)\,. \tag{3.1}$$

The function contains three continuous parameters x, y, and z defined on $\mathbb{R}$, which can be assumed to be uncertain, and different probability distributions can be applied to model the random variables. The value of interest is the result of the $f_{ex1}(x, y, z) \in \mathbb{R}$ function. f_{ex1} is used in this chapter to discuss and demonstrate various aspects of UQ simulations, and in Chapter 4 to serve as a test model for the runtime prediction mechanism proposed in Section 4.5, because it can produce a similar function shape and behaviour of the runtime, but with smaller values as in the runtime and scheduling behaviour results of scenario 2: evacuation of a building with separated families in Section 6.4. This makes it a perfect candidate to develop and test new ideas efficiently.

For the usage of f_{ex1} in this thesis, the parameter support is mostly used in a defined value range, which is documented in Table 3.1.

Parameter	Minimum	Maximum
x	0.1	0.5
y	0.8	1.2
z	1.4	1.8

Table 3.1: Parameters with defined minimum and maximum values for the model function f_{ex1} (Equation (3.1)).

Figure 3.1 shows the resulting 2D function surface of f_{ex1} for the three combination possibilities of the x, y, and z parameters. This gives an overview of the function shape and the expected, resulting values.

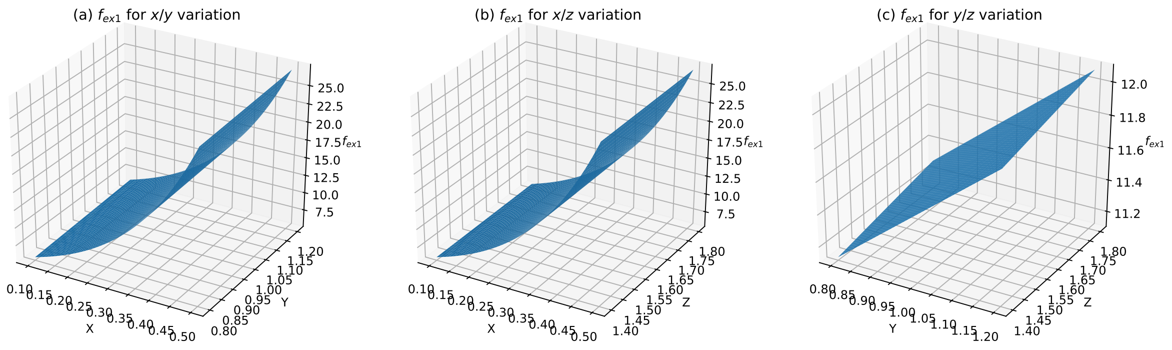


Figure 3.1: Visualisation of the resulting function surface of f_{ex1} (Equation (3.1)) for varying values of the input parameters: (a) contains the resulting f_{ex1} values for the x/y parameter variation, (b) for x/z, and (c) for y/z.

Example 2: Function with discontinuity

As a second academic test function, f_{ex2} is defined as

$$f_{ex2} = e^{-x^2+2\,\mathrm{sign}(y)} + z\,. \tag{3.2}$$

This function is more challenging to use in uncertainty quantification compared to f_{ex1}, because its discontinuity may involve some numerical errors, which makes it a good choice to study the accuracy of various UQ methods. The function is proposed in [49] with the two parameters $x \in \mathtt{R}$ and $y \in \mathtt{R}$, to investigate gPCE based methods in combination with sparse grids collocation schemes. In this thesis, a third parameter $z \in \mathtt{R}$ is introduced to have a more challenging three-dimensional uncertain problem. The value of interest is the result of $f_{ex2} \in \mathtt{R}$. f_{ex2} is used in this chapter to discuss and demonstrate various aspects of UQ simulations, and in Chapter 4 to serve as a test model with a non-smooth behaviour to simulate waiting times for the evaluation of the runtime prediction quality.

The parameter values for f_{ex2} are mainly used within the minimum and maximum values defined in Table 3.2.

Parameter	Minimum	Maximum
x	-2.5	2.5
y	-2.0	2.0
z	5.0	15.0

Table 3.2: Parameters with defined minimum and maximum values for the model function f_{ex2} (Equation (3.2)).

The resulting 2D function shape of f_{ex2} is visible in Figure 3.2. It contains a plot for each of the three combinations of the x, y, and z parameters. As can be seen in the left (a) and the right (c) plot of Figure 3.2, the y parameter has a discontinuity at 0, which makes it hard for polynomials to accurately cover its shape.

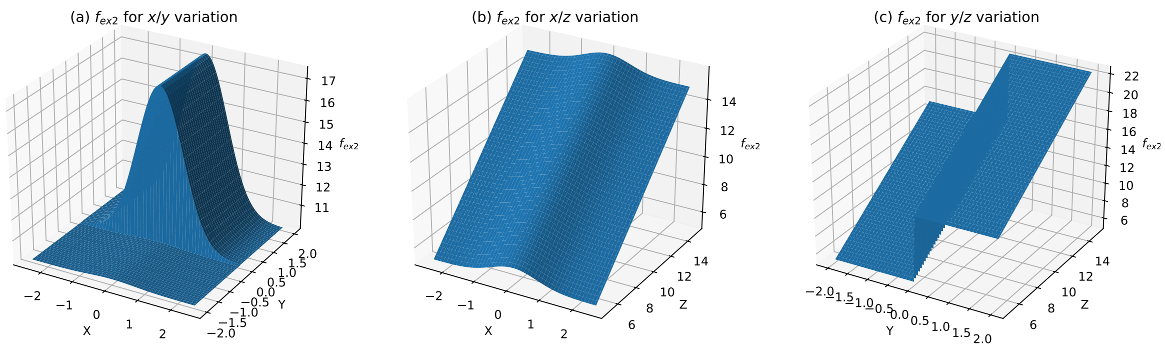


Figure 3.2: Visualisation of the resulting function surface of f_{ex2} (Equation (3.2)) for varying values of the input parameters: (a) contains the resulting f_{ex2} values for the x/y parameter variation, (b) for x/z, and (c) for y/z.

Example 3: Simple ODE

For an academic example of an ordinary differential equation (ODE), a predator-prey population problem of the form

$$\begin{aligned}\dot{C} &= (-\gamma + \delta S)C \\ \dot{S} &= (\alpha - \beta C)S\end{aligned} \tag{3.3}$$

is chosen. The predators are considered as "coyote" and the prey as "sheep". It is a dynamical system with a Lotka-Volterra based model [203], and the following parameters with the used value range are documented in Table 3.3. The six parameters may be assumed to be uncertain and allow to study different aspects of UQ methods in the following sections.

Parameter	Description	Minimum	Maximum
C	population size coyote	30.0	70.0
γ	death rate coyote	0.000,5	0.000,51
δ	augmentation	1.5	3.5
S	population size sheep	1,500.0	2,500.0
α	birth rate sheep	0.005	0.005,09
β	voracity coyote	0.000,001,8	0.000,002,0

Table 3.3: Parameters with defined minimum and maximum values for Equation (3.3) as the model function.

The ODE is considered as an initial value problem and solved numerically using the `scipy.integrate.ode.solve_ivp`[1] function, which is configured to use a backward differentiation formula (BDF) to solve Equation (3.3) iteratively. The time step size is set to 0.01, and the simulation stops after $70 * 365 = 25{,}550$ time units, which can be interpreted as a simulation of 70 years. Equation (3.3) is used in this chapter to discuss and demonstrate various aspects of UQ simulations. Because the ODE is iteratively solved and the population size data of the sheep

[1] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.ode.html`

and the coyote are available for every time step, this offers additional analysis and visualisation possibilities for UQ simulations.

Figure 3.3 shows the resulting values of the dynamical system for a fixed set of the parameter values: The population size of the sheep S are plotted in Figure 3.3(a) and the population size of the coyote C in Figure 3.3(b). It nicely shows the dynamics of the system and the alternating growth and shrinkage of both populations.

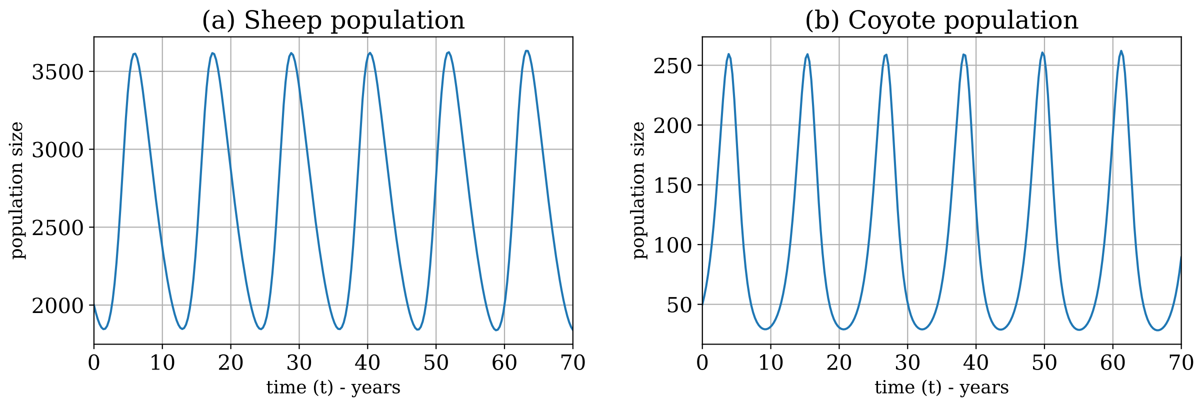


Figure 3.3: Visualisation of the resulting population sizes of the predator-prey population problem of Equation (3.3). Figure 3.3(a) shows the population size S of the sheep, and Figure 3.3(b) the population size C of the coyote. The initial parameters values used for the simulation are: $C = 50$, $\gamma = 0.0005$, $\delta = 2.5$, $S = 2000$, $\alpha = 0.005$, and $\beta = 0.0000019$.

A more general overview of the possible results of Equation (3.3) within the defined parameter value ranges of Table 3.3 is given by the attractors in Figure 3.4. It shows the possible population values in relation to the two populations of coyote and sheep and gives a first impression of the impact on the uncertainty of the six parameters. It can be seen that β, S, C, and δ, have a lot of impact on the population sizes, while γ and α do have little.

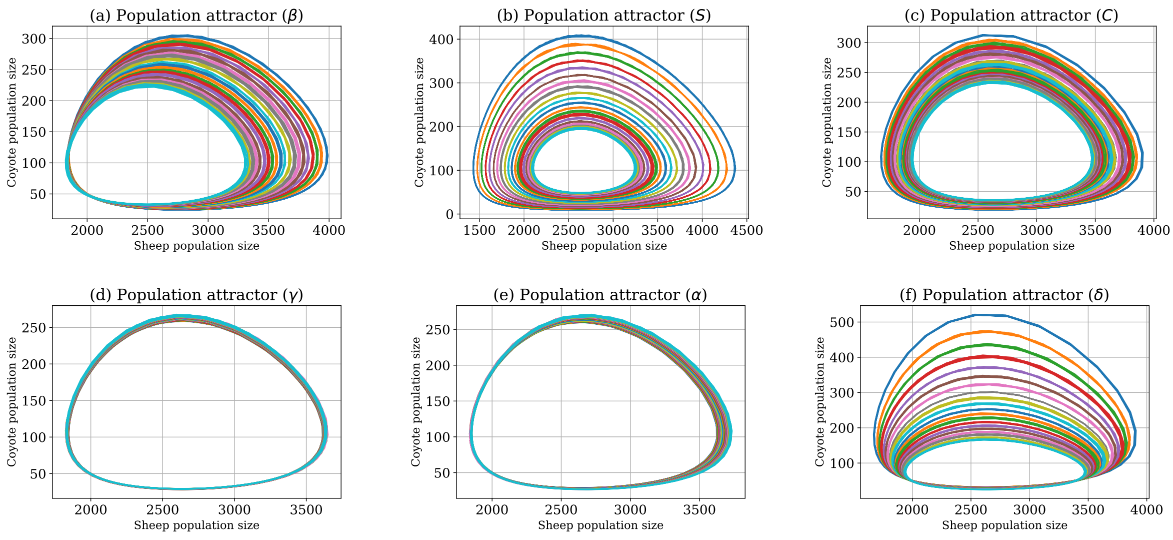


Figure 3.4: Visualisation of the attractors for the resulting values of Equation (3.3) in relation of the coyote C and the sheep S population sizes. For each parameter in Table 3.3, the attractor is plotted to see the individual impact of the parameter to the resulting population sizes and the dynamics of the system.

3.2 Selecting a suitable UQ method

In the standard literature [123, 178, 186] for UQ, a basic description of the various UQ methods and its properties is present. But there is no general guideline documented on how to choose a suitable UQ method for the underlying problem. UQ is a young field that is actively developed and in the focus of today's research, which constantly presents many improvements in different directions and UQ methods. In [53], the authors could not even find a common notation to define a common thread throughout the different contributions, which shows the diversity and the huge field of the different UQ methods and possible fields of application.

In this section, also, no general guideline on how to select a proper UQ method for a given problem is defined to avoid the dilemma mentioned above. Instead, several aspects are discussed in the following, which play a significant role when choosing a UQ method. Primarily, the described non-intrusive forward UQ methods in Section 2.2 are used for the discussion with some citations and side nodes for other methods or appropriate improvements.

Number of uncertain input parameters

The number M of uncertain parameters $\boldsymbol{\zeta}$ does play a significant role to apply gPCE based UQ methods. For the stochastic collocation with the pseudo-spectral approach (Section 2.2.2), the full tensor grid with $Q = q^M$ collocation points is required to solve the coefficients c_j (Equation (2.14)) of the gPCE $u_N(x, t, \boldsymbol{\zeta})$ (Equation (2.7)). For the point collocation method (Section 2.2.3), the number of uncertain parameters influences the $N + 1$ (Equation (2.8)) number of expansion terms to solve Equation (2.20), which requires $\geq (N+1)$ model evaluations. Depending on the runtime of the model, gPCE based methods are feasible for low (1–3) to mid (4–12) number of uncertain parameters M. In combination with different techniques such as sparse grids [34], M can be higher but do reach a certain limit.

If a high number of uncertain parameters M is given, then Monte Carlo (Section 2.2.1) based methods are a good choice, because their convergence does not depend on the number of uncertain parameters M.

Numerical stability of QoI (smoothness)

To achieve a certain level of accuracy for the QoI, proper settings for the UQ methods have to be found. This depends strongly on the shape and the smoothness of the model function whose uncertainty should be quantified. For non-smooth functions, gPCE based methods are not the first choice because the polynomials cannot cover discontinuities or steep gradients well [66]. With a high number of collocation points per parameter q, the accuracy can be improved, but not perfectly. Improvements, such as adaptive methods [200, 38, 40] may then be a good choice.

It also depends on the required statistical moments for the QoI. For the mean μ, fewer model evaluations are usually required for gPCE based methods as well as for Monte Carlo based methods. For higher statistical moments, such as the variance σ^2 or the percentiles, more information and therefore more model evaluations are required. Here, the Monte Carlo based methods suffer from the slow convergence rate, and the gPCE based methods have an advantage.

Representation of the QoI

The required representation for the QoI does also play a significant role in the choice of the UQ method. If only the statistical moments such as the mean μ or the variance σ^2 are required, sampling methods such as Monte Carlo are also suitable. If the whole solution $U(x, t, \boldsymbol{\zeta})$ of the model function is required, then gPCE based methods can be used, because they can provide a solution of the model under uncertainty in form of the gPCE expansion, e.g. $u_N(x, t, \boldsymbol{\zeta})$ in Equation (2.7).

If for a Monte Carlo simulation the propagation through the model has already been performed, then it is possible via the point collocation method Section 2.2.3 to create the gPCE afterwards, which allows generating the gPCE with a certain accuracy.

Runtime of a model and available computational power

The available computational power does also play a significant role in the choice of the UQ method when performing the UQ simulations. The limiting factor is usually the runtime of a single black-box model evaluation for non-intrusive forward UQ problems. Hence, UQ methods with fewer model evaluations such as the gPCE based methods do have an advantage over the sampling-based methods.

Because of the long runtimes of the model, it is often required to perform the model evaluations in parallel on large and distributed computing centres such as HPC systems. The advantage of the non-intrusive UQ methods is that the parallelisation of the individual black-box model runs can be embarrassingly parallel. As the analysis in Section 4.4.2 shows, here, a lot of potential for improvement is available, which is the topic of Section 4.6. Case studies for the improved scheduling can be found in Chapters 6 and 7 as well as in [103].

If the results for a UQ simulation are required for real-time decision-making, it is often not feasible to use the model directly for the propagation of the uncertainties. In such cases, it is possible to use a surrogate model representation with faster evaluations instead, as proposed in Section 4.7. The surrogate model contains only a subset of the information of the original model and introduces an additional source of errors. A case study can be found in this thesis in Section 6.5 and in [28].

Stochastically dependent uncertain input parameters

Usually, stochastically independent random variables are assumed as the uncertain parameters for UQ problems. The gCPE based methods profit heavily from this assumption in the small number of required model evaluations compared to sampling-based methods. If stochastic dependent input parameters are given for a UQ problem, Monte Carlo based methods can be used. In [42], the authors propose a new approach for handling dependent input variables for gPCE based methods by transforming the input parameters and decorrelating the orthogonal basis, even for multivariate probability distributions.

When dealing with UQ simulations with a large-scale simulation scenario, it is an advantage if it is easy to switch between the UQ methods without changing a lot of source code. Especially if the proper UQ method is not yet known, or a test run with a different UQ method should be performed for comparison purposes. The UQEF framework does support different UQ methods (see Section 5.5), and no source code changes are necessary to change the UQ method. Additionally, its software design allows to develop and integrate new UQ methods easily.

Table 3.4 summarises the discussed aspects and gives a coarse overview of a possible situation and the aptitude of the discussed UQ methods, which are also available in the UQEF framework.

Stochastic dimensionality		**Smoothness of model**		**QoI represantation**		**Model runtime**		**Parameter dependency**		**UQ method**	
low/mid	high	smooth	non-smooth	statistical moments	full solution of $U(x,t,\boldsymbol{\zeta})$	short	long	independent	dependent	**MC**	**gPCE**
x		x		x		x		x		1	1
x		x		x		x			x	1	1
x		x		x			x		x	3	1
x		x		x			x	x		3	1
x		x			x	x		x		$2^{*)}$	1
x		x			x	x			x	$2^{*)}$	1
x		x			x		x	x		3	1
x		x			x		x		x	3	1
x			x	x		x		x		1	2
x			x	x		x			x	1	2
x			x	x			x	x		3	2
x			x	x			x		x	3	2
x			x		x	x		x		$2^{*)}$	2
x			x		x	x			x	$2^{*)}$	2
x			x		x		x	x		3	2
x			x		x		x		x	3	2
	x	x		x		x		x		1	3
	x	x		x		x			x	1	3
	x	x		x			x	x		3	3
	x	x		x			x		x	3	3
	x	x			x	x		x		$2^{*)}$	3
	x	x			x	x			x	$2^{*)}$	3
	x	x			x		x	x		3	3
	x	x			x		x		x	3	3
	x		x	x		x		x		1	3
	x		x	x		x			x	1	3
	x		x	x			x	x		3	3
	x		x	x			x		x	3	3
	x		x		x	x		x		$2^{*)}$	3
	x		x		x	x			x	$2^{*)}$	3
	x		x		x		x	x		3	3
	x		x		x		x		x	3	3

Table 3.4: Choice of UQ method depending on various aspects. The Monte Carlo based methods (MC) and the gPCE based methods are categorised with: 1 = good, 2 = possible with extensions—but somehow limited, 3 = not good. $^{*)}$ Denotes Monte Carlo with the extension of the point collocation method.

3.3 Parameter selection for UQ simulations

Once a proper UQ method is chosen, e.g. with considering the discussed aspects of Section 3.2 and using Table 3.4, then further method-specific details have to be specified.

For Monte Carlo based methods, the number of samples $\mathcal{M}$ that should be used for the propagation have to be determined. There is no general rule, but usually, a large $\mathcal{M}$ is needed because of the slow convergence rate, depending on the smoothness of the model function f and the required statistical moments. In [110, 133], there are methods and hints on how to determine the number of required samples $\mathcal{M}$ for Monte Carlo based methods.

For the stochastic collocation with the pseudo-spectral approach, the number of $N+1$ terms (Equation (2.8)) of the expansion depending on the number M of uncertain parameters $\boldsymbol{\zeta} = (\zeta_1, \dots, \zeta_M)$ and the highest order P of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$, as well as the number of collocation points Q (Equation (2.14)) to compute the coefficients c_j in Equation (2.14) have to be defined. Since the Gaussian quadrature is used in Equation (2.14) with $Q = q^M$ collocation points z_i, a polynomial representing the model function f with a degree of up to $2Q+1$ can be exactly approximated [212]. Because the polynomial degree of f is often unknown (or the model does not have a polynomial representation) for large-scale scenarios, it is hard to determine Q and therefore also the number of collocation points q per uncertain parameter ζ. In [34], the connection between the highest order P of the orthogonal polynomials and the q number of collocation points q per uncertain parameter suggests $q = 2P + 1$, which results in $Q = q^M = (2P+1)^M$ total number of collocation points.

Another way of determining P and q for large-scale simulation scenarios is a parameter study. The values of P and q are iteratively increased until the mean and the variance are stabilising. For a scalar QoI this means that the changes of the values (between the increasing steps of P and q) for the mean μ or the variance σ^2 does not exceed a certain threshold. If the QoI is a time series, then the mean and especially the variance may be oscillating heavily between subsequent points in time if P and q are not properly set. After enough information are available, and P and q are high enough and do fit properly together, the values are going to stabilise and do not oscillate that much. Because the μ can be obtained by only determining c_0 according to Equation (2.17), and for the variance σ^2 also the other coefficients c_j, $j > 1, \dots, N$ are required (Equation (2.18)), the variance σ^2 is more fragile than the mean μ. The developed UQEF software framework does support such parameters studies through its outer-loop support. Details can be found in Section 5.7.

In the context of this work, it is shown that q should always satisfy $q \geq P$. For $q \leq 6$, good results have been achieved with $q = P$, and for $q > 6$ it showed that P should be fixed at $P = 6$ or 7 for $M = 1, \dots, 4$ number of uncertain parameters. However, this depends very much on the underlying model function f and is therefore hard to generalise.

3.4 Uncertain parameter modelling

The identification and the modelling of the uncertain parameters of a model is a very important part of the assimilation phase in UQ. If the distribution of the parameters is well known and do follow one of the well known probability distributions (e.g. uniform, Gaussian, ..., c.f. Table 2.2), then the whole assimilation phase is easy to handle.

For complex, large-scale simulation scenarios, this is often not exactly known [123] and the uncertain input parameters of the model have to be analysed first, to find proper approaches to model them as uncertain variables. Here, the whole field of parameter estimation [97, 10] can be used.

In this thesis, two further possibilities are taken into account: The "interview with domain experts" and the "automatic measurement-driven distribution generation", which are briefly described in the following.

Interview with domain experts

For certain fields, such as the field of pedestrian dynamics (Chapter 6), where behavioural parameters of humans are taken into account, often no or not many experiments can be performed to quantitatively determine the setup for the uncertain parameters. In such cases, interviews with domain experts may help to determine valid lower and upper bounds for certain parameters, or to find valid probability distributions with its specific setup. For the pedestrian dynamics scenarios in Section 6.3 and Section 6.4, the probability distributions for the uncertain parameters have been determined through interviews with psychologists who scientifically investigate human behaviour in specific situations (e.g. in emergency situations [175, 174, 30, 32, 31]), which was very helpful to find a proper setup for the UQ simulations.

Automatic measurement-driven distribution generation

Another way—especially in physical systems—to find proper probability distributions is to observe the possible values for the input parameters. Once, a set of data $\boldsymbol{\chi} = \{\chi_1, \chi_2, \dots, \chi_{N_\chi}\}$ with N_χ values could be observed for an uncertain parameter ζ, it can be processed by automatic measurement-driven distribution generation to obtain a probability distribution that can be used for uncertainty quantification.

For the automatic measurement-driven process, kernel density estimation (KDE) [57] is used. It estimates the probability density function (PDF) of a random variable based on observed values $\boldsymbol{\chi}$. In a univariate KDE driven process, the PDF can be approximated as follows

$$\mathrm{PDF}_h(x) = \frac{1}{N_\chi} \sum_{i=1}^{N_\chi} K_h(x - \chi_i) = \frac{1}{N_\chi h} \sum_{i=1}^{N_\chi} K\Big(\frac{x - \chi_i}{h}\Big) \tag{3.4}$$

using a kernel K and a smoothing operator h, which is also known as the bandwidth. It is important to choose the bandwidth h properly to obtain good results for the density. Here, the Scott-Silverman [164, 169] rule of thumb

$$h = 1.06 \min\left\{\sigma, \frac{\mathrm{IQR}}{1.34}\right\} N_\chi^{-1/5} \tag{3.5}$$

is used, where IQR denotes the interquartile range, which contains 50 percent of the values around the mean μ. Several types of kernels K exist [57]. In this thesis, a Gaussian kernel in the following form

$$K(x) := \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \tag{3.6}$$

is used. In [35], it is shown that the choice of the kernel does not have a strong effect on the resulting error. Obviously, Gaussian kernels are well suited to estimate Gaussian distributed data and may not be the best choice, e.g. for uniformly distributed data.

The KDE based constructed PDF of Equation (3.4) is then used in the assimilation phase of a UQ simulation to generate the nodes n_i on which the model function f is sampled in the propagation phase. It can be used for both, Monte Carlo based UQ simulations as well as for gPCE based method like the stochastic collocation with the pseudo-spectral approach (Section 2.2.2) as presented in [35].

In the context of this work, a KDE based distribution called *sample_dist* with its construction function *SampleDist* has been implemented and merged into the Chaospy library. *SampleDist* creates an object of the *sample_dist* distribution which is derived from the Chaospy *Dist* class and is therefore fully integrated into Chaospy with the same functionality as the other provided probability distributions. It uses the Gaussian kernel (Equation (3.6)) implementation (`gaussian_kde`) from the `scipy.stats` Python package to construct the kernel based on the given observations $\boldsymbol{\chi}$. The KDE is configured to use the described Scott-Silverman rule of thumb in Equation (3.5) to determine the bandwidth h automatically.

Listing 3.1 shows an example of how to use the *SampleDist* function to construct a probability distribution. For that, some "observed" data ($\boldsymbol{\chi}$) is simulated by drawing them from a $N(0,1)$ probability distribution (lines 6–7). In line 10, the probability distribution `sample_dist` is created using the *SampleDist* function. It requires only the observed data `observed_samples` to automatically construct the probability distribution. The distribution object `sample_dist` can then be used as a regular Chaospy distribution: e.g. for drawing samples as shown in line 11, or to generate nodes and weights to perform a gPCE based UQ simulation as explained in Listing 2.7.

```python
1  import chaospy as cp
2
3  num_samples = 10000
4
5  # simulate observed values for an uncertain parameter
6  example_dist = cp.Normal(0, 1)
7  observed_samples = example_dist.sample(num_samples)
8
9  # automatic measurement-driven distribution generation with SampleDist
10 sample_dist = cp.SampleDist(observed_samples)
11 generated_samples = sample_dist.sample(num_samples)
```

Listing 3.1: Example of creating the KDE based *sample_dist* distribution using the *SampleDist* function of the Chaospy library.

Figure 3.5 contains the histogram and the PDF of the observed (given) data compared to the sampled (generated) data with *SampleDist*. It can be seen that they qualitatively have a similar shape. Figure 3.6 further shows the mean μ, the standard deviation σ, as well as the p_5 and p_{95} percentile of both, the observed (given) distribution in Figure 3.6(a) and the sampled (generated) distribution in Figure 3.6(b). Again, the shape of the values looks qualitatively similar.

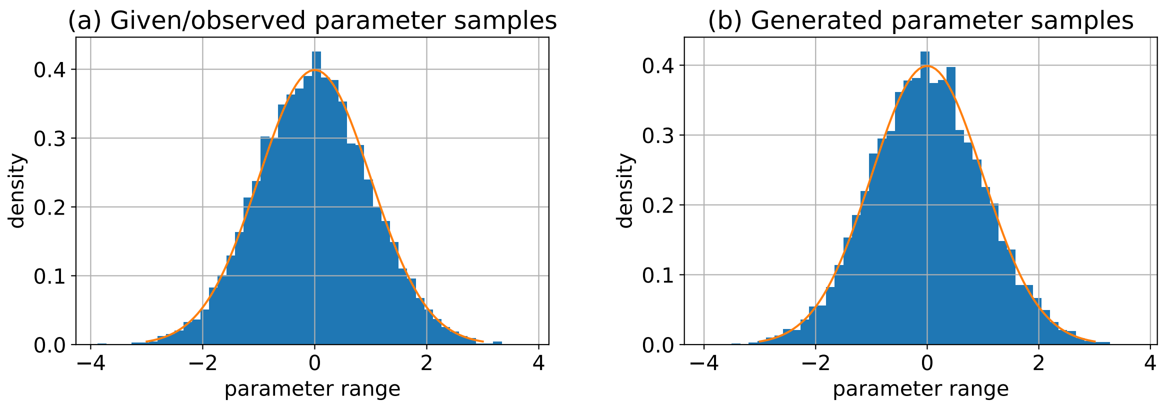


Figure 3.5: Histogram of the observed (given) data in (a) for a given $N(0,1)$ distributed parameter and the sampled (generated) data in (b). For the generation of the data, a sample size of 10,000 and the *SampleDist* function based on KDE is used.

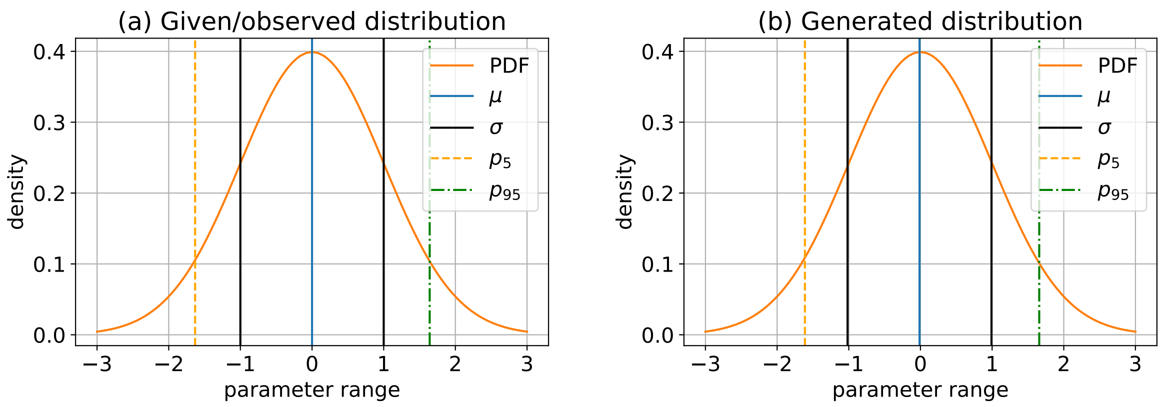


Figure 3.6: Probability density function and statistical moments (mean μ, standard deviation σ, and p_5 and p_{95} percentiles) for the observed (generated) probability distribution in (a) for a given $N(0,1)$ distributed parameter, and the sampled (generated) probability distribution in (b). For the generation of the data, a sample size of 10,000 and the *SampleDist* function based on KDE is used.

The probability distributions are further analysed using different sample sizes (100, 500, 1,000, 5,000, 10,000, 50,000, and 100,000) and comparing the statistical moments of the observed (given) distribution to the sampled (generated) distribution. The errors are determined by taking the absolute values of the differences for each statistical moment: e.g. the error ϵ_μ for the mean μ is determined with $\epsilon_\mu = |\mu_{\text{observed}} - \mu_{\text{generated}}|$. The results are listed in Table 3.5. As expected, with an increasing sample size, the errors are going to be smaller. Hence, the more observed data are available for an uncertain parameter, the better the generated probability distribution.

	Number of samples						
	100	500	1,000	5,000	10,000	50,000	100,000
ϵ_μ	0.131	0.048	0.008	0.009	0.020	0.001	0.002
ϵ_σ	0.073	0.038	0.015	0.027	0.019	0.008	0.003
ϵ_{p_5}	0.303	0.128	0.063	0.019	0.011	0.011	0.004
$\epsilon_{p_{95}}$	0.239	0.037	0.069	0.035	0.012	0.001	0.012

Table 3.5: Error comparison of measurement-driven generated probability distribution for various statistical moments compared to the original used $N(0,1)$ distributed probability distribution for different sample sizes.

The kernel density estimation technique is also available for multivariate random variables, as described in [165]. In this thesis, only the univariate case is considered, because Chaospy easily offers to create multivariate probability distributions with its join operator `J` (c.f. Listing 2.4) given multiple univariate probability distributions.

The *SampleDist* functionality is further used in Section 3.5, which offers additional use cases.

3.5 Interpretation of simulation results

In computational science and especially in UQ, it is very important to present the results of simulations in such a way that the users can easily understand the numbers and use them for decision-making. A general introduction to the visualisation of data can be found in [13, 89, 207].

In Section 2.5, the interpretation of the results of a UQ simulation has also been identified as an important aspect to understand the influence of the uncertainty efficiently. The different aspects are discussed separately in: representation of uncertainty, uncertain model parameter analysis via sensitivity indices, reconstruction of the QoI distribution, and reconstruction of the model function via the constructed gPCE.

Representation of uncertainty

The representation of uncertainty is also an own sub-area of current research [65, 88, 84, 11, 68, 70, 83, 44] in the broad field of UQ.

In many UQ related research articles, often only the resulting mean μ and the variance σ^2 of the last time step of the model is calculated. The case studies in Chapters 6 and 7 consider dynamical models that evolve over time. For these models, representing the QoI with the mean μ and the variance σ^2 or standard deviation σ give useful information—but how the uncertainty evolves cannot be answered.

In this thesis, it is proposed to consider the OoI for the model time steps from the beginning of the simulation until the end, and calculating the QoI for every time step. This gives additional valuable information because it shows how the uncertainty evolves in the model during the simulation. For that, the OoI has to be extracted for every time step, and the QoI also needs to be calculated for these time steps. This requires additional computational effort for the certification phase of a UQ simulation. For the propagation phase, only the extraction of the VoI for every time step can be interpreted as additional effort, the number of required model evaluations do not change.

The visualisation of the QoI is very domain-specific, which means that it depends strongly on the model and its OoI. Often, the QoI values have to be mapped back into the model's domain (e.g. into the map) to give useful information. Therefore, it is not possible to define a "standard" visualisation scheme; every model has to be considered individually.

As an example, Equation (3.3) is used as the model to demonstrate the QoI representation for every time step. All six parameters of Table 3.3 are considered to be uncertain. The defined probability distributions are listed in Table 3.6. Due to demonstration purposes, the probability distributions and their parameters are chosen based on some practical experience and do not represent a specific scenario, but it is well suited for showing the time stepwise OoI extraction and QoI representation.

Parameter	Description	Distribution
C	population size coyote	N(50, 2)
γ	death rate coyote	U(0.0005, 0.00051)
δ	augmentation	N(2.5, 0.1)
S	population size sheep	N(2000, 10)
α	birth rate sheep	U(0.00501, 0.00509)
β	voracity coyote	U(0.0000018, 0.0000020)

Table 3.6: Defined uncertain parameters for the simple ODE Equation (3.3) as the model function with associated probability distributions.

For the example UQ simulation, stochastic collocation with the pseudo-spectral approach is chosen. As the highest order of the orthogonal polynomials $P = 2$ is set, and the number of collocation points for each parameter is $q = 6$, which results in $Q = 6^6 = 117{,}649$ number of total collocation points.

Table 3.7 lists the QoI statistics for the population size of the sheep and the coyote for the last step of the simulation. With the values of the statistical moments, it can be seen that the

	Sheep	Coyote
μ	1,901.4	119.2
σ	65.5	37.7
σ^2	4,294.3	1,427.0
p_5	1,796.0	58.9
p_{95}	2,007.5	179.3

Table 3.7: QoI statistics of Equation (3.3) for the last time step of the simulation for the population size of the sheep and the coyote.

uncertain parameters influence the population size. But only looking at QoI of the last time step, does not give an impression of the model behaviour under the given uncertainty.

With the proposed way of calculating the QoI for every time step and plotting the data properly, the QoI for the population sizes gives the impression of the uncertain influence: Figure 3.7(a) shows the mean population size $\mu(S)$ with the p_5 and p_{95} percentiles. The corresponding standard deviation σ is plotted in Figure 3.7(b). It can be seen that the most uncertainty is in

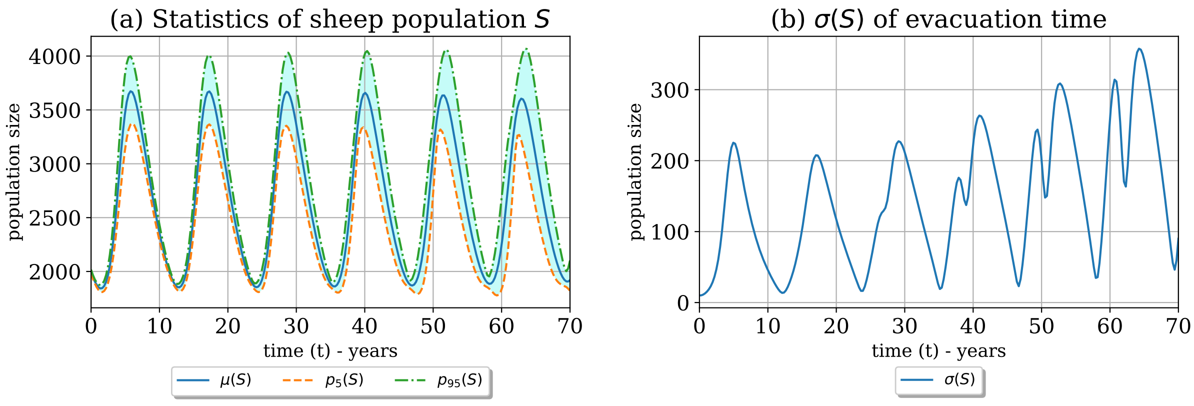


Figure 3.7: QoI results for the sheep population size S for Equation (3.3) with all six parameters of Table 3.6 as uncertain input. (a) shows the mean $\mu(S)$ and the percentiles ($p_5(S)$ and $p_{95}(S)$) for each time step; (b) shows the corresponding standard deviation $\sigma(S)$ for every time step.

the growth cycles of the sheep, while in the down cycles (shrink of sheep population), the uncertainty is much smaller. During the time, the standard deviation σ is increasing. The QoI for the population size of the coyote given in Figure 3.8 shows a similar behaviour, with more uncertainty in the growth cycle and an increasing standard deviation $\sigma(C)$ over time.

This example shows that with the generation of the QoI for every time step and a proper visualisation and representation of the QoI values, it is possible to efficiently understand the influence of the uncertainty for a model. It allows to gain additional insights, which is not possible by only considering the QoI of the last time step of a simulation.

Uncertain model parameter analysis via sensitivity indices

After having a first impression of the influence of the uncertainty during the QoI statistics above, the next question arises: Which parameter has the most influence on the uncertainty, and is the influence the same for the whole simulation time? To answer that, a global sensitivity analysis with the gPCE $u_N(x, t, \boldsymbol{\zeta})$ of Equation (2.7) as the surrogate has been performed on top of the UQ analysis as described in Section 2.3.

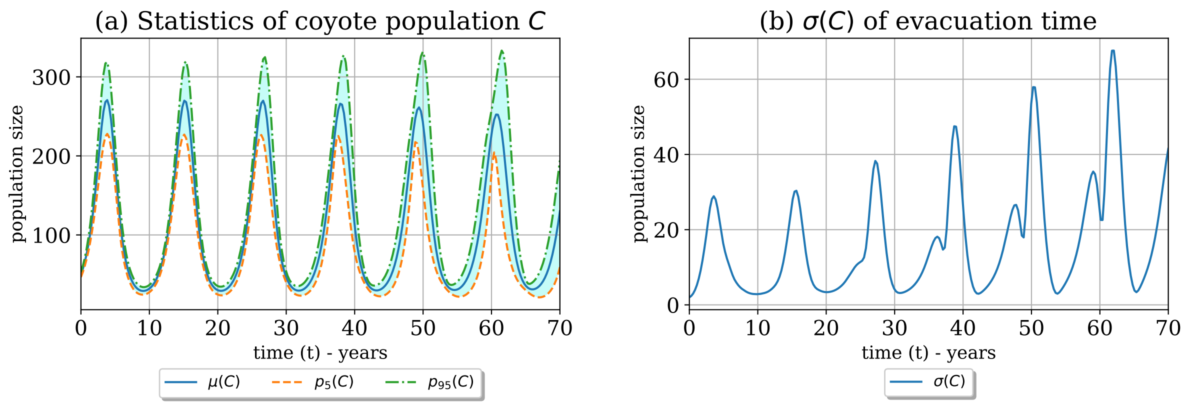


Figure 3.8: QoI results for the sheep population size C for Equation (3.3) with all six parameters of Table 3.6 as uncertain input. (a) shows the mean $\mu(C)$ and the percentiles ($p_5(C)$ and $p_{95}(C)$) for each time step; (b) shows the corresponding standard deviation $\sigma(S)$ for every time step.

Figure 3.9(a) shows the first-order sensitivity indices for the population size of the sheep. It can be seen that the voracity β of the coyote contributes most to the VoI. The second most influence have the parameters C (initial population of coyote) and δ (augmentation rate). It can be seen that in the shrinkage phase of the sheep, β becomes less relevant while C and δ contribute more to the VoI. In the start of the shrinkage phase of the sheep, S and γ contribute to the VoI, and α contributes in the growth phase of the sheep. On si_int, it becomes visible that on the shrinkage phase, there is some interaction between the parameters. The total-order sensitivity indices in Figure 3.9(b) show that β contributes most, followed by C, δ, and the others.

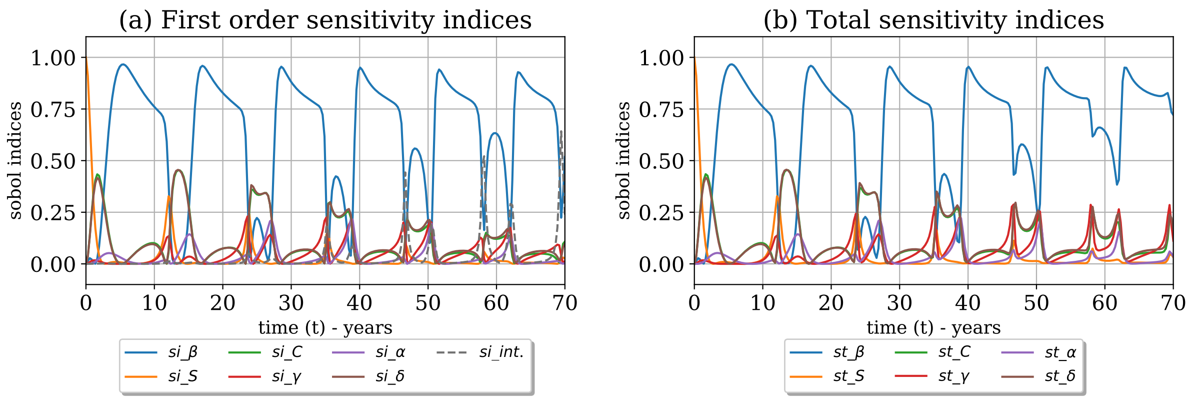


Figure 3.9: Sensitivity indices for the population size of the sheep with all of the six uncertain parameters (Table 3.6) for Equation (3.3). (a) shows the first-order sensitivity indices and (b) shows the corresponding total-order sensitivity indices.

The sensitivity indices for the population size of the coyote can be found in Figure 3.10. It can be seen that β has the most influence again, but oscillates much more than for the population size for the sheep. The parameters C and δ do have a lot of influence in the first cycles but then do have little until the end of the simulation time. γ, α, do contribute little at the beginning, but do then contribute more in the shrink and grow cycles. The initial population size S of the sheep do have less influence on the VoI during the whole simulation time. The interaction

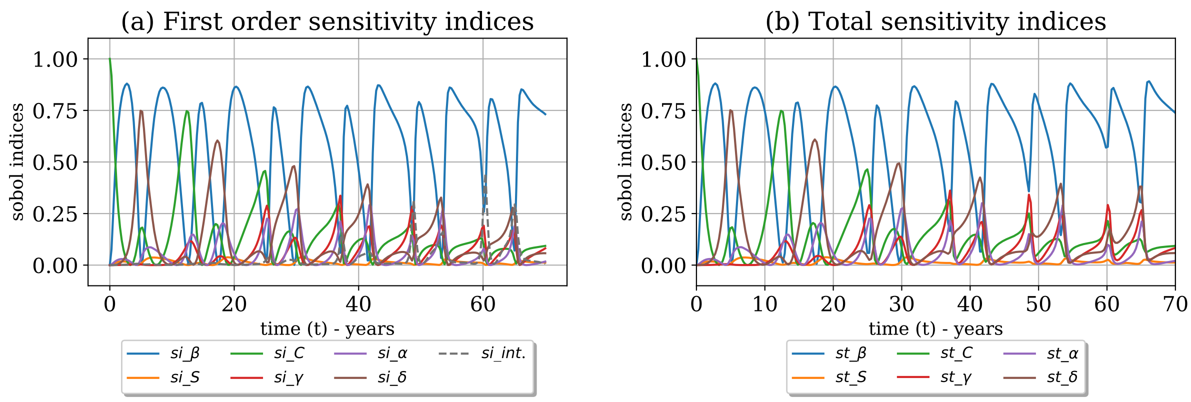


Figure 3.10: Sensitivity indices for the population size of the coyote with all of the six uncertain parameters (Table 3.6) for Equation (3.3). (a) shows the first-order sensitivity indices and (b) shows the corresponding total-order sensitivity indices.

between the parameters is less in the beginning and do increase over time for some cycles.

If it is of interest which parameters do interact with each other, then the higher-order sensitivity indices of Equation (2.27) can be computed to answer that. The sensitivity indices do greatly answer the amount of contribution, but they do not give the information why. For this, the equations of the model have to be analysed.

Here, it is also important to not only calculate the sensitivity indices for the last time step, because in such dynamical systems as in the Lotka-Volterra based model of Equation (3.3), additional knowledge about the uncertain parameter behaviour can be obtained by studying the sensitivity indices for the whole time series. With such a technique, the questions above, which uncertain parameters contribute most, and does the contribution change during the simulation time, can be answered.

An additional use case arises with the knowledge of the contribution of the uncertain parameters: After identifying the most important parameters with a low number of collocation points q for each parameter, the simulation can be repeated with only these parameters, but with higher q, to obtain better results with less computational effort, compared to the use of all uncertain parameters. This technique is stated as stochastic dimensionality reduction (or two-step strategy) [183], and can also be used in more advanced reduction techniques such as in an adaptive multi-level UQ as described in [39, 38].

Reconstruction of the QoI distribution

The resulting QoI of a UQ analysis is usually expressed with some statistical moments such as the mean μ and the variance σ^2. This gives already very useful information, especially if the statistical moments are determined for all time steps of a dynamical model, as already discussed above. But sometimes, a probability distribution of the OoI should be the result for the QoI. This construction of the probability distribution as the QoI is discussed in the following for Monte Carlo based methods as well as for gPCE based methods.

For Monte Carlo based methods, the *SampleDist* functionality (introduced above), can be used. All gathered VoIs can be given as observations into *SampleDist* (cf. Listing 3.1), which generates a KDE based probability distribution.

For gPCE based methods, it is more complicated because the information is contained in the coefficients c_j in combination with the orthogonal polynomial $\Phi_j(\boldsymbol{\zeta})$. Right now, there is no method known by the author of constructing the probability distribution directly. Instead, the constructed $u_N(x, t, \boldsymbol{\zeta})$ is used as a surrogate model for a secondary UQ simulation based on

Monte Carlo samples. For that, the same probability distributions are used as for constructing $u_N(x, t, \boldsymbol{\zeta})$. The results of the secondary Monte Carlo simulation are then the observed VoIs, which can then be used with a KDE to create a probability distribution.

A function called *QoI_Dist* is implemented in Chaospy, which takes a gPCE object, a (multivariate) distribution object and the number of samples for the secondary Monte Carlo simulation as input. The result is a probability distribution, more concrete, a *sample_dist* object. The function performs an internal Monte Carlo simulation with the given number of samples using the implemented *SampleDist* function to construct a KDE based probability distribution object of the *sample_dist* class. As part of this work, this implementation is upstreamed into the Chaospy library.

Listing 3.2 shows a usage example of the *QoI_Dist* function. It is assumed that the (multivariate) distribution and the gPCE have already been constructed with either the pseudo-spectral approach (Section 2.2.2) or with point-collocation (Section 2.2.3). Then, in line 4, the *QoI_Dist* function is called with the gPCE and the distribution object as well as with the number of samples for the Monte Carlo simulation, here $\mathcal{M} = 10{,}000$. The result is a `qoi_dist` object of the *sample_dist* class, which can then be used as any other Chaospy distribution.

```
1 #dist = ...
2 #gPCE = cp.fit_quadrature(...)
3
4 qoi_dist = cp.QoI_Dist(gPCE, dist, 10000)
```

Listing 3.2: Example of creating the QoI distribution based on *sample_dist* using the *QoI_Dist* function of the Chaospy library.

As an example, the *QoI_Dist* function is used together with the academic test functions f_{ex1} (Equation (3.1)) and f_{ex2} (Equation (3.2)). The uncertain parameters with their defined probability distribution are defined in Tables 3.8 and 3.9 for f_{ex1} and f_{ex2}, respectively.

Parameter	Distribution
x	$U(0.1, 0.5)$
y	$U(0.8, 1.2)$
z	$U(1.4, 1.8)$

Table 3.8: Defined uncertain parameters for model function f_{ex1} (Equation (3.1)) with associated probability distributions.

Parameter	Distribution
x	$U(-2.5, 2.5)$
y	$U(-2.0, 2.0)$
z	$U(5.0, 15.0)$

Table 3.9: Defined uncertain parameters for model function f_{ex2} (Equation (3.2)) with associated probability distributions.

For each academic test function, a UQ simulation using the stochastic collocation with the pseudo-spectral approach has been performed with $q = 6$ and $P = 2$. For the secondary Monte Carlo simulation, $\mathcal{M} = 10{,}000$ individual random samples are used. The resulting probability distributions with the statistical moments for both academic test functions are visible in Figure 3.11. Considering the PDF, this gives additional insights about the uncertainty in the OoI.

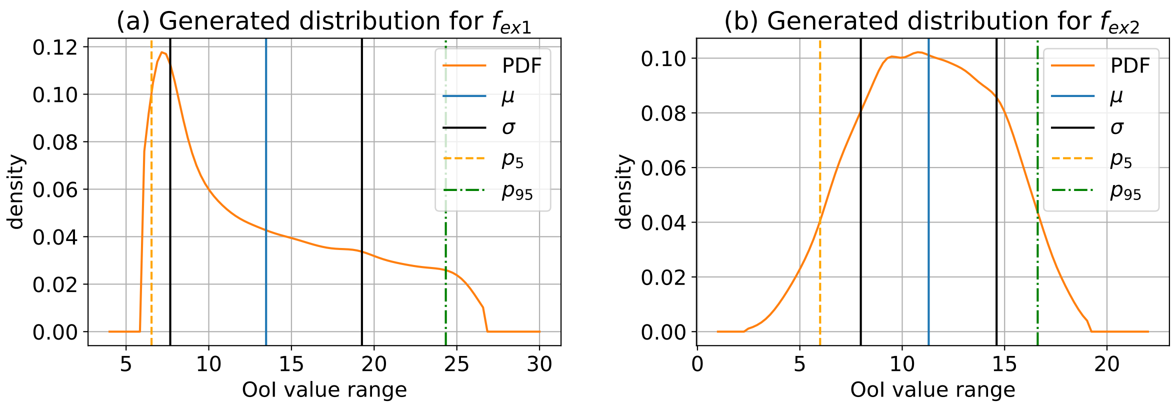


Figure 3.11: Probability density function and statistical moments (mean μ, standard deviation σ, and p_5 and p_{95} percentiles) for the probability distribution of the QoI. (a) shows the results for the QoI distribution of f_{ex1} (Equation (3.1)), and (b) of f_{ex2} (Equation (3.2)). For the generation of the data, the *QoI_Dist* function is used with a secondary Monte Carlo simulation using $\mathcal{M} = 10{,}000$ samples.

If the VoIs are available for every time step—as for the Lotka-Volterra example (Equation (3.3))—the QoI distribution can also be generated for every time step. The resulting PDFs can be plotted in a 3D plot as it is shown in Figure 3.12. With this technique, the dynamic in the resulting QoI distribution becomes visible.

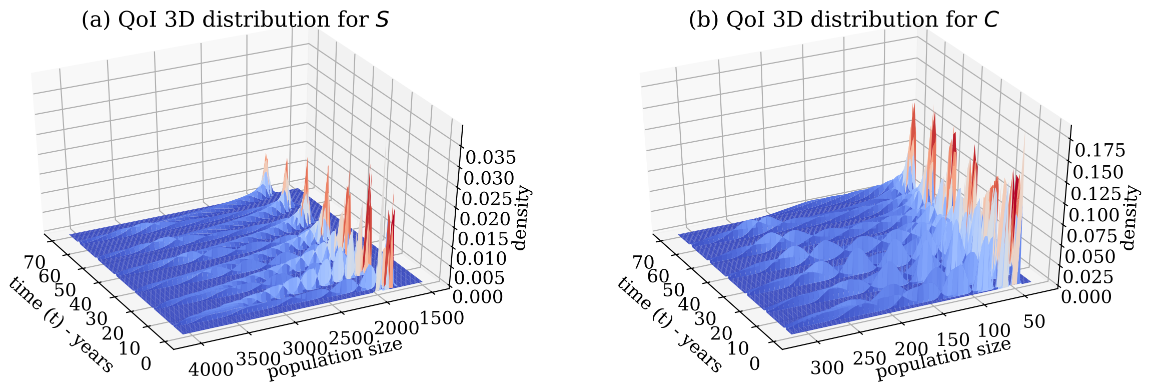


Figure 3.12: Visualisation of the PDF for each time step of the population sizes for Equation (3.3). (a) shows the PDF for the population size S of the sheep, and (b) for the population size C of the coyote. For the generation of the data, the *QoI_Dist* function is used with a secondary Monte Carlo simulation using $\mathcal{M} = 10{,}000$ samples.

Besides the visualisation of the PDF, it is also possible to use the generated QoI distribution for another UQ simulation as an input for an uncertain parameter. This allows to couple different systems where the QoI distribution serves as an input for the secondary UQ simulation.

Reconstruction of the model function via the constructed gPCE

For gPCE based methods, the constructed gPCE (Equation (2.7)) can be used as a surrogate for its original model function f. The resulting values of the gPCE reflect the situation under the uncertain conditions, and it returns the most likely value for the given uncertain input parameters. The gPCE can be used to visualise the function shape or can be used as a surrogate model for other purposes. Because the gPCE contains already all information in the coefficients c_j and the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$, the original model function f does not have to be queried again. Therefore, the evaluation of the gPCE is computationally cheap.

As an example, the gPCE is used to visualise the function shape of f_{ex1} (Equation (3.1)) and f_{ex2} (Equation (3.2)) under the given uncertain conditions. Figure 3.13 shows the function shape for the combination of the x, y, and z parameters for f_{ex1}. This looks relatively similar compared to Figure 3.1 because f_{ex1} is a very smooth model function and the resulting QoI distribution (see Figure 3.11(a)) shows a density over the whole OoI value range.

As a comparison, the gPCE based function shape for f_{ex2} is visualised in Figure 3.14, which shows a somehow similar shape compared to Figure 3.11(b) but with significant differences. The discontinuity is visible but stretched and not as clean as in the original f_{ex2} (as expected due to the polynomial structure). Additionally, at some borders, it produces wiggles with small values which are not present in f_{ex2}.

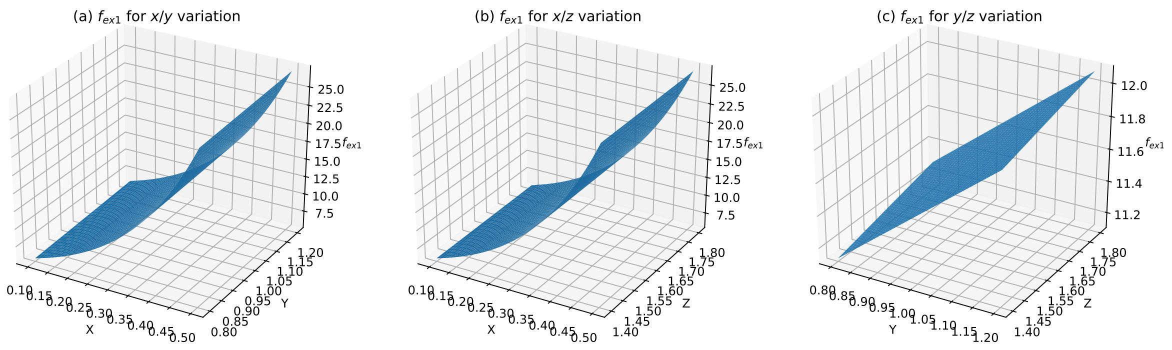


Figure 3.13: Visualisation of the resulting gPCE based function surface of f_{ex1} (Equation (3.1)) for varying values of the input parameters: (a) contains the resulting values for the x/y parameter variation, (b) for x/z, and (c) for y/z.

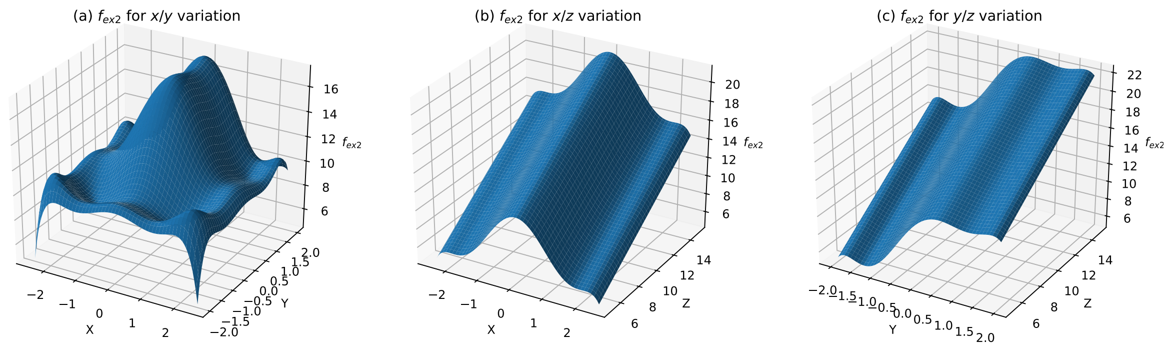


Figure 3.14: Visualisation of the resulting gPCE based function surface of f_{ex2} (Equation (3.2)) for varying values of the input parameters: (a) contains the resulting values for the x/y parameter variation, (b) for x/z, and (c) for y/z.

As a conclusion, the constructed gPCE can be used to visualise the function shape of the original model f. For smooth models, this works well and may produce similar shapes. For non-smooth models with discontinuities, the original shape can be seen, but it likely is not shown exactly and it may contain features (like wiggles) that are not present in the original model.

4 Efficient uncertainty propagation on large compute intense systems

As defined in Section 2.5, this work aims at efficient UQ simulations. The ideas in this chapter address the efficiency aspects of simulation time and computational resources for UQ simulations. The simulation time should be as short as possible, and the used computational resources fully utilised. In this thesis, the models are used as a black-box, and therefore the improvement of the models is not the subject of this thesis. Usually, the computational resources are limited and exclusively assigned to a job. Therefore, the workload should be well balanced across the computational resources, such that all resources participate in the work and as less time as possible is wasted with idling.

This chapter describes the main contributions to improve the propagation phase of a UQ simulation. After listing some preliminaries (Section 4.1), runtime definitions (Section 4.3) for UQ simulations are defined, followed by an analysis of standard scheduling strategies (Section 4.4) and the problems that can arise when they are used in the propagation phase of UQ simulations. After that, the novel idea of how to use the runtime of the black-box model runs as an additional quantity of interest (Section 4.5) is explained. With the runtime information, the standard scheduling strategies are optimised in Section 4.6. If a model is too compute intensely for repetitive UQ simulations, the use of surrogate models is a good choice, which is described in Section 4.7.

4.1 Preliminaries

For non-intrusive UQ simulations, a model is used as a black-box, which means that the underlying mathematical model and the sources for the model itself are not modified throughout the propagation. The non-intrusive UQ methods (Section 2.2) typically require many black-box runs of the model. The higher the required accuracy of the statistical moments of the QoI, the more black-box runs have to be invested.

The time for users for the development, for the actual execution of the UQ simulation, and the possibility to use computing systems is mostly limited. Also, the number of resources is limited, e.g. the number of computing units, the number of CPUs (cores per CPU), or the amount of RAM. Users usually want to have an all-in-one solution: from assimilation to propagation to certification. All this should be done inside one UQ simulation.

Because the black-box model runs are independent, it is possible to “embarrassingly” parallelise [126, 131, 67] the individual runs. This implies some computing system, especially for large-scale simulation scenarios, which is discussed in Section 4.2. One model run may already require a lot of computational resources and can, therefore, take a lot of time. For a UQ analysis of such large-scale simulation scenarios, many of this individual black-box runs add up to very long runtimes.

A crucial part of the propagation is to distribute the work to the individual computing units. A computing unit (also called worker) can be a CPU core, a group of CPU cores, a complete computing node, or a group of computing nodes. If a straightforward mapping of the work to the computing units is used, this may end up in significant performance losses. This can happen if some computing units get more work then others, which ends up in idling of some computing units. Models whose uncertain parameters influence the runtime are a common cause for the unequal distribution of workload. Moreover, because in UQ simulations, the model is run many times with different parameter values, this can end in non-predictable runtimes for the whole UQ simulation.

In this chapter, different standard scheduling strategies and their behaviour are investigated. After that, the runtime is used as a synthetic QoI[1], and with this information, a gPCE with the runtime information is created. This allows to predict the runtime of a single black-box run. While the prediction is already useful for users, the idea is to go one step further: The prediction of the runtime for each black-box run is used to control the scheduling to distribute the work among the workers equally.

4.2 Computing systems

The development of UQ simulation source codes is typically done on a development system: a personal computer (PC) in the form of a notebook or a workstation. Nowadays, typically, CPUs with 2–8 cores and 8–32 GB of RAM are used. Often, different operating systems like Linux, Windows, or MAC OSX run on development systems. Usually, threading is chosen to parallelise tasks on these development systems to utilise the resources fully.

Many different types of computing (production) systems exist in the scientific community. Usually, many computing nodes with a high-speed interconnect (e.g. InfiniBand or Intel Omni-Path) form a cluster (also called high-performance computing (HPC) system). Such computing nodes typically have CPUs with 28–64 cores and 32–128 GB of RAM. Often, Linux-based operating systems are used, which limits the direct access to the computing nodes for the users. Because many users want to use these clusters at the same time, long-term job scheduling systems such as SLURM [176] control the execution of the users' jobs. Users cannot start processes directly. Instead, they have to write a job description with the required compute resources, a time limit, and the processes to start. The job scheduling systems decides then, which jobs are started when and which computing nodes are acquired. To utilise the acquired computing nodes within a job, the message passing interface (MPI) [127] is commonly used, and within a computing node, threading is used as well.

It is possible to submit each individual black-box model run separately as a long-term scheduling job, but this splits the whole UQ simulation into separate jobs, which results in a considerably more complicated process of collecting the outputs and performing the certification phase. Another problem is that the job description files are not very interchangeable between different job scheduling systems.

An additional important aspect is to support the transition from a development system to a computing system. Since different technologies for the parallelisation are used, different scheduling mechanisms are required on the systems. If this transition requires no additional work (except for a few configuration settings) by the user, then this can significantly reduce the development time (time to solution).

The simulations in this chapter have been executed on the Linux-Cluster CoolMUC2 [115] of the Leibniz Supercomputing Centre [114]. All cluster nodes have an Intel Xeon E5-2697 v3 ("Haswell") CPU with 28 cores and 64 GB of DDR4 RAM. To submit jobs to the cluster, SLURM [176] is used for long-term scheduling.

In this work, it is assumed that the models run on one CPU core. The developed concepts in

[1]In [148, 21], there is also a synthetic output of interest used as the QoI of a UQ simulation. The authors used this information to predict the number of internal iterations of a model. The prediction of the iterations where used to group the individual runs with a similar number of iterations to a so-called ensemble group to perform them in parallel. For that, the model source code had to be changed to support parallel C++ based template operators.

this work can be translated into parallelised models, which results in more complicated scenarios, measurements, and calculations.

4.3 Runtime definitions for UQ simulations

To evaluate and compare the runtime of different scheduling strategies for UQ simulations, it is necessary to define runtime measurements. All runtime measurements are denoted with the T symbol, and equivalent predictions or estimations with $\mathbb{T}$. Figure 4.1 gives a first overview of such possible runtime measurements.

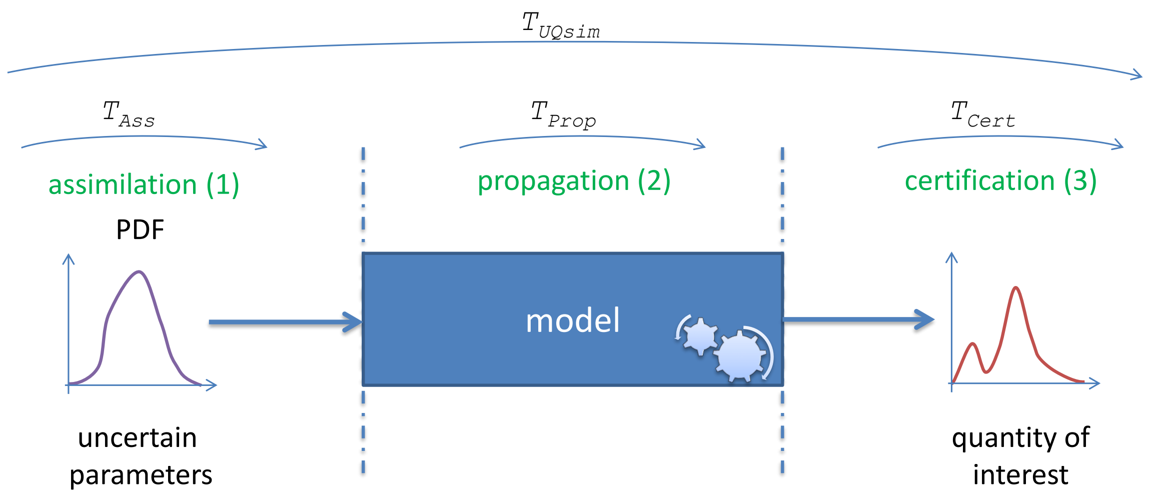


Figure 4.1: Visualisation of the runtime measurements for UQ simulations. T_{UQsim} represents the time for a whole UQ simulation. The three different UQ phases are denoted as T_{Ass}, T_{Prop}, and T_{Cert}.

The time for a complete UQ simulation is given by T_{UQsim}. For each phase of a UQ simulation, a separate measurement exists with T_{Ass}, T_{Prop}, and T_{Cert}, respectively, for the technical part of the assimilation, the propagation, and the certification.

The runtime measurement T_{UQsim} is therefore defined as

$$T_{UQsim} = T_{Ass} + T_{Prop} + T_{Cert} \, . \tag{4.1}$$

Different scheduling strategies distribute the workload differently to their computing units. The nodes n_i that are used in the propagation phase are usually grouped, which results in so-called work packages WP_j, $j = 1, \dots, J$.[2] An example illustration for the resulting work packages from a set of nodes n_i is given in Figure 4.2. For each node n_i, which contains a value for every uncertain model parameter, the model is called once in the propagation phase. A scheduling strategy is responsible for defining work packages and filling them with nodes n_i. Usually, a bidirectional mapping between a work package and a computing unit exists. However, it is not guaranteed that all work packages contain the same number of nodes n_i. It is crucial that a backward mapping from the local index $p = 1, \dots, P_j$ inside a work package WP_j to the original index i of the nodes n_i exists. Some UQ methods use only parts of the value of interest (VoI) or do some weighting when using the VoI in the certification phase.

[2]In stochastic collocation with the pseudo-spectral approach, the input nodes n_i are called collocation points z_i, $i = 1, \dots, Q$ (cf. Equation (2.14)).

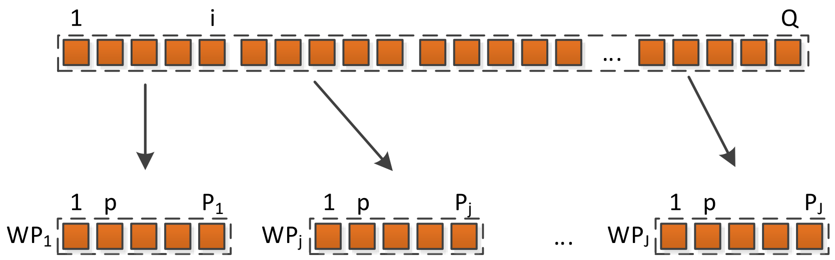


Figure 4.2: Illustration of resulting work packages, based on a set of nodes n_i for the propagation phase.

Table 4.1 contains the complete list of denoted timing measurements and their symbols. The time for solving the black-box model run with index p inside the work package WP_j is denoted as $T^p_{WP_j}$.

Denotation	Meaning
T_{UQsim}	Time of the whole UQ simulation
T_{Ass}	Time of the assimilation phase (only the technical part)
T_{Prop}	Time of the propagation phase
T_{Cert}	Time of the certification phase
$\widehat{T}_{Prop}$	Theoretical optimal propagation time (no idling, cannot get faster)
T_S	Maximum time of executions of black-box model runs over all work packages
T^i_S	Time of solving the black-box model run i
T_C	Maximum total time of communication
T^j_C	Time of communication with work package j
T_I	Maximum total time of idling over all work packages
T^j_I	Time of idling of each work package
T_{WP_j}	Time of solving work package j (including idling)
$T^p_{WP_j}$	Time of solving black-box model run with index p in work package j

Table 4.1: Definition and description of various time measurements to compare different scheduling strategies. For the measurements, the T symbol is used, and for the predicted or estimated equivalent $\mathbb{T}$ is used (later in this chapter).

The measurement for solving the whole work package (all P_j black-box model runs) is the sum of all $T^p_{WP_j}$ with

$$T_{WP_j} = \sum_{p=0}^{P_j} T^p_{WP_j} \; + \; T^j_I \; , \tag{4.2}$$

including the idling time T^j_I of the corresponding computing unit that works on WP_j.

The maximum time for solving all work packages $j = 1, \dots, J$, the communication of a computing unit with its master, and the idling of a computing unit is defined as T_S , T_C, and T_I, respectively:

$$T_S := \max_j(T_{WP_j}) \quad (4.3) \qquad T_C := \max_j(T^j_C) \quad (4.4) \qquad T_I := \max_j(T^j_I) \quad (4.5)$$

In T_C^j, the time for all communication to distribute and collect the work from or to a computing unit working on WP_j, is contained. In UQ simulations, the amount of data for the nodes and the resulting VoI values are typically low, thus, the communication part does not take much time. Usually, $T_C << T_I$ and $T_C << T_S$, and therefore T_C is not further investigated and optimised in this work. Additionally, it is assumed that the model is already optimised and T_S cannot be further optimised when using it as a block-box. The limiting performance factor within the different scheduling strategies is the idling time T_I^j of the individual work packages WP_j. Therefore, the idling is further analysed in more detail, which is part of the next section.

The whole runtime for the propagation phase,

$$T_{Prop} = T_S + T_C \ , \tag{4.6}$$

consists of the T_S and T_C component, where usually $T_C << T_S$ holds. If and only if all work packages are equally distributed and contain no idling, then the optimal runtime $\widehat{T}_{Prop}$ with a perfect load-balancing could be achieved:

$$T_{Prop} \to \widehat{T}_{Prop} \quad \iff \quad T_I^j \to 0 \quad \forall j = 1, \ldots, J \ . \tag{4.7}$$

4.4 Standard scheduling strategies

In this work, three different standard scheduling strategies (see Table 4.2) are used for the propagation phase in the UQ simulations and are subject of further investigations. For a general introduction to the scheduling topic, it is referred to [149, 29, 9].

Abbreviation	Scheduling strategy	Section
SWP	Static work packages	Section 4.4.2
SWPT	Static work packages with thread pool on node level	Section 4.4.3
DWP	Dynamic work packages	Section 4.4.4

Table 4.2: The investigated standard scheduling strategies with their abbreviation.

The problem of idling is defined in Section 4.4.1 before analysing the standard scheduling strategies in detail with two academic test functions f_{ex1} (Equation (3.1)) and f_{ex2} (Equation (3.2)) as the model in the UQ simulations. The return values of the functions are both used as the VoI for the uncertainty quantification and additionally interpreted as the model's runtime (in seconds). The model implementations of the functions use this value to perform a sleep command, before returning the VoI. Therefore, the runtime for Equation (3.1) is denoted as $\hat{f}_{ex1} = f_{ex1} = 2(e^{5\cdot|x|} + \max(y, 0) + 0.2 \cdot |z|)$ and for Equation (3.2) as $\hat{f}_{ex2} = f_{ex2} = e^{-x^2+2\,\mathrm{sign}(y)} + z$. This produces model parameter specific (synthetic) runtimes (waiting times) and is well suited to analyse different scheduling strategies because each model run can have its runtime. The runtime depends on the values of the function parameters, and therefore on the model parameter. The scheduling strategies and their optimisations are also analysed for a large-scale scenario (Section 6.4). The results of the runtime and scheduling behaviour can be found in Section 6.4.4.

The function $\hat{f}_{ex1}$ is a) used because an analytical solution exists which gives the opportunity to determine the accuracy, and b) it is designed to have a similar behaviour as the runtime of "Scenario 2: Evacuation of a building with separated families" (Section 6.4). All of the three function parameters are defined to be uncertain, with associated probability distributions listed in Table 4.3.

Parameter	Distribution
x	$U(0.1, 0.5)$
y	$U(0.8, 1.2)$
z	$U(1.4, 1.8)$

Table 4.3: Defined uncertain parameters for model function $\hat{f}_{ex1}$ (Equation (3.1)) with associated probability distributions.

To have a more challenging and non-smooth setup for the scheduling strategies, $\hat{f}_{ex2}$ (Equation (3.2)) is used. It contains a discontinuity (as can be seen in Figure 3.2), which will involve some numerical problems. $\hat{f}_{ex2}$ has been proposed in [49] with two uncertain parameters. For this investigation, a third uncertain parameter is introduced. The three uncertain parameters and its associated probability distributions are listed in Table 4.4.

Parameter	Distribution
x	$U(-2.5, 2.5)$
y	$U(-2.0, 2.0)$
z	$U(5.0, 15.0)$

Table 4.4: Defined uncertain parameters for model function $\hat{f}_{ex2}$ (Equation (3.2)) with associated probability distributions.

For all tests in this chapter, stochastic collocation with the pseudo-spectral approach (Equation (2.7)) is used, because it is a very prominent UQ method and often chosen in combination with that low number of uncertain parameters.

Remark: For non-smooth models, the non-intrusive stochastic collocation with the pseudo-spectral approach (Equation (2.7)) is not the method of choice. Other approaches like Kriging-based uncertainty quantification method with dynamic adaptive sampling [85], or the non-smooth polynomial chaos expansion (nsPCE) method [139] exists for such cases. In our case, the scheduling is one part of a UQ simulation, and there is no strong general correlation between the VoI and the runtime of a model. Additionally, the behaviour of the runtime is typically not known in advance. Another argument is that the UQ analysis of a model is usually not executed for the runtime information itself, it is done to investigate some model VoIs under the uncertain conditions, and for that, the right UQ method is chosen. Therefore, the UQ method for the models' original VoIs is reused for the runtime and scheduling investigations.

4.4.1 Idling: due to non-optimal workload

In this work, a computing unit is defined to be idling if it is not 100% utilised within a considered time period. While a computing unit or a part of it is idling, then the idling parts are waiting and are not productive anymore. In Figure 4.3, such a situation is graphically visualised: Two cluster nodes have been acquired for a job. On node 1, core 2 works the whole time, while the others have finished their work earlier and idle. The cores on node 2 do also work for some time, but they finish their work much earlier than the cores 1–N of node 1 and do idle for the rest of the acquired time.

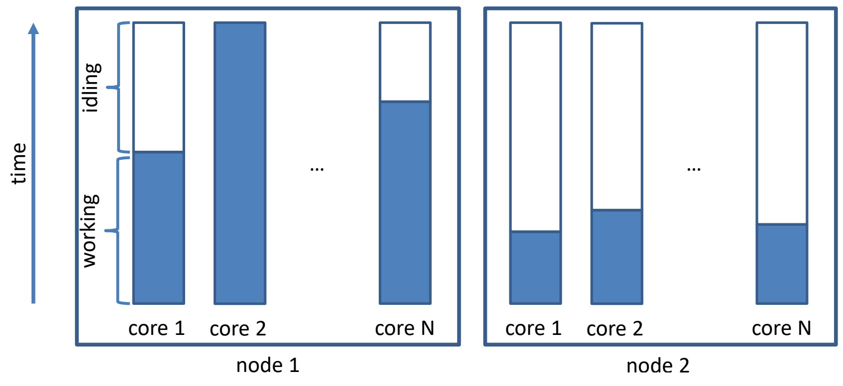


Figure 4.3: Visualisation of the idling problem for two computing nodes with their CPU cores. The filled solid blue areas represents the time a core is working, and the white area represents the idling time.

Such an idling can happen in the propagation phase of UQ simulations if the uncertain model parameters influence the runtime of the used model. Especially if these parameters influence the following parts of a model:

- initial conditions,
- boundary conditions,
- time step sizes, or
- stopping criteria of the simulation.

The idling is problematic and has at least these three consequences:

1. The runtime of the UQ simulations takes longer than expected.
2. The runtime of the whole UQ simulation is not reliably predictable.[3]
3. When a computing unit or its parts are idling, then these resources are wasted for that time period. Often, computing units are exclusively assigned to jobs until they have finished.[4]

How much idling occur depends on the runtime variation of the model and the specific scheduling strategy. In the following sections, the three standard scheduling strategies and their specific idling behaviour are analysed.

4.4.2 Static work packages

A scheduling strategy that prepares the work packages WP_j on a master and distributes the work to the computing units at the beginning of the propagation phase is called static work packages (SWP). The results of the work packages WP_j are sent back to the master after all have finished their work (end of the propagation phase). During the propagation phase, no data are transferred between the master and the worker. The master can also participate in the work by obtaining its work package.

For the performed UQ simulations on CoolMUC2 [115], such a SWP scheduling situation is visualised in Figure 4.4. The whole set of collocation points z_i is in the initial first come first served

[3]This is somehow problematic when a UQ simulation is submitted to a job scheduling system. Usually, many users wants to run jobs. The job scheduling system decides based on different properties which jobs are started first. Such properties are the number of required resources as well as the specified time limit. If shorter time limits are used, then the chance is higher that a job is started earlier. If the limit is too short and the job doesn't terminate within that limit, the job scheduling system stops (cancel) it usually hard, which typically has the consequence that the whole job (with a greater time limit) has to be repeated.

[4]There exist already different approaches to dynamically request and release resources during a job. This is the field of invasive computing (see [163])–but this is not yet generally available.

(FCFS) order filled into the work packages WP_j. For each core on each acquired computing node, one corresponding work package is created, which results in $J = \#num\_cores \times \#num\_nodes$ number of work packages. Each work package contains the same number of collocation points, except if for the last work package $Q \bmod J \neq 0$. Another exception is, if $Q < J$, but this happens typically only for rather inaccurate computations with a small number of collocation points per dimension (q). The implementation in the context of this work uses the message passing interface (MPI) [127] through the mpi4py [19, 113] Python package. On each CPU core, one MPI process is started. The master MPI process (rank 0) prepares the work packages with the containing collocation points and the corresponding parameter values for the model runs. With the MPI `scatter()` method, the data for the work packages are transferred to the worker MPI processes. Each MPI process works then individually on its own. After all have finished their work, the resulting data (VoI) is transferred back to the master via the `gather()` method. The master process finishes then the certification phase on its own.

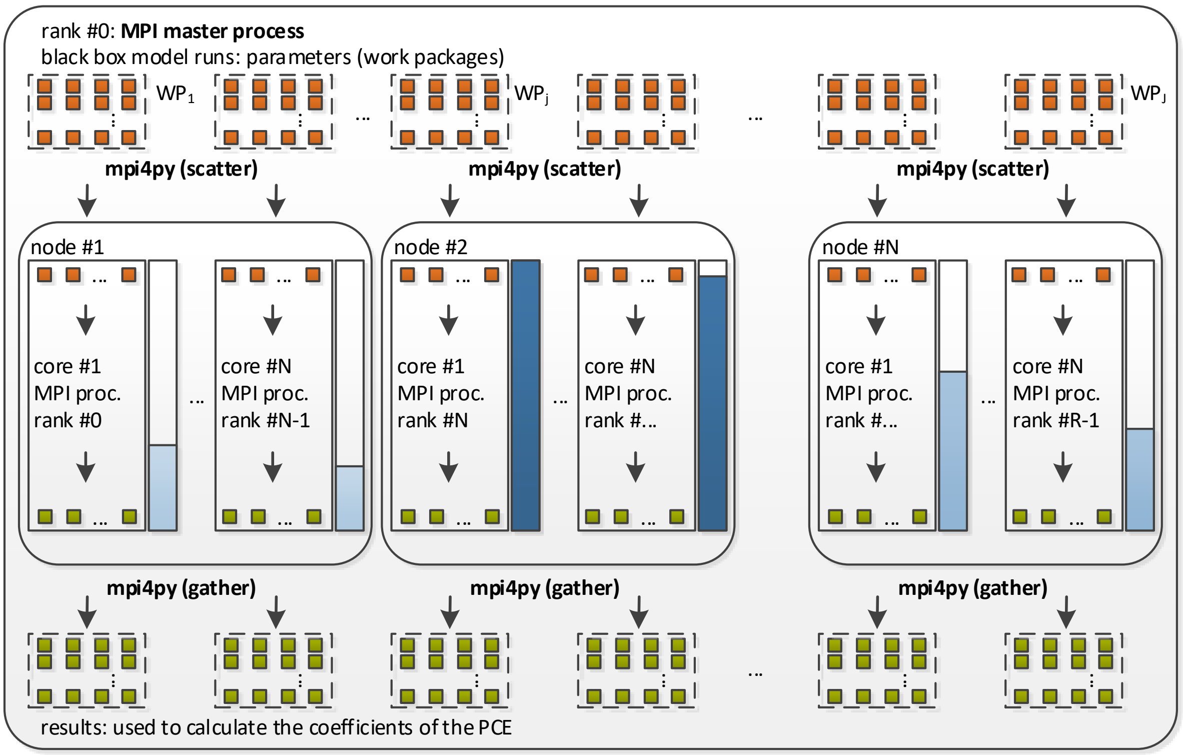


Figure 4.4: Visualisation of a static work packages (SWP) scheduling situation: The set of collocation points is equally distributed to work packages WP_j, and transferred to MPI worker processes. For each CPU core on each cluster node, exactly one work package exists. Beside each core, the utilisation is visualised: blue indicates the working time, and white the idling time.

Figure 4.5 shows the work-package-specific runtime T_{WP_j} with $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)) as the model. For these scenarios, the stochastic collocation with the pseudo-spectral approach with $Q = 8^3 = 512$ number of collocation points has been performed on 4 cluster nodes with $4 \times 28 = 112$ CPU cores (equals $J = 112$ work packages). For $\hat{f}_{ex1}$ (Figure 4.5(a)) the workload is not well balanced, because each work package runtime is different, and a lot of work packages contain only less work. The workload for $\hat{f}_{ex2}$ (Figure 4.5(b)) is also not well balanced because a few work packages contain longer runtimes than most of the others.

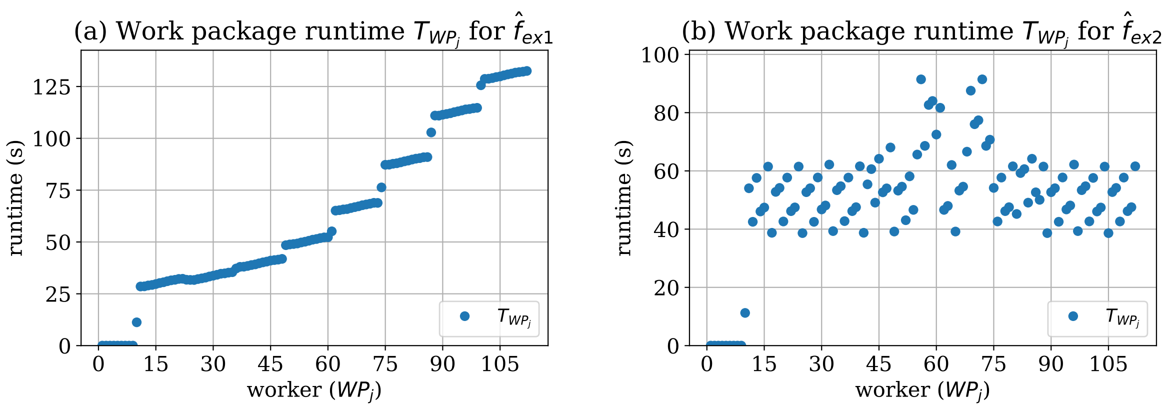


Figure 4.5: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve each work package WP_j using SWP scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

Even though all work packages WP_j contain the same number of collocation points z_i, idling may occur easily in SWP scheduling. Because each individual black-box model run within the work package has its specific runtime $T^p_{WP_j}$, the runtime for each T_{WP_j} can be different. The consequence is that some workers finish earlier than others and do than have their particular idle time T^j_I. In cases when the black-box model runs with the long runtimes are distributed over a few work packages, then the workers of the other work packages will idle for a long time.

4.4.3 Static work packages with thread pool on node level

The scheduling strategy that uses static work packages with a thread pool on node level (denoted by SWPT) is similar to SWP. All work packages WP_j are prepared on the master at the beginning of the propagation phase, and the results are sent back after all have finished. The difference to SWP is that in SWPT, only one MPI process per computing node is used. Again, all MPI processes participate in the work, and during the propagation phase, there is no data transfer.

Figure 4.6 visualises a SWPT scheduling situation. The set of collocation points z_i are similar to SWP in FCFS order filled into the work packages WP_j. On each computing node, one MPI process is started. Hence, there exist as much work packages as computing nodes $J = \#num\_nodes$. For the data transfer between the MPI master (rank 0) and the MPI worker, `scatter()` and `gather()` is used (as in SWP). Within the computing nodes, a thread pool is instantiated with the Python joblib [47] library. The thread pool uses exactly the same number of threads as a CPU offers CPU cores. To utilise the CPU cores, the thread pool takes one collocation point after another and gives this to the next free thread, which performs the black-box model run on its own. As soon as a thread has finished, it gets the next collocation point.

In Figure 4.7, the work-package-specific runtime T_{WP_j} with $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)) is plotted. As in the SWP example (Section 4.4.2), the same UQ setup with $Q = 512$ number of collocation points is used. Again, 4 cluster nodes with each 28 CPU cores are used. As Figure 4.7(a) shows for $\hat{f}_{ex1}$ (Equation (3.1)), the workload between the work packages is poorly distributed, because WP_1, WP_2, and WP_3 take significantly less time than WP_4. For $\hat{f}_{ex2}$ (Figure 4.7(b)) the workload seems to be distributed better–but WP_2 and WP_3 take longer than the others.

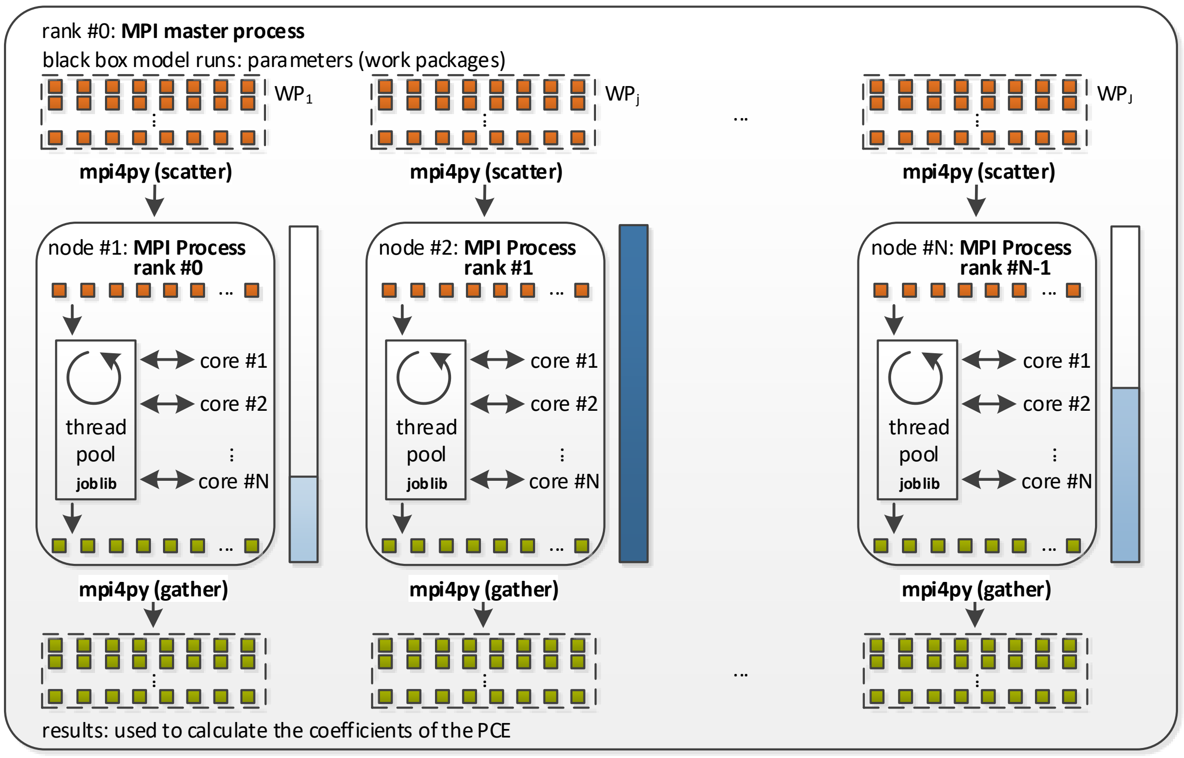

rank #0: MPI master process
black box model runs: parameters (work packages)
WP₁
WPⱼ
WPⱼ
mpi4py (scatter)
node #1: MPI Process rank #0
node #2: MPI Process rank #1
node #N: MPI Process rank #N-1
thread pool joblib
core #1
core #2
core #N
mpi4py (gather)
results: used to calculate the coefficients of the PCE

Figure 4.6: Visualisation of a static work packages with thread pool on node level (SWPT) scheduling situation: The set of collocation points is equally distributed to work packages WP_j, and transferred to MPI worker processes. On each cluster node, a thread pool is instantiated that further distributes the work to the CPU cores. Beside each computing node, the utilisation is visualised: blue indicates the working time, and white the idling time.

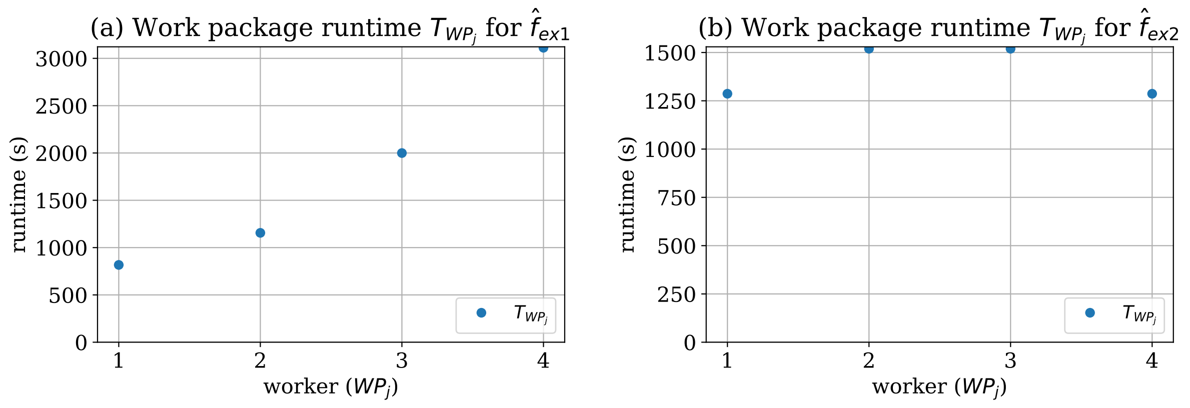


Figure 4.7: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve each work package WP_j using SWPT scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

In SWPT scheduling, idling do also exists strongly. It is usually not that prominent as in SWP scheduling, because within each computing node the dynamics of the thread pool utilises the CPU cores as long as there are unprocessed entries in the work package.

4.4.4 Dynamic work packages

The idea of having dynamic scheduling over different computing nodes is realised with the dynamic work packages (DWP) scheduling strategy. Instead of building fixed work packages WP_j at the beginning, they emerge dynamically through the propagation phase: The master process distributes entry by entry to the next free worker. A worker process waits until it receives a collocation point, starts the black-box model with the received parameters, and after the black-box run has finished, the results are sent back to the master process.

A visualisation of such a DWP scheduling is in Figure 4.8. With mpi4py [113], a pool is set up on MPI level. On each CPU core of each computing node, one MPI process is started. The MPI process with rank 0 is defined to be the master, and all others are the worker processes. For that, the `MpiCommExecutor` class from the `mpi4py.futures` package is used. Internally, the `MpiCommExecutor` class uses the `isend()` and `irecv()` MPI methods to transfer data. An MPI pool is a similar idea to a thread pool–but beyond computing node boundaries. In this setting, the MPI master process (rank 0) does not participate in the propagation. It only manages the data transfer from and to the MPI worker processes. The MPI master dynamically distributes the collocation points in first come first served (FCFS) order.

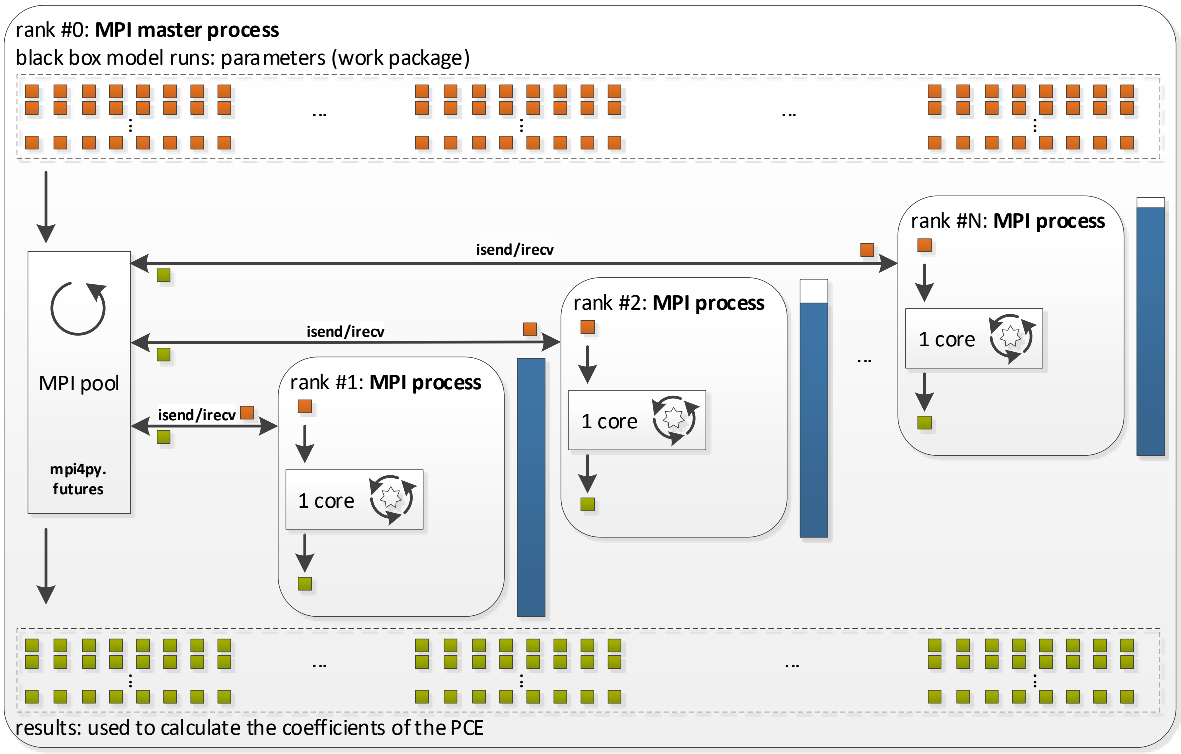


Figure 4.8: Visualisation of a dynamic work packages (DWP) scheduling situation: The MPI master process manages the MPI worker processes: it transfers the collocations points entry by entry to the workers and collects the results. Beside each MPI worker process, the utilisation is visualised: blue indicates the working time, and white the idling time.

The work-package-specific runtime T_{WP_j} for $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)) is visualised in Figure 4.9. Again, as in the SWP and SWPT example, a UQ setup with $Q = 512$ number of collocation points is used. $J = (\#num\_cores \times \#num\_nodes) - 1$ number of work packages (one for each MPI worker) exist and are dynamically filled during the propagation. Figure 4.9(a) shows the dynamically created work packages WP_j for $\hat{f}_{ex1}$ (Figure 4.9(a)) as the model. It can be seen that the workers tend to fully participate in the work. However, the workload is not completely evenly distributed. For $\hat{f}_{ex2}$ (Figure 4.9(b)), the workload is also not evenly distributed but it does not spread as much as in the SWP (Figure 4.5(b)) example.

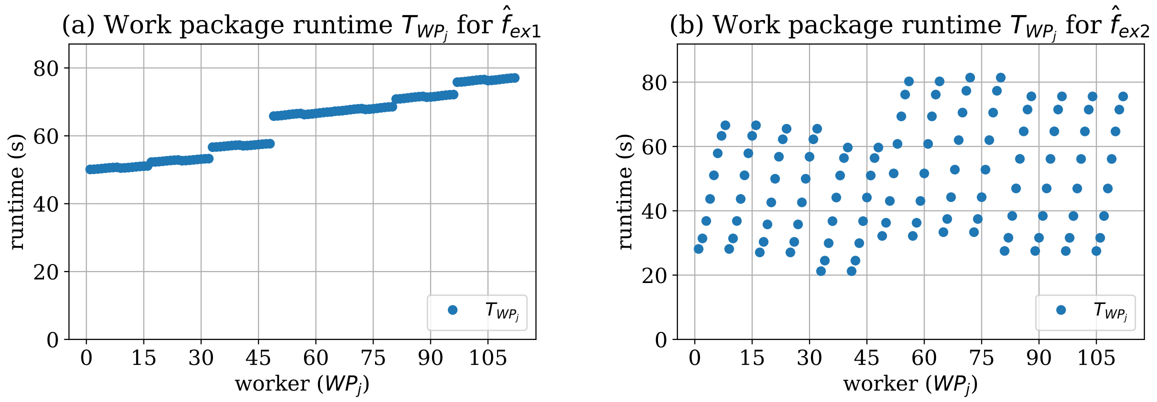


Figure 4.9: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve the dynamically emerged work packages WP_j using DWP scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

The DWP scheduling can reduce the idling through its dynamic emerging of the work packages WP_j. The workload is still not completely evenly distributed but in many cases better than with SWP and SWPT. Long idling can still arise for some workers when a few black-box model runs with long runtimes are started at the very end because then the other finished workers will have to idle for that time.

4.4.5 Summary of standard scheduling strategies

For a detailed analysis of the propagation runtimes T_{Prop} for the two examples $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)), various simulations have been performed: T_{Prop} is measured for a different number of collocation points $q = 4, 5, \dots, 12$ and for a different number of cluster nodes $cn = 2, 3, 4, 5$. Figure 4.10 contains the corresponding plots. For both examples, SWP shows the longest propagation runtime, SWPT the second longest, and DWP the shortest.

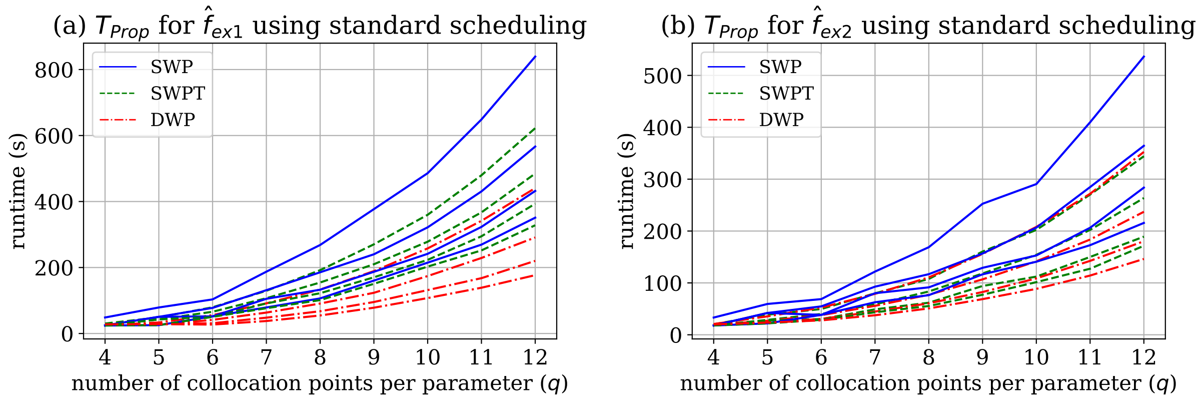


Figure 4.10: Measured propagation runtimes T_{Prop} in seconds for the three standard scheduling strategies SWP, SWPT, and DWP: (a) contains the runtimes for $\hat{f}_{ex1}$ (Equation (3.1)) and (b) for $\hat{f}_{ex2}$ (Equation (3.2)). The propagation runtimes T_{Prop} are measured for different $q = 4, 5, \dots, 12$ and for different number of cluster nodes $cn = 2, 3, 4, 5$.

Figure 4.11 contains the corresponding speed-ups[5] for the measured propagation runtimes T_{Prop}. The speed-up of SWP to SWPT for $\hat{f}_{ex1}$ (Figure 4.11(a)) is about 1.0–1.4 (for $q \geq 6$), and for SWP to DWP about 1.6–2.1 (for $q \geq 6$). For $\hat{f}_{ex2}$ (Figure 4.11(b)) the speed-up for SWP to SWPT is about 1.2–1.6 (for $q \geq 6$) and for SWP to DWP about 1.3–1.7 (for $q \geq 6$).

The problem of SWP and SWPT is that there is no dynamic update of the workload after some computing units have finished its initial work package, and the work packages contain the same number of entries, but not the same amount of work (runtime). The more dynamic scheduling aspects a scheduling strategy contains, the less work-package-specific idling T_I^j occurs, and therefore the propagation runtime T_{Prop} is shorter. The presented standard scheduling strategies still contain some idling, which indicates the potential for further improvements. Another point that is still missing is the prediction of how much runtime the propagation phase of a UQ simulation will take, which is part of the next section.

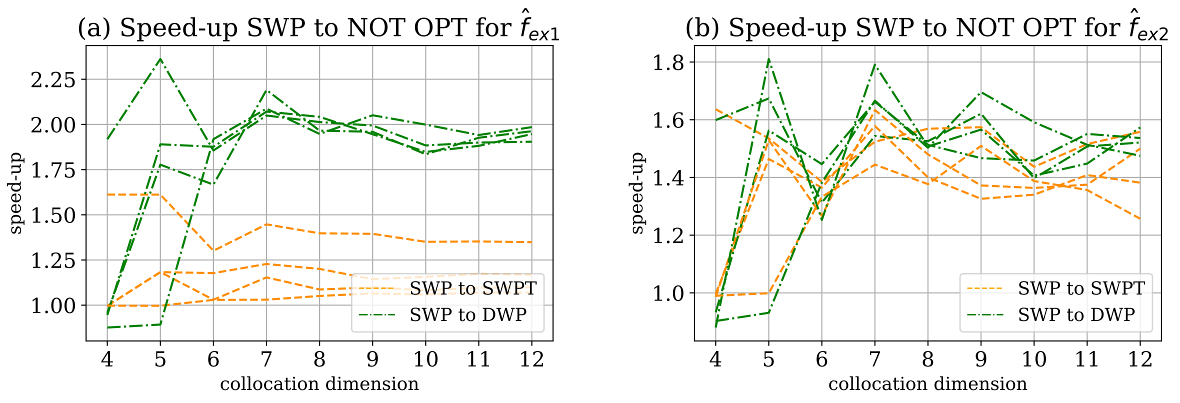


Figure 4.11: Speed-up for T_{Prop} of SWP compared to SWPT and DWP: (a) contains the speed-up for $\hat{f}_{ex1}$ (Equation (3.1)) and (b) for $\hat{f}_{ex2}$ (Equation (3.2)). Each plot contains 4 speed-up lines (of identical colour and line style) for SWP to SWPT and DWP that corresponds to the different number of cluster nodes $cn = 2, 3, 4, 5$.

4.5 Runtime as quantity of interest

The knowledge about the runtime of a UQ simulation is of value for users, to know when the results can be expected. Additionally, when they submit jobs to a long-term job scheduling system, they can determine better time limits for their jobs (not too short to not time out, and not too long not to have to wait for a long time to get scheduled). It is hard to predict the runtime of a model when the runtime depends on the input parameters of the model, especially when these parameters are uncertain and not exactly known, as it is the case for UQ simulations. To estimate the propagation runtime T_{Prop}, a runtime predictor is required, which can estimate the runtime of the model depending on its input parameter values.

4.5.1 Runtime prediction of UQ simulations

The idea is to measure the runtime T_S^i of each individual model run within a UQ simulation, as an additional value of interest (VoI). With this information, a runtime predictor based on the UQ modelling is created. $f(x, t, \zeta)$ from Equation (2.14) is extended with the runtime measurement, and the additional part comprising the runtime is denoted by $\hat{f}(x, t, \zeta)$.

[5]All speed-ups in this thesis have been rounded down to one digit after the decimal point.

For stochastic collocation with the pseudo-spectral approach, the runtime predictor rp_N is constructed with

$$rp_N(x,t,\boldsymbol{\zeta}) := \sum_{j=0}^{N} \hat{c}_j(x,t) \cdot \Phi_j(\boldsymbol{\zeta}) \ , \tag{4.8}$$

representing the whole gPCE for the runtime of a model. The corresponding coefficients $\hat{c}_j(x,t)$ are determined with

$$\hat{c}_j(x,t) \approx \frac{1}{\gamma_j} \sum_{i=1}^{Q} \hat{f}(x,t,z_i)\Phi_j(z_i)w_i \ , \tag{4.9}$$

where the runtime component $\hat{f}(x,t,z_i)$ is used as the value of interest (VoI).

The runtime specific gPCE—the rp_N—strongly depends on the uncertain parameters $\boldsymbol{\zeta}$. When calling the rp_N with specific values for the uncertain parameters, then the most certain runtime for this input parameter combination is returned. To create and use the rp_N, the workflow is as follows: First, a training phase is required where the runtime of $\hat{f}(x,t,z_i)$ is measured as an additional value of interest for the UQ simulation during the propagation phase. The next step is to create the runtime predictor rp_N with Equation (4.8) in the certification phase and to save it into a file. From now on, the rp_N can be loaded from the file for the next UQ simulations of the same model. Starting with the technical phase of the assimilation, the collocation points z_i are created as usual. With each collocation point z_i, the runtime predictor rp_N is called once, to predict the most certain runtime $\mathbb{T}_S^i$ for each single model run. With this information, each scheduling strategy can, therefore, predict the runtime $\mathbb{T}_{WP_j}^p$ of each entry in the work packages, and therefore the runtime $\mathbb{T}_{WP_j}$ for each work package. Because it is known that $T_C << T_S$, the prediction of $\mathbb{T}_{Prop}$ depending on the scheduling strategy is possible. If T_{Ass} and T_{Cert} are also known, the whole UQ simulation time $\mathbb{T}_{UQsim}$ can be predicted. As for every other QoI, it is also possible to create the statistical QoI moments also for the runtime. Therefore, the mean μ_{rp_N} (Equation (2.17)), the variance $\sigma^2_{rp_N}$ (Equation (2.18)), the percentiles, and even a sensitivity analysis (Section 2.3) is possible, to determine which parameter or its combination has the most influence on the model's runtime.

A scheduling strategy can also use the runtime prediction of the individual model runs to optimise the scheduling, as is demonstrated in Section 4.6.

Remark: It is obvious that a training phase is required to construct a runtime predictor for stochastic collocation with the pseudo-spectral approach. Usually, a UQ simulation is repeated multiple times with different values, and therefore there is a possibility to reuse the runtime predictor. Other approaches with adaptive evaluations like adaptive sparse-grid-based approaches such as [52] or [38] exist, where the runtime predictor can already be helpful within the first UQ simulation.

4.5.2 Determining runtime prediction quality

For a detailed analysis of the quality of the constructed runtime predictors rp_N, the errors are determined with the measured runtimes T_S^i compared to the predicted runtimes $\mathbb{T}_S^i$ of each individual black-box model run. For the absolute error ϵr_i,

$$\epsilon r_i := |T_S^i - \mathbb{T}_S^i| \ , \tag{4.10}$$

is defined, and for the relative error $\epsilon r_{i,\text{rel}}$

$$\epsilon r_{i,\text{rel}} := \frac{\epsilon r_i}{\max(|T_S^i|, |\mathbb{T}_S^i|)} = \frac{|T_S^i - \mathbb{T}_S^i|}{\max(|T_S^i|, |\mathbb{T}_S^i|)} \ , \tag{4.11}$$

to determine the runtime prediction error for each individual black-box run.

For the comparison of different runtime predictors rp_N, the $\mu(\epsilon r)$ as the mean error, and the discrete $L^2(\epsilon r)$ error norm is defined as:

$$\mu(\epsilon r) := \frac{1}{Q}\sum_{i=1}^{Q}\epsilon r_i \qquad (4.12) \qquad\qquad L^2(\epsilon r) := \sqrt{\frac{1}{Q}\sum_{i=1}^{Q}\epsilon r_i^2} \qquad (4.13)$$

4.5.3 Numerical results of runtime prediction

In this section, a scheduling-independent analysis of the numerical results for the runtime prediction is given. Again, the test functions $\hat{f}_{ex1}$ Equation (3.1) and $\hat{f}_{ex2}$ (Equation (3.2)) are used as models. For both models, various UQ simulations have been performed with $q = 4, 5, \ldots, 12$ different numbers of collocation points per parameter and for different numbers of cluster nodes $cn = 2, 3, 4, 5$. For each performed UQ simulation, a runtime predictor rp_N is created and used for the numerical analysis in the following.

Figure 4.12(a) shows the real runtime T_S^i (blue circles) vs the predicted runtime $\mathbb{T}_S^i$ (filled green dots) for $\hat{f}_{ex1}$ Equation (3.1). Because the green filled dots are plotted over the blue circles, and the blue circles are almost hidden, the accuracy of the prediction quality is high. The prediction quality for $\hat{f}_{ex2}$ is not as high as for $\hat{f}_{ex1}$, as can be seen in Figure 4.12(b) (some blue circles are visible). This can be explained with the discontinuity in function $\hat{f}_{ex2}$ (and therefore in the runtime as can be seen in Figure 3.2), because around the discontinuity and on the edges, the rp_N cannot predict the values precisely.

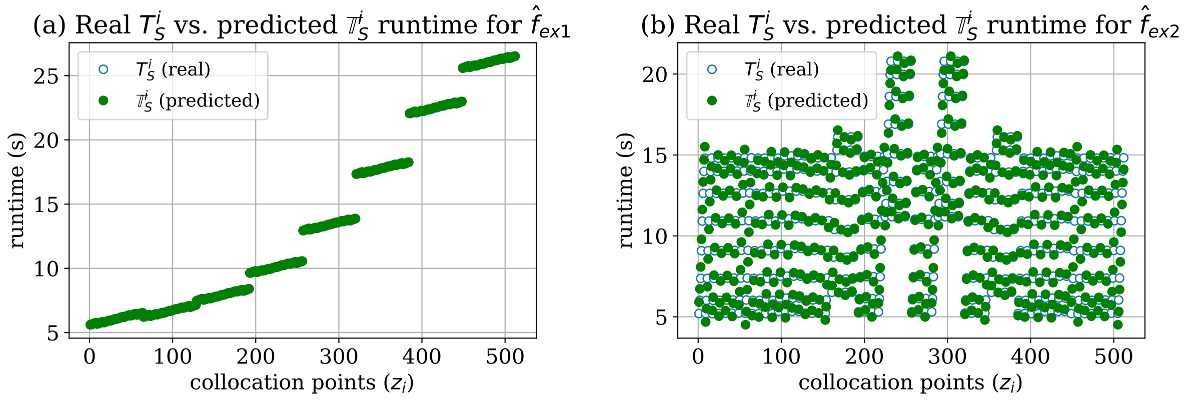


Figure 4.12: Real (measured) runtime T_S^i (blue circles) vs the predicted runtime $\mathbb{T}_S^i$ (filled green dots) for $\hat{f}_{ex1}$ in (a) and for $\hat{f}_{ex2}$ in (b). For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

In Figure 4.13(a), the absolute error ϵr_i for $\hat{f}_{ex1}$ is calculated by comparing the predicted values against the analytical solution of Equation (3.1) for $q = 8$. The absolute error is with values of the order 10^{-5} seconds rather small. For $\hat{f}_{ex2}$, the absolute error ϵr_i values (prediction compared to Equation (3.2)) ranges from 0.01 to 0.71 seconds, as can be seen in Figure 4.13(b). The higher absolute error values were already qualitatively assumed in Figure 4.12(b) and are hereby confirmed quantitatively.

The corresponding relative errors $\epsilon r_{i,\text{rel}}$ for $\hat{f}_{ex1}$ are plotted in Figure 4.14(a). The values are small in the order of 10^{-6}. Figure 4.14(b) shows the relative error $\epsilon r_{i,\text{rel}}$ for $\hat{f}_{ex2}$, whose values range mainly from 0 to 0.75, only a few reaching up to 1.35.

A broader overview of the absolute error trend for $\hat{f}_{ex1}$ can be found in Figure 4.15(a). The mean error $\mu(\epsilon r)$ as well as the 5th $p_5(\epsilon r)$ and the 95th $p_5(\epsilon r)$ percentile for different $q = 4, 5, \ldots, 12$ is plotted. It can be seen that the error is visible for $q \leq 6$ because the rp_N has too less information for an accurate prediction. Starting from $q \geq 7$, the errors are negligibly small.

The absolute error statistics for $\hat{f}_{ex2}$ in Figure 4.15(b) shows a significant error for all tested $q = 4, 5, \ldots, 12$.

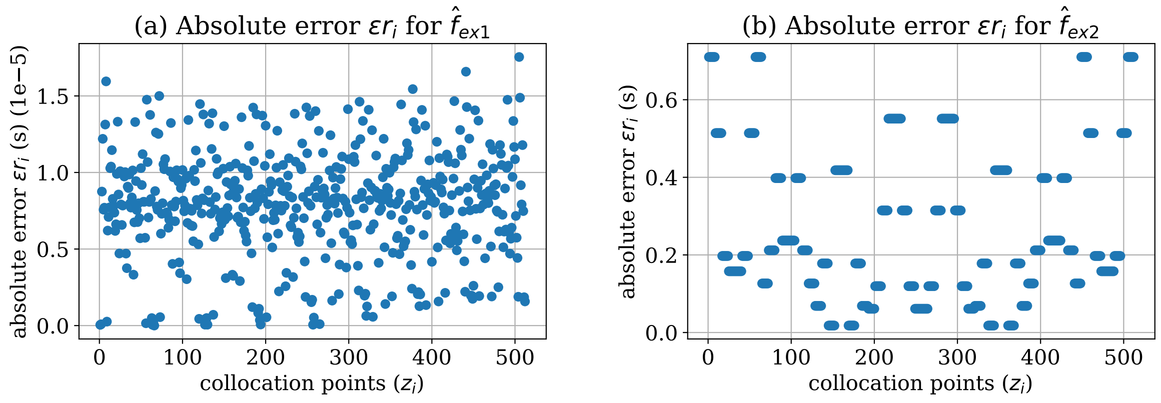


Figure 4.13: Absolute error ϵr_i for $\hat{f}_{ex1}$ in (a) and for $\hat{f}_{ex2}$ in (b). For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

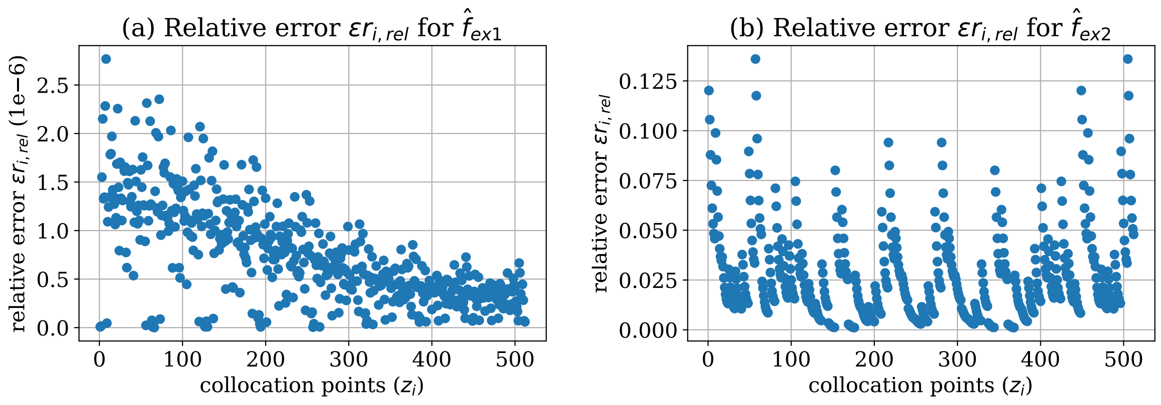


Figure 4.14: Relative error $\epsilon r_{i,\mathrm{rel}}$ for $\hat{f}_{ex1}$ in (a) and for $\hat{f}_{ex2}$ in (b). For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

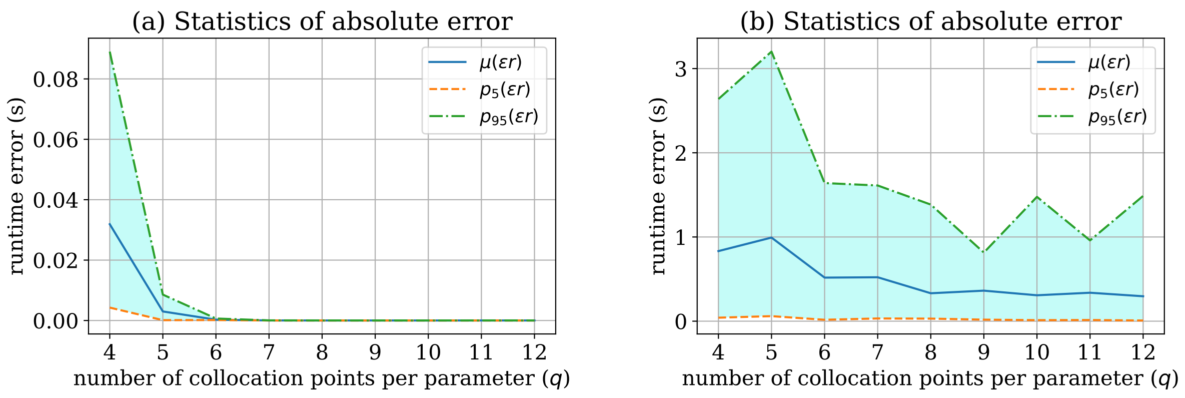


Figure 4.15: Absolute error statistics with mean error $\mu(\epsilon r)$ as well as the 5th $p_5(\epsilon r)$ and the 95th $p_5(\epsilon r)$ percentile for $\hat{f}_{ex1}$ in (a) and for $\hat{f}_{ex2}$ in (b). For the UQ simulations, different number of collocation points $q = 4, 5, \ldots, 12$ per parameter have been used.

Figure 4.16(a) shows the $L^2(\epsilon r)$ error norm for $\hat{f}_{ex1}$. The decreasing error down to 10^{-5} for increasing q is confirmed by the $L^2(\epsilon r)$ error norm. For $\hat{f}_{ex2}$, the $L^2(\epsilon r)$ error norm is also decreasing for higher q, as can be seen in Figure 4.16(b), but it is still in the order of one.

The numerical results in this section show that the error values for $\hat{f}_{ex1}$ are rather small and the runtime predictor rp_N seems to work well for the prediction of the runtime $\mathbb{T}_S^i$ for the individual black-box model runs. As expected, the runtime predictor rp_N for $\hat{f}_{ex2}$ is not that accurate. However, the relative error values are still $< 1\%$, and the rp_N is likely to be of value for runtime predictions and accurate enough to estimate the whole propagation runtime $\mathbb{T}_{Prop}$. In the next sections, the predictions are used as an input for the scheduling strategies.

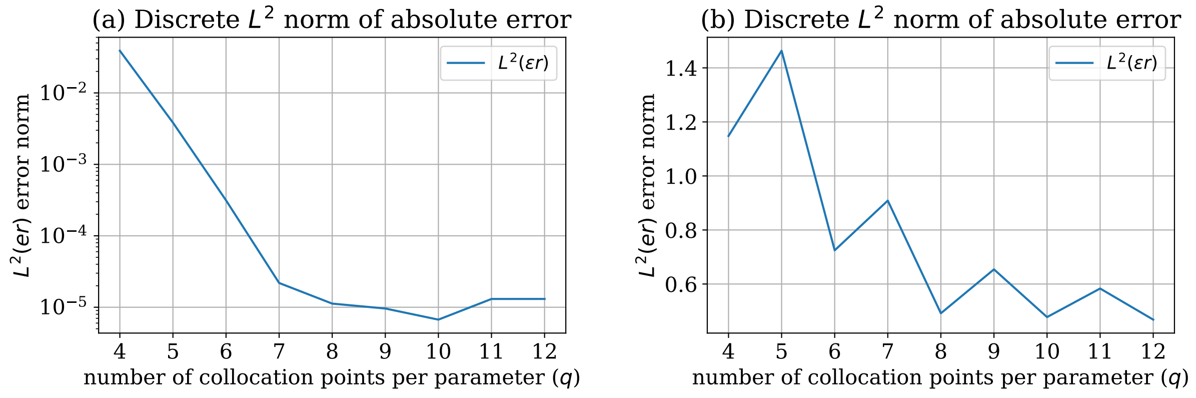


Figure 4.16: Discrete $L^2(\epsilon r)$ norm of the absolute error for $\hat{f}_{ex1}$ in (a) and for $\hat{f}_{ex2}$ in (b). For the UQ simulations, different number of collocation points $q = 4, 5, \dots, 12$ per parameter have been used.

4.6 Optimised scheduling strategies via work reordering

In this section, the idea of the optimised scheduling strategies via work reordering is presented. During a UQ simulation, a runtime predictor rp_N is created (as introduced Section 4.5). For the next UQ simulations, the runtime predictor rp_N is used to predict the runtime $\mathbb{T}_S^i$ for each black-box model run. This information is further used by the scheduling strategy to optimise the scheduling by reordering the work and controlling the work package creation (distribution of work)–but this depends strongly on the specific scheduling strategy. Before diving into the optimised scheduling strategies, a general overview of how to create and use a runtime predictor rp_N within a UQ simulation using the stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is given.

Figure 4.17 visualises the sequence of steps within a UQ simulation. The white boxes are always performed, and the green boxes are the additional steps to create a runtime predictor rp_N. In the propagation phase (2), only the measurement of the runtime T_S^i for each black-box run in step 2.3b is additionally required. The certification phase (3) additionally creates the runtime predictor rp_N and the runtime statistics in step 3.1b, and finally save it to a file in 3.2b. This shows that the creation of the runtime predictor rp_N can–with small changes–be directly integrated into a UQ simulation workflow.

For the usage of the runtime predictor rp_N to optimise a scheduling strategy, only the propagation phase (2) has to be extended, as can be seen in Figure 4.18. The activity 2.1 *Create work packages depending on scheduling strategy* is extended and split into several parts: 2.1a loads the runtime predictor rp_N from a file. In 2.1b, the runtime predictor rp_N is used to predict the runtime $\mathbb{T}_S^i$ for each single black-box model run, depending on the values of the collocation points z_i. After that, in 2.1c, the runtime information $\mathbb{T}_S^i$ is used by the scheduling strategy to reorder the individual black-box model runs. The new order is then used in 2.1d to create the work packages WP_j. It is very important that the collocation points z_i are not only reordered–a backward mapping has to be possible because some UQ methods rely on the correct order of the collocation points z_i and the corresponding results. In this work, lists of the original and the sorted indices are created, which allows a backward mapping of the results (VoIs) to match the original order of the collocation points z_i. This reordering of the results (VoIs) is performed in 2.5a.

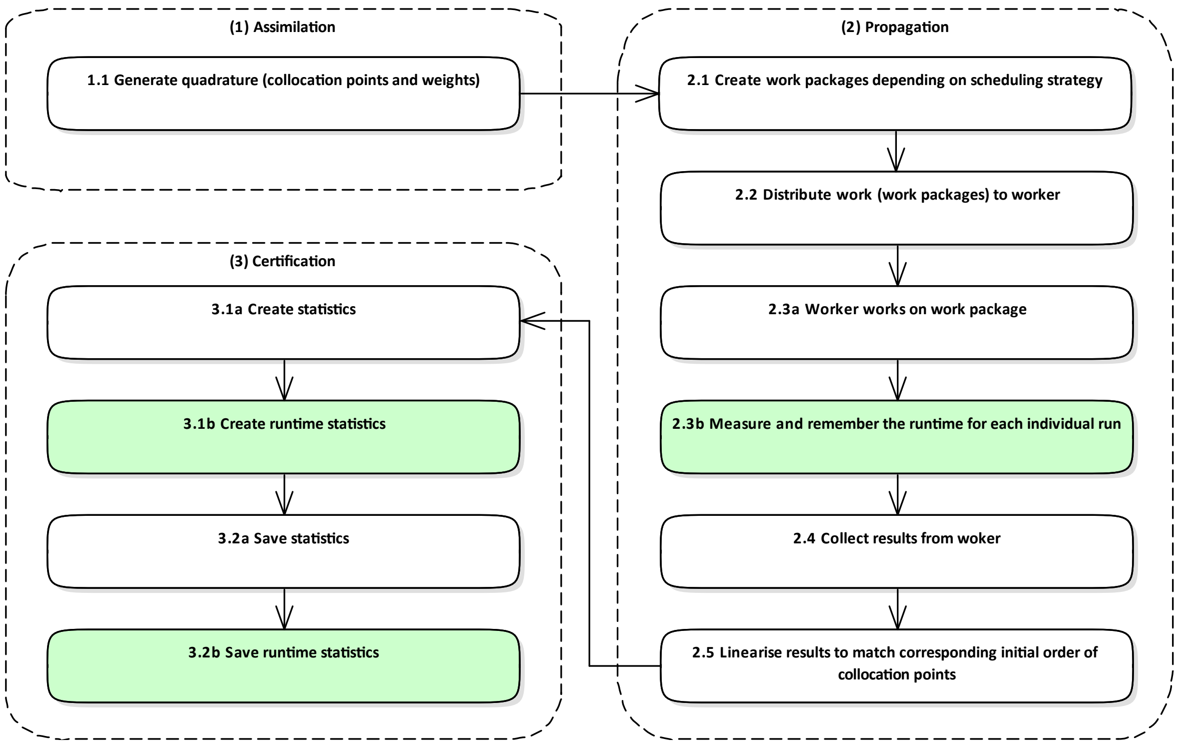

(1) Assimilation
1.1 Generate quadrature (collocation points and weights)
(2) Propagation
2.1 Create work packages depending on scheduling strategy
2.2 Distribute work (work packages) to worker
2.3a Worker works on work package
2.3b Measure and remember the runtime for each individual run
2.4 Collect results from woker
2.5 Linearise results to match corresponding initial order of collocation points
(3) Certification
3.1a Create statistics
3.1b Create runtime statistics
3.2a Save statistics
3.2b Save runtime statistics

Figure 4.17: Visualisation of the sequence of activities (UML activity diagram) to create the runtime statistics with a runtime predictor rp_N within a UQ simulation that uses a scheduling strategy. The white boxes are always executed, and the green boxes are the additional activities to determine and save the runtime statistics.

As described in Section 4.4, idling occurs because of the lack of knowledge about the runtime and the FCFS order when scheduling the work. The authors of [63] also analysed FCFS for various settings and showed that it could have a varying or poor utilisation. If $T_I \to 0$, the order would be optimal. The three standard scheduling strategies, presented in Section 4.4, are now extended with the creation and the usage of the runtime prediction to control the scheduling with the goal of minimising the idling. This results in optimised scheduling strategies listed in Table 4.5. For comparison, all optimised scheduling strategies are tested with $\hat{f}_{ex1}$ Equation (3.1) and $\hat{f}_{ex2}$ Equation (3.2) as the model-specific runtimes.

Abbreviation	Scheduling strategy	Basis	Section
SWP$_{OPT}$	Optimised static work packages	SWP	Section 4.6.1
SWPT$_{OPT}$	Optimised static work packages with thread pool	SWPT	Section 4.6.2
DWP$_{OPT}$	Optimised dynamic work packages	DWP	Section 4.6.3

Table 4.5: List of optimised scheduling strategies with their abbreviation and their scheduling strategy basis.

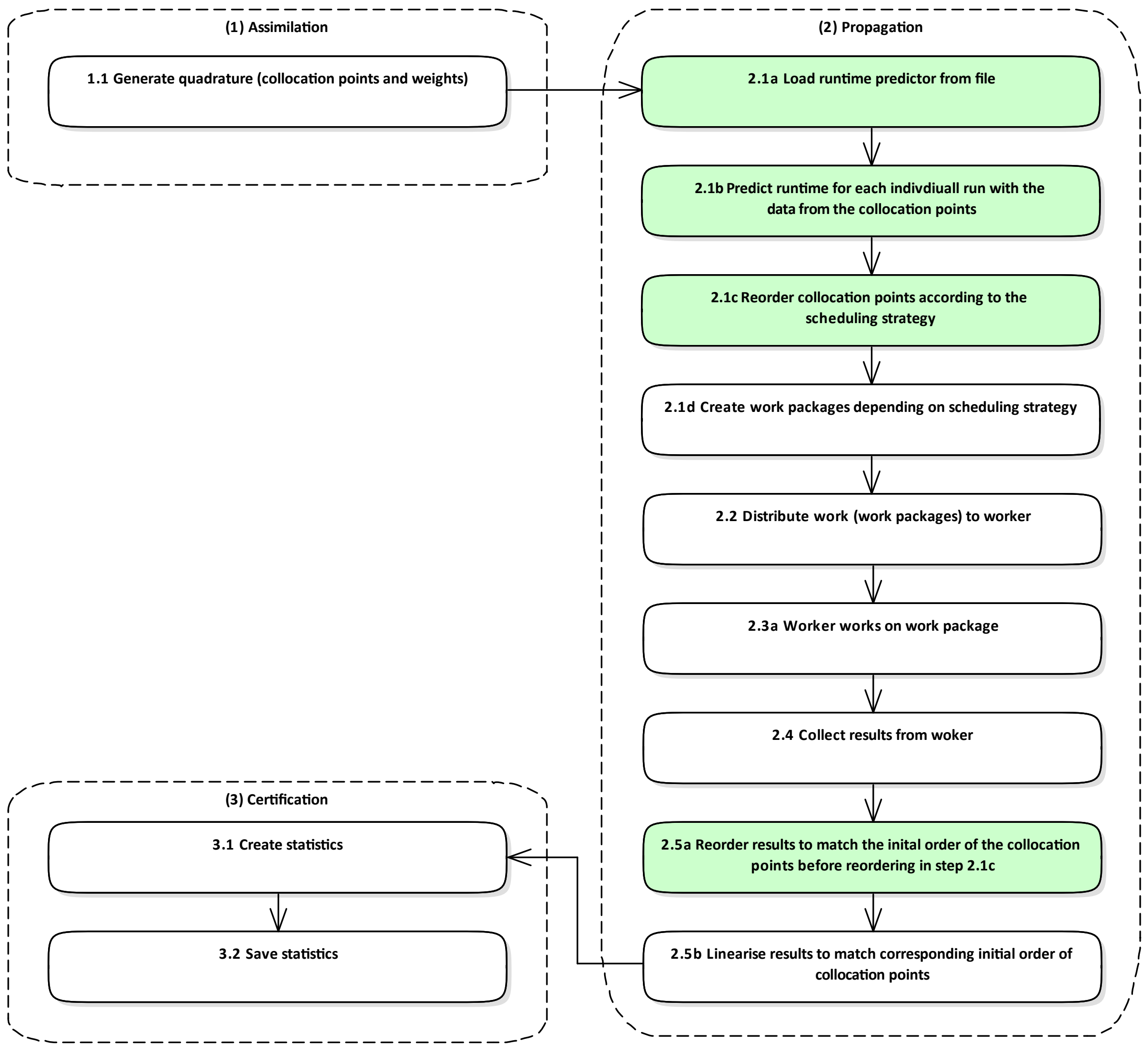

(1) Assimilation
1.1 Generate quadrature (collocation points and weights)
(2) Propagation
2.1a Load runtime predictor from file
2.1b Predict runtime for each indivdiuall run with the data from the collocation points
2.1c Reorder collocation points according to the scheduling strategy
2.1d Create work packages depending on scheduling strategy
2.2 Distribute work (work packages) to worker
2.3a Worker works on work package
2.4 Collect results from woker
2.5a Reorder results to match the inital order of the collocation points before reordering in step 2.1c
2.5b Linearise results to match corresponding initial order of collocation points
(3) Certification
3.1 Create statistics
3.2 Save statistics

Figure 4.18: Visualisation of the sequence of activities (UML activity diagram) to use a runtime predictor rp_N for reordering the work within a UQ simulation. The white boxes are always executed, and the green boxes are the additional activities to create and use the runtime information $\mathbb{T}_S^i$ to optimise the scheduling.

4.6.1 Optimised static work packages

For the SWP scheduling, the collocation points z_i are taken in their initial FCFS order to create the work packages WP_j. Each work package WP_j contains the same number of collocation points P_j. In the optimised static work packages (SWP$_{OPT}$) scheduling, the goal is to create the work packages in such a way that they contain the same work (i.e. the same runtime T_{WP_j}) and not necessarily the same number P_j of collocation points z_i anymore. This problem belongs to the P_m category [149, p. 14] in scheduling theory, where m denotes the same number of processors or computing units, respectively. An optimal distribution of the work ($T_I \rightarrow 0$) in the P_m category is proven to be NP-hard [149, p. 114], unfortunately. The problem of preparing the work packages is also known as the bin-packing problem: packing different sized volumes into a finite number of bins. For the scheduling problems in this thesis, this reads: packing collocation points z_i with different runtime $\mathbb{T}_S^i$ into a finite number of work packages. Because the bin-packing problem is well known, several heuristics exist to tackle the NP-hardness. For this work, the MULTIFIT heuristic is used, which is described in [14]. MULTIFIT is chosen because of its good results for large problems (as it is the case in this work) and its relatively less computational time [104] compared to other heuristics such as LISTFIT [59] or COMBINE [109]. Its worst-case factor $R_m(MF)$ is in general estimated with

$$R_m(MF) \leq 1.222\,. \tag{4.14}$$

In [14, 102], further investigations and improvements can be found on that. This means that the optimal propagation time $\widehat{T}_{Prop}$ multiplied by $R_m(MF)$ is the longest propagation time T_{Prop} in the worst-case.

The workflow of the MULTIFIT heuristic is as follows: It starts with the unordered list of collocations points z_i. Then, the entries are sorted by their runtime T_S^i in descending order, starting with the longest runtime first. Usually, MULTIFIT builds the work packages upon its procedure, and it is unknown how many work packages (bins) should be used. For the work packages in this work, the number of work packages J is defined to be a fixed constant for MULTIFIT, hence it will always generate exactly J work packages. MULTIFIT works iteratively to find a package size capacity C with k iterations[6], to fairly put the same runtime T_{WP_j} into all work packages WP_j. For each step in the iteration, the first fit decreasing (FFD) algorithm is applied. FFD starts with the ordered list of collocation points z_i and fills the first work package, as long as C is not exceeded. After that, it continues by filling the next work package. To also fill in collocation points z_i with small runtimes T_S^i in already filled work packages, FFD always starts with the first work package to fill in a collocation point z_i. After the k iterations, where FFD is employed multiple times and the capacity C is iteratively refined, the workload of the work packages is—with the worst-case factor $R_m(MF)$—almost equally distributed.

As an example, Figure 4.19 contains nine entries which should be distributed into three work packages. The first row shows the unordered entries with their relative runtime (broader entries take more time). After MULTIFIT is executed on the unordered list, three work packages with similar runtime T_{WP_j}, but not with the same number of entries, have been produced. The order of the entries inside each work package is in descending order regarding there runtime, which comes from the initial ordering step in combination with the FFD algorithm.

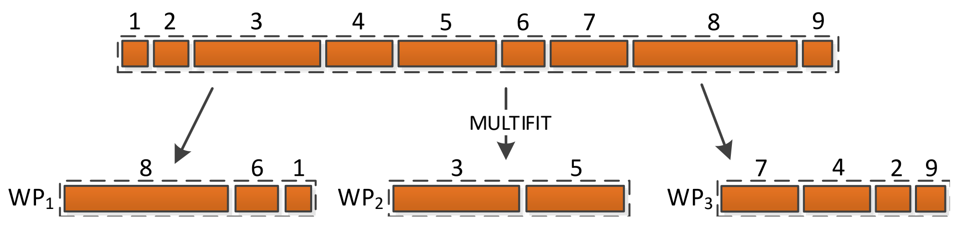


Figure 4.19: Example of resulting work packages with the MULTIFIT heuristic. Each rectangle (entry) represents a work item and its relative width is the length of the runtime. The broader an entry, the longer its runtime. The first row shows the unordered entries and the second row the resulting work packages WP_1, WP_2, WP_3.

The resulting work packages for $\hat{f}_{ex1}$ (Equation (3.1)) using SWP$_{OPT}$ are visualised in Figure 4.20(a). Compared to the SWP scheduling (Figure 4.5(a)), the measured work package runtimes T_{WP_j} are relatively similar to each other. Also for $\hat{f}_{ex2}$ (Equation (3.2)), the measured work package runtimes T_{WP_j} are relatively equally distributed, as Figure 4.20(b) shows. Compared to Figure 4.5(b) with the SWP scheduling, this is a significant improvement.

The SWP$_{OPT}$ scheduling with the MULTIFIT heuristic can produce work packages with a relatively similar runtime T_{WP_j} using the constructed runtime predictor rp_N and the MULTIFIT heuristic. This reduces the idling significantly for the tested model functions.

[6]The number of iterations k can be specified in the MULTIFIT implementation. In this thesis $k = 100$ is used.

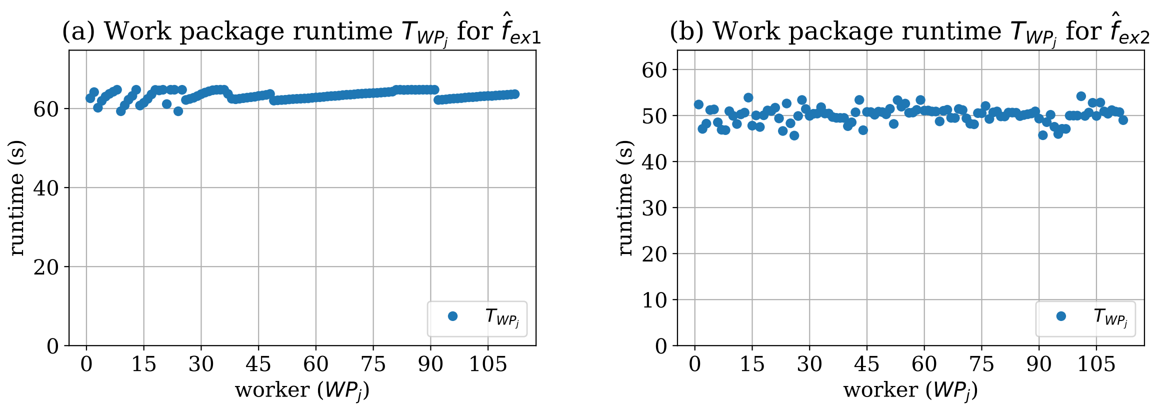


Figure 4.20: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve each work package WP_j using SWP$_{OPT}$ scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

4.6.2 Optimised static work packages with thread pool on node level

The optimised static work packages with thread pool on node level (SWPT$_{OPT}$) scheduling strategy is the improved version of SWPT (Section 4.4.3). As for the SWP$_{OPT}$ scheduling strategy, the goal is to have work packages with an equally distributed workload. The idea for SWPT$_{OPT}$ is to use the MULTIFIT heuristic, too, combined with its dynamic scheduling on each computing node.

As an example, Figure 4.21(a) shows the resulting work package runtimes T_{WP_j} for $\hat{f}_{ex1}$ (Equation (3.1)) using SWPT$_{OPT}$. Now, the work package runtimes T_{WP_j} are almost equal. For $\hat{f}_{ex2}$ (Equation (3.2)), the work package runtimes T_{WP_j} are also very well balanced, as Figure 4.21(b) shows. Compared to the plots in Figure 4.7 for the SWPT scheduling, this is a major improvement.

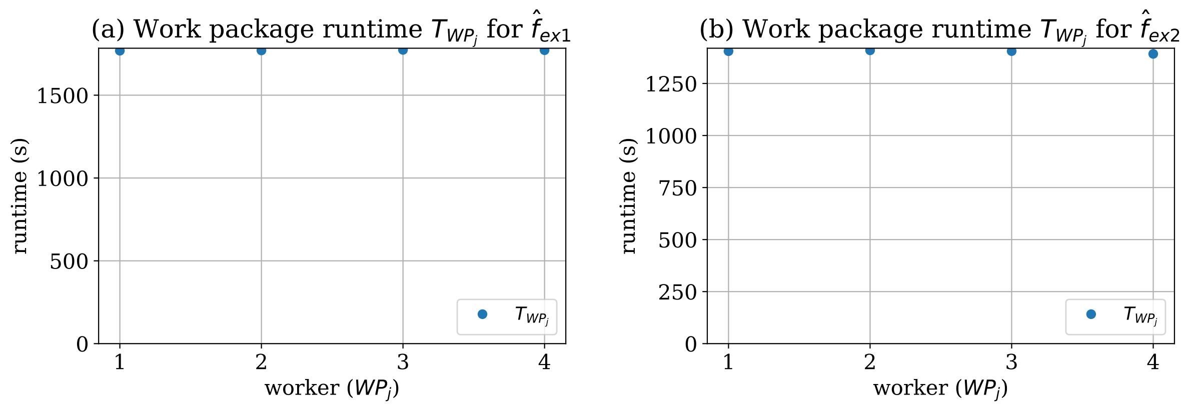


Figure 4.21: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve each work package WP_j using SWPT$_{OPT}$ scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

By using SWPT$_{OPT}$ scheduling, the resulting work packages runtimes T_{WP_j} are almost equally distributed. Due to MULTIFIT which constructs the work packages WP_j with almost the same workload and the dynamic scheduling on node level, a good load-balancing can be achieved.

4.6.3 Optimised dynamic work packages

With the optimised dynamic work packages DWP$_{OPT}$ scheduling, the DWP standard scheduling (Section 4.4.4) is improved. DWP$_{OPT}$ starts with the initial (unordered) list of collocation points z_i and sorts the entries according to their predicted runtime $\mathbb{T}^i_S$. This is known as the longest processing time first (LPT) order[7]. This ordered list is then used by the MPI pool that starts distributing the work with the first entry (that has the longest predicted runtime). The worst-case factor $R_m(LPT)$ [14] reads as follows

$$R_m(LPT) = \frac{4}{3} - \frac{1}{3m} \ , \tag{4.15}$$

with m representing the individual computing units. In the worst-case, $\widehat{T}_{Prop}$ multiplied by $R_m(LPT)$ has to be assumed for the propagation time T_{Prop}.

Figure 4.22 shows an example of the LPT ordering. The first row contains nine entries in their initial order. The relative width of the rectangles represents their individual runtime (broader means longer runtime). After the LPT ordering, the order looks like it is illustrated in the second row: the entries are in descending order.

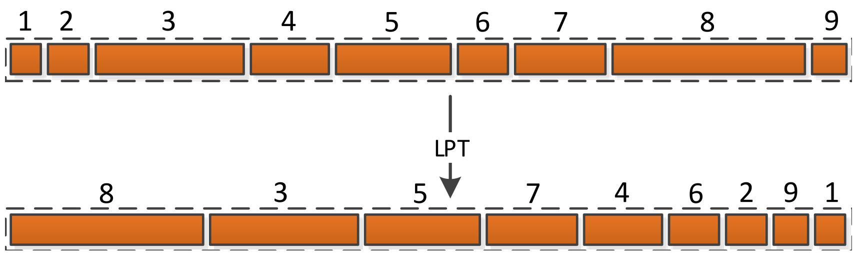


Figure 4.22: Example of the resulting list of work items with the LPT order. Each rectangle (entry) represents a work item and its relative width is the length of the runtime. The broader an entry, the longer its runtime. The first row shows the unordered entries and the second row the resulting ordered list.

DWP$_{OPT}$ is also evaluated with the two test functions: In Figure 4.23(a), the resulting work package runtimes T_{WP_j} for $\hat{f}_{ex1}$ (Equation (3.1)) are plotted. DWP$_{OPT}$ can slightly improve the workload in this example, compared to the DWP scheduling in Figure 4.9(a). For $\hat{f}_{ex2}$ (Equation (3.2)), Figure 4.23(b) shows the results, which also results in slightly better-balanced work packages (compared to Figure 4.9(b)).

The optimised dynamic work packages (DWP$_{OPT}$) scheduling can also improve the dynamical creation of the work packages to produce a better-balanced workload among the computing units. The improvement seems to be not as huge as for the other optimised scheduling strategies (SWP$_{OPT}$ and SWPT$_{OPT}$). However, due to the dynamic part, the underlying DWP standard scheduling produces already better-balanced work packages than SWP and SWPT.

[7] The MULTIFIT heuristic, that is used in SWP$_{OPT}$ and SWPT$_{OPT}$, also sorts the collocation points z_i in descending order as a first step. Hence, MULTIFIT also uses LPT.

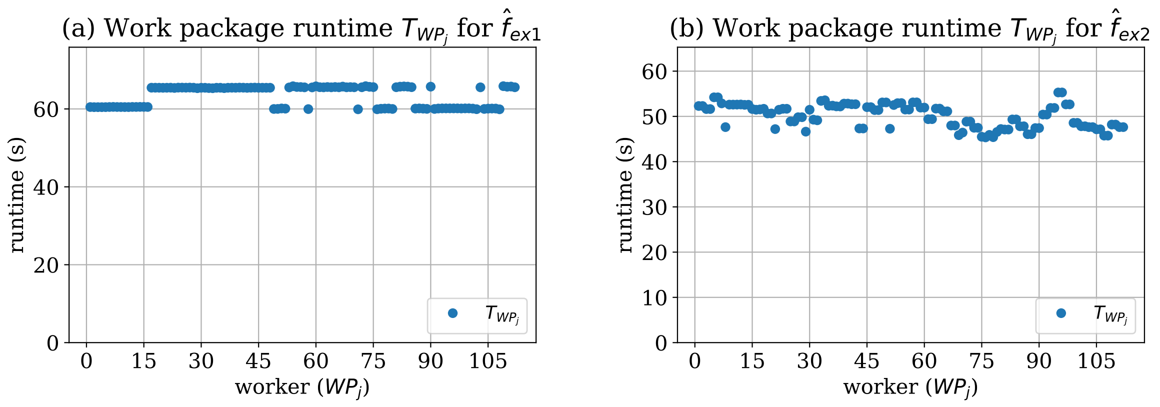


Figure 4.23: Visualisation of the runtime T_{WP_j} (working time) in seconds to solve the dynamically emerged work packages WP_j using DWP$_{OPT}$ scheduling. In (a), the working time for the UQ simulation with $\hat{f}_{ex1}$ (Equation (3.1)) as the model is shown, and in (b), the runtime with $\hat{f}_{ex2}$ (Equation (3.2)) as the model.

4.6.4 Summary of optimised scheduling strategies

In this section, the developed scheduling strategies are compared against each other with their resulting propagation runtimes T_{Prop}[8] for the UQ simulations with the two test functions $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)). Various simulations, similar to Section 4.4.5, have been performed with $q = 4, 5, \ldots, 12$ different number of collocation points per parameter and a different number of cluster nodes $cn = 2, 3, 4, 5$.

Figure 4.24(a) contains the propagation runtimes T_{Prop} for $cn = 2$ for $\hat{f}_{ex1}$ (Equation (3.1)) using all six scheduling strategies. With the SWP scheduling, the T_{Prop} is about 839 seconds for $q = 12$, followed by SWPT with 622, and DWP with only 440 seconds. All optimised scheduling strategies have similar propagation runtimes like DWP, ranging from 430–444 seconds. For $\hat{f}_{ex2}$ (Equation (3.2)), as can be seen in Figure 4.24(b), SWP takes about 536 seconds, and the others about 344 to 373 seconds, for $q = 12$.

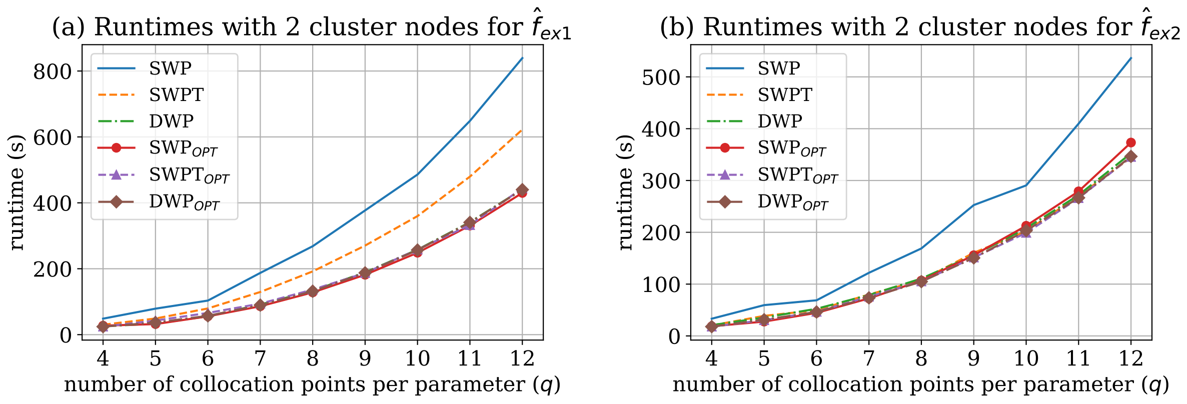


Figure 4.24: Measured propagation runtimes T_{Prop} for two cluster nodes ($cn = 2$) and a varying number $q = 4, 5, \ldots, 12$ of collocation points per parameter: (a) is the plot with the three standard scheduling strategies and their optimised versions for $\hat{f}_{ex1}$ (Equation (3.1)), and (b) for $\hat{f}_{ex2}$ (Equation (3.2)).

[8]The resulting propagation runtimes T_{Prop} are slightly different in certain cases in this thesis, compared to [103], due to some small measurement errors during the time of writing the article.

The optimised scheduling strategies with $cn = 5$ (and also $cn = 3, 4$, which are not plotted[9]) behave qualitatively similar to $cn = 2$, as can be seen in Figure 4.25(a) for $\hat{f}_{ex1}$ (Equation (3.1)) and in Figure 4.25(b) for $\hat{f}_{ex2}$ (Equation (3.2)).

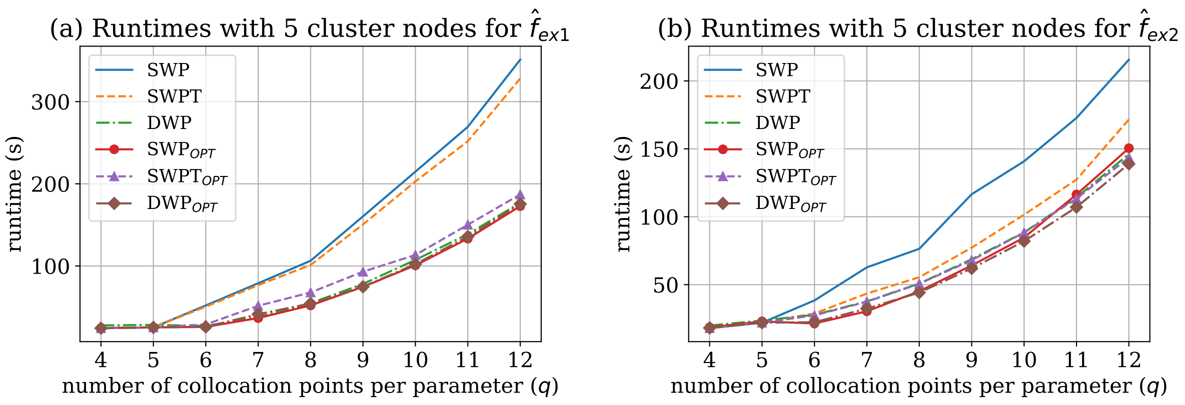


Figure 4.25: Measured propagation runtimes T_{Prop} for five cluster nodes ($cn = 5$) and a varying number $q = 4, 5, \ldots, 12$ of collocation points per parameter: (a) is the plot with the three standard scheduling strategies and their optimised versions for $\hat{f}_{ex1}$ (Equation (3.1)), and (b) for $\hat{f}_{ex2}$ (Equation (3.2)).

Table 4.7 lists the propagation runtimes T_{Prop} for $q = 10, 11, 12$ and $cn = 2, 5$ for the two example functions. It can be seen that in almost all cases, the optimised scheduling strategies produce smaller propagation runtimes T_{Prop} than their corresponding standard scheduling strategy. The measured T_{Prop} values for all performed variations in cn and q are listed in Appendix A.3.

			Scheduling strategies (time in seconds)					
$\hat{f}$	cn	q	SWP	SWPT	DWP	SWP$_{OPT}$	SWPT$_{OPT}$	DWP$_{OPT}$
$\hat{f}_{ex1}$	2	10	485.9	359.8	257.9	248.6	258.3	256.7
		11	648.6	479.5	341.4	330.9	332.4	340.5
		12	839.2	622.3	440.6	430.0	444.0	439.9
	5	10	214.7	202.8	107.4	100.8	113.4	102.2
		11	269.3	251.8	138.7	133.3	150.2	135.8
		12	351.0	328.0	176.8	172.8	187.0	175.4
$\hat{f}_{ex2}$	2	10	290.5	202.1	207.9	212.6	198.9	203.3
		11	409.6	270.2	271.7	279.0	265.9	266.6
		12	536.2	344.1	352.5	373.2	345.4	346.4
	5	10	140.8	101.4	88.4	84.6	88.3	81.9
		11	172.7	127.3	114.1	116.4	113.3	107.0
		12	215.6	171.5	146.1	150.4	143.6	138.9

Table 4.7: List of propagation runtimes T_{Prop} (in seconds) for the academic example functions $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)) for $q = 10, 11, 12$ and $cn = 2, 5$.

A different view for assessing the optimised scheduling strategies is given by the speed-up plots in Figure 4.26, which compares the SWP scheduling with the optimised scheduling strategies. The speed-ups for $\hat{f}_{ex1}$ (Equation (3.1)), as shown in Figure 4.26(a), are as follows (for $q \geq 6$): SWP to SWP$_{OPT}$ is about 1.8–2.2, SWP to SWPT$_{OPT}$ about 1.5–2.0, and SWP to DWP$_{OPT}$

[9]All measured propagation runtimes are listed in the appendix in Table A.2 for $\hat{f}_{ex1}$ and in Table A.4 for $\hat{f}_{ex2}$.

about 1.8–2.2. For $\hat{f}_{ex2}$ (Figure 4.26(b)), the speed-ups (for $q \geq 6$) for SWP to SWP$_{OPT}$ is about 1.3–2.1, for SWP to SWPT$_{OPT}$ about 1.2–1.8, and for SWP to DWP$_{OPT}$ 1.3–2.1.

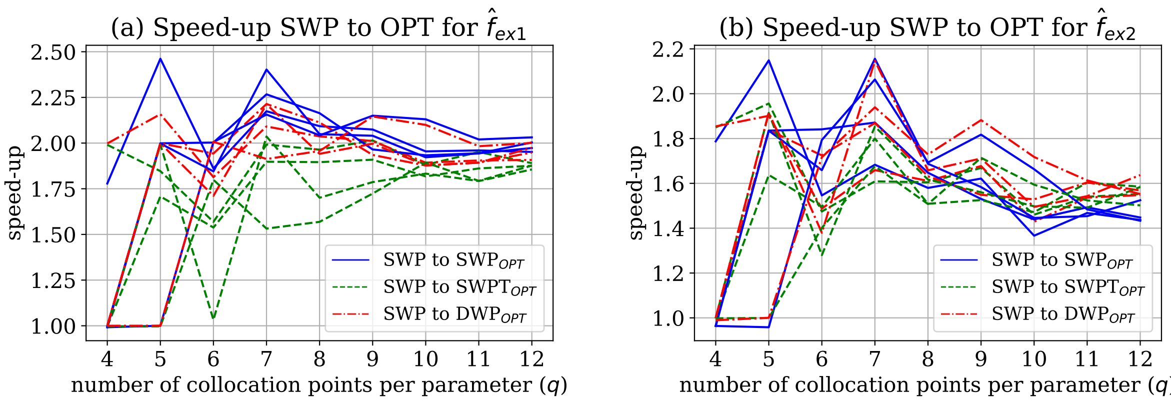


Figure 4.26: Speed-up for T_{Prop} of SWP compared SWP$_{OPT}$, SWPT$_{OPT}$, and DWP$_{OPT}$: (a) contains the speed-up for $\hat{f}_{ex1}$ (Equation (3.1)) and (b) for $\hat{f}_{ex2}$ (Equation (3.2)). Each plot contains 4 speed-up lines (of identical colour and line style) that corresponds to the different number of cluster nodes $cn = 2, 3, 4, 5$.

To compare each standard scheduling strategy with its optimised version, the speed-ups are plotted in Figure 4.27. For $\hat{f}_{ex1}$ (Figure 4.27(a)), SWP to SWP$_{OPT}$ is about 1.8–2.2, SWPT to SWPT$_{OPT}$ about 1.0–1.7, and DWP to DWP$_{OPT}$ about 1.0–1.1. Figure 4.27(b) shows the speed-ups for $\hat{f}_{ex2}$: SWP to SWP$_{OPT}$ is about 1.3–2.1, SWPT to SWPT$_{OPT}$ about 1.0–1.1, and DWP to DWP$_{OPT}$ about 1.0–1.1. The speed-ups for all tested cn and q are listed in Appendix A.4 for both academic examples.

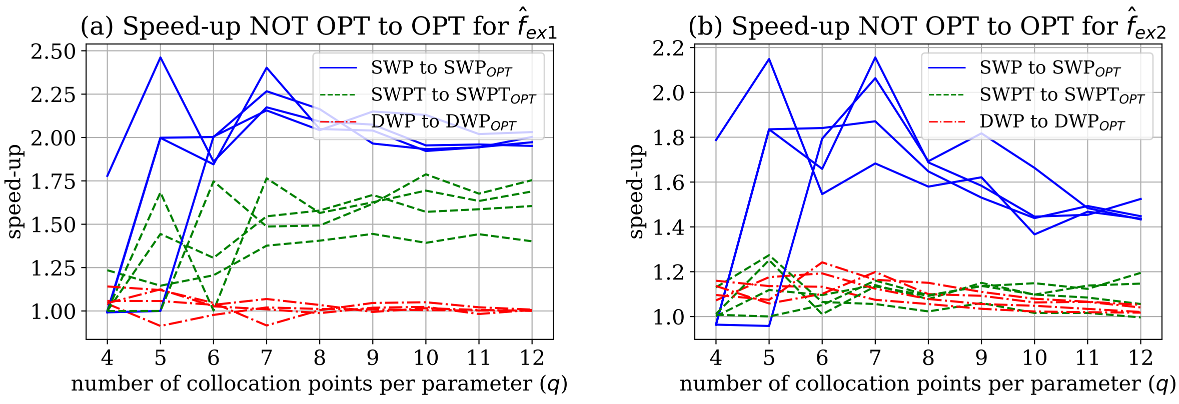


Figure 4.27: Speed-up for T_{Prop} of SWP to SWP$_{OPT}$, SWPT to SWPT$_{OPT}$, and DWP to DWP$_{OPT}$: (a) contains the speed-up for $\hat{f}_{ex1}$ (Equation (3.1)) and (b) for $\hat{f}_{ex2}$ (Equation (3.2)). Each plot contains 4 speed-up lines (of identical colour and line style) that corresponds to the different number of cluster nodes $cn = 2, 3, 4, 5$.

The plots above show impressively that the optimised scheduling strategies can further reduce the propagation time T_{Prop} compared to the SWP scheduling. The comparison of each scheduling strategy with its optimised version shows that SWP to SWP$_{OPT}$ contains the most potential for improvement and SWPT to SWPT$_{OPT}$ could also significantly reduce T_{Prop} for $\hat{f}_{ex1}$. While DWP, with its dynamic scheduling, could already significantly reduce the idling T_{WP_j}, DWP to

DWP$_{OPT}$ shows only little to no further speed-ups. But DWP$_{OPT}$ has the advantage not to exceed the worst-case boundary $R_m(LPT)$ (Equation (4.15)).

Fortunately, there is only a small overhead for loading and saving a runtime predictor rp_N, which is about 1.6 milliseconds for all scheduling strategies in both examples. The usage of the rp_N to predict all runs for all collocation points is also cheap and takes between 0.04 (for $q = 4$) and 4.76 seconds (for $q = 12$). With a maximum time of about 0.26 milliseconds to perform the resorting of the results to match the original order, the overall usage time is possible in less than 0.1 seconds for $q = 4$ and less than 5 seconds for $q = 12$. This makes the usage of the prediction mechanism almost always possible without noticeable performance losses[10] if an appropriate rp_N is available.

The decision of which scheduling strategy should be used depends on the situation and the available computing system. Table 4.9 lists different situations and recommendations of which scheduling strategy is suitable under the described conditions. As a simple rule of thumb, DWP and DWP$_{OPT}$ is a good choice if an (MPI) pool mechanism is available. In cases where no pool mechanism is available, SWPT$_{OPT}$ and SWP$_{OPT}$ should be used over SWPT and lastly SWP.

Situation	**Suggested scheduling**
Only one production run to quantify the uncertainty is possible	DWP
The runtime prediction has an acceptable or good quality	DWP$_{OPT}$, DWP, SWPT$_{OPT}$, SWP$_{OPT}$
The runtime prediction has very poor quality	DWP
There is no (MPI) pool across computing nodes available	SWPT$_{OPT}$, SWP$_{OPT}$
No information about the runtime of a model is available	DWP
The model does not vary in runtime with different input parameters values	DWP, SWPT, SWP

Table 4.9: List of possible situations and their suggested scheduling strategy.

4.7 Surrogate models

Computer simulators for complex, physical systems can result in computationally intensive implementations. The source code of some simulators may not be publicly available (especially commercial ones) and have therefore to be used as a black box. If additionally the time is limited, and the usage of the simulator through the original model exceeds the time limits, other solutions are required. Surrogate models are a good choice for such non-intrusive methods, where the model is only available as a black box. A surrogate model is a replacement for an original model whose objective is to behave qualitatively (or even quantitatively) similar—but usually with less computational effort. Therefore, the accuracy is mostly not as high as for the original model, but this can be acceptable for certain cases. Otherwise, if running the original simulator is prohibitive for a large number of parameters, running it for a small number (instead of the surrogate) will also lead to larger errors.

To construct a surrogate model, a reformulation of the equations from a microscopic model into a macroscopic model is sometimes possible. This would result in less computational effort when executing the simulation but requires access to the source code, which is assumed to be

[10]This means for both examples $\hat{f}_{ex1}$ (Equation (3.1)) and $\hat{f}_{ex2}$ (Equation (3.2)), that there is an overhead of less than $< 1\%$ compared to the total propagation time T_{Prop}. For the pedestrian dynamics scenario 2 (Section 6.4) with $cn = 5$ and $q = 12$ using DWP$_{OPT}$ scheduling, the overhead is about 0.09%.

not available in this thesis. Another feasible approach is to reduce the computational effort of a model by simplifying the time domain by using larger time steps or shorten the investigated time period. The same idea can be used for the spatial domain by reducing its resolution. This leads to the technique of reduced-order models [160, 161, 152], which is a combination of equation-driven [8] and data-driven [143, 141, 144] approaches.

If the time limit for one simulation run is too small, then it is even more so for many simulations runs as required in forward uncertainty quantification. Therefore, surrogate models are frequently used in UQ to reduce the computational effort (see [81, 101, 118, 155, 142, 141]). As shown in Figure 4.28, the surrogate model is used for the propagation of the uncertainty, instead of the original model.

A review of different surrogate model techniques for UQ can be found in [152] for the context of the water resources field. The techniques are categorised in (a) response surface surrogates and (b) low-fidelity physically-based surrogates. Response surface surrogates are called data-driven because they are constructed by observing the output of the original model. The low-fidelity physically-based surrogates use a coarser time or spatial grid, a reduced-order setting, or a different model formulation. A further categorisation can be found in [2], which also ranks non-intrusive general polynomial chaos expansions as a data-driven surrogate model approach. However, the gPCE itself as a surrogate model can not deal well with changes in the probability distributions and its parameter values for the uncertain parameters of the models. If there are any changes and a certain "accuracy" is required, the whole gPCE has to be recreated. Because a gPCE usually relies on certain evaluation points (e.g. collocation points z_i), it cannot deal with changes in these points. Therefore, the whole UQ simulation has to be repeated.

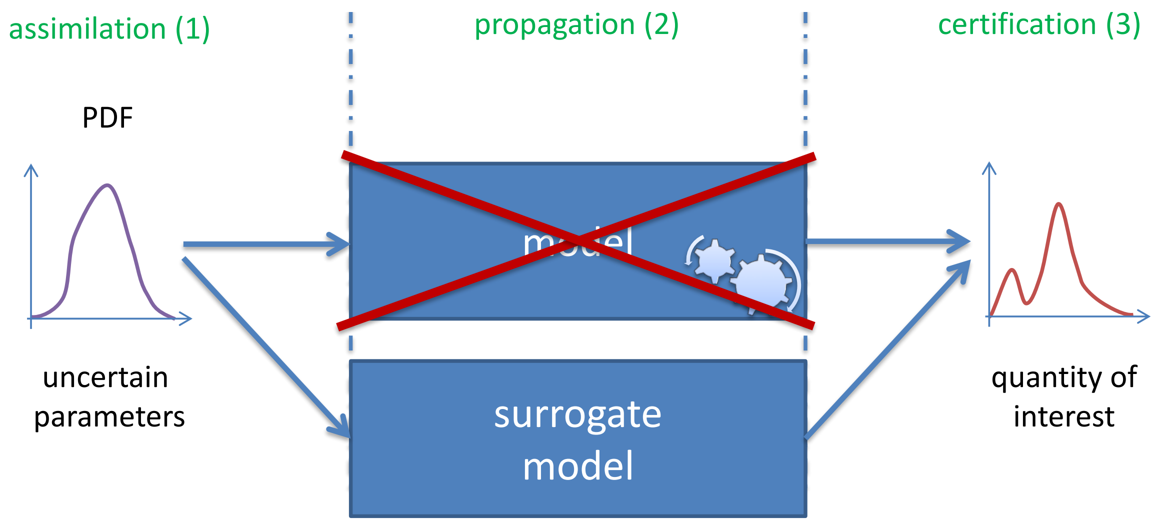


Figure 4.28: Illustration of the forward UQ method with a surrogate model instead of the original model.

In the next sections, a data-driven surrogate method is described, which is then used in UQ simulations to quantify the uncertainty efficiently. Examples of this technique in combination with UQ can be found in [28] and a scenario with detailed error measurements and runtime comparisons in this thesis in Section 6.5.

4.7.1 The closed observables surrogate model

In this thesis, a dynamic, data-driven surrogate model approach with closed observables (COSM) is used. The COSM is constructed by observing the values of interest of the original model. A surrogate model is called closed on its observables, if it can produce a subsequent value on a given point, without the need to query the original model. This is achieved through Takens' delay embedding theorem [190], which ensures to have a map to future values for each data point. To reduce the dimension of the embedding space, the COSM uses diffusion maps [15]. This produces surrogate models with reduced intrinsic states that can be saved with little required space. An introduction to the COSM can be found in [27, 24], while applications are described in [25, 28], which shows comparable results for smooth models.

To observe the values of interest from a model and to construct a corresponding surrogate model, the procedure of [27] is used. Within the surrogate, all information about the VoIs is "stored". This has the major advantage that the surrogate model can be used several times with different probability distributions, without the need for adding further observations of the original model. The surrogate can completely work on its own once it is generated.

4.7.2 Uncertainty quantification using the closed observables surrogate model

For non-intrusive UQ scenarios, it is feasible to use a closed observables surrogate model for the forward propagation of uncertainties. Figure 4.29 illustrates the necessary process for that. The COSM is divided into two phases: the offline and the online phase.

In the offline phase, the original model is sampled several times by specifying appropriate values for the input parameters and collecting the output values (VoIs). For each uncertain input parameter $P_1, P_2, \dots, P_M$ of the model, values have to be generated. In this thesis, the parameter space is sampled with a full grid. It is important to specify the sampling interval and the rate in such a way that the whole parameter space is covered. For all other input model parameters that are not defined to be uncertain, the parameters are kept fixed at predefined values. For each model run, the values of interest $VoI_1, VoI_2, \dots, VoI_V$ are collected for every timestep of the model. The surrogate model uses the generated input parameter values and the collected time series of VoIs to construct the closed observables with its intrinsic states and saves the closed observables into a file. From now on, the original model is not required anymore. Only if a different parameter space has to be covered, if the number of the parameter changes, or additional VoIs are required, the original model has to be sampled again.

The propagation of the uncertainties with the COSM can start after the closed observables are created. In the offline phase, the COSM loads the closed observables from a file into the memory. Once the data are loaded, the surrogate model is ready to reproduce the VoIs according to some given input parameter values. The usage of the COSM for the propagation of the uncertainties using the stochastic collocation with the pseudo-spectral approach[11] is then straightforward: The technical phase of the assimilation is as usual and creates the collocation points z_i, where every z_i contains specific values for every input parameter $P_1, P_2, \dots, P_M$ of the surrogate model, exactly as in the case of the original model. The difference is in the propagation phase, where the surrogate model is called instead of the original model—but with significantly less computational effort. For that, Equation (2.7) is rewritten into

$$U(x,t,\boldsymbol{\zeta}) \approx \tilde{u}_N(x,t,\boldsymbol{\zeta}) := \sum_{j=0}^{N} \tilde{c}_j(x,t) \cdot \Phi_j(\boldsymbol{\zeta}) \ , \tag{4.16}$$

and the corresponding coefficients $\tilde{c}_j$ are computed with

$$\tilde{c}_j(x,t) \approx \frac{1}{\gamma_j} \sum_{i=1}^{Q} s(x,t,z_i)\Phi_j(z_i)w_i \ , \tag{4.17}$$

[11] It is also possible to use the COSM with other non-intrusive UQ methods such as point collocation (Section 2.2.3) or Monte Carlo (Section 2.2.1).

where $s(x, t, z_i)$ denotes the surrogate model. After all VoIs are collected, the certification phase can take place to approximate the coefficients $\tilde{c}_j$, construct the gPCE $\tilde{u}_N(x, t, \boldsymbol{\zeta})$, and finally to calculate the QoIs. It is crucial to evaluate the accuracy of the results, before making decisions based on the QoI results that are determined with the COSM. How this is done in this thesis is part of the next section.

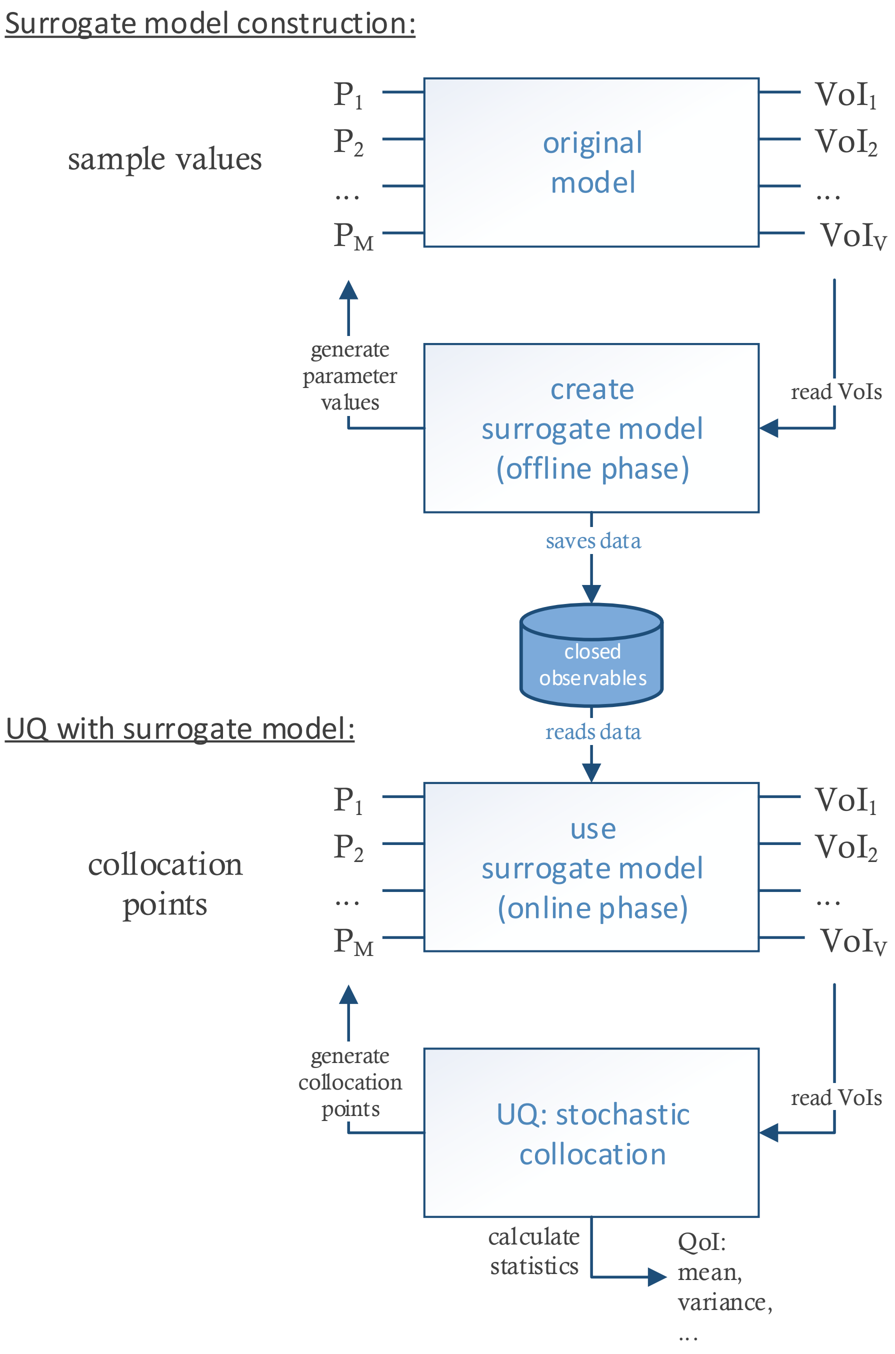


Figure 4.29: Illustration of the process to create a COSM and to use it in a UQ simulation. The parameters of the model are denoted with $P_1, P_2, \ldots, P_M$ and the values of interest with $VoI_1, VoI_2, \ldots, VoI_V$, which is the output of the model. The data flow between the steps is visualised with the arrows.

4.7.3 Determining the quality of a UQ simulation using the COSM

To assess the quality of a UQ simulation using the COSM, a comparison of the quantities of interest is proposed. The required process for that is illustrated in Figure 4.30.

In the previous (slow) path, the original model is used for the propagation of the uncertainties and the QoIs with the preferred statistical moments (Section 2.2.2) like the mean μ_o (Equation (2.17)) and the variance σ_o^2 (Equation (2.18)) are determined.

The proposed (fast) path with the COSM is also performed with exactly the same setting for the uncertain parameters $\boldsymbol{\zeta}$ and the resulting collocation points z_i. As for the original model, the VoIs are collected and the counterparts of the preferred statistical moments like the mean μ_s and the variance σ_s^2 are determined.

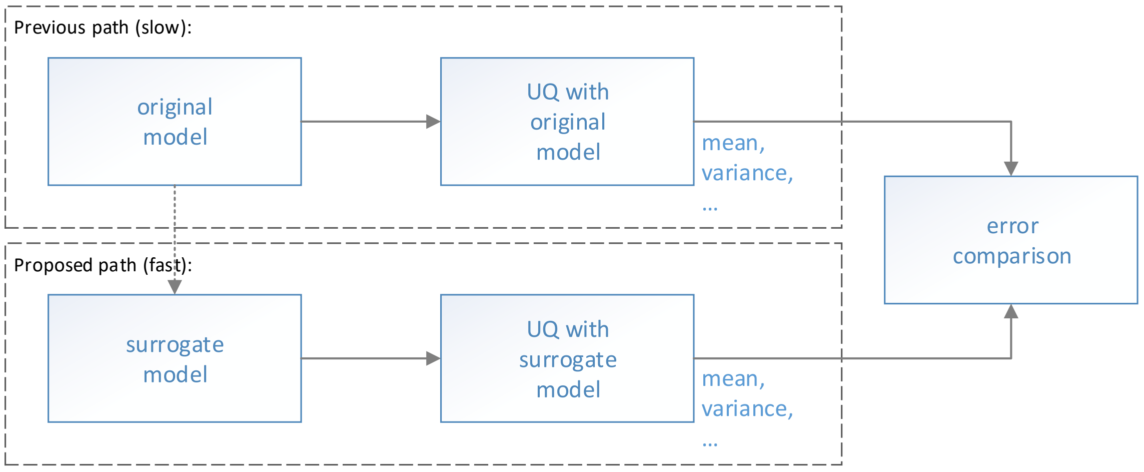


Figure 4.30: Illustration of the proposed error comparison process of the original model (previous path) with the surrogate model (proposed path) for UQ simulations using the COSM.

Once the QoIs with the statistical moments have been determined for both paths, the quality can be assessed. Similar to the error measurements defined in Section 4.5.2 to compare the runtime, here the absolute mean error ϵs_i is defined as

$$\epsilon s_i := |\mu_{oi} - \mu_{si}| \ , \tag{4.18}$$

to assess the mean μ from the original model μ_o compared to the mean μ_s of the surrogate model. The mean μ is indexed with i because the error is determined for many time steps $i = 1, 2, \ldots, \tau$.

Remark: For this definition, the surrogate model should use the same time steps to generate the VoIs as the original model. Otherwise, a mapping between different time steps is required, which can introduce an additional source of error. If required, such a synchronisation approach of time series data can be found in [106].

For the relative error,

$$\epsilon s_{i,\text{rel}} := \frac{\epsilon s_i}{\max(|\mu_o|)} = \frac{|\mu_{oi} - \mu_{si}|}{\max(|\mu_o|)} \tag{4.19}$$

is used. Note that $\max(|\mu_o|)$ is used instead of $\max(|\mu_{oi}|)$. This is more useful for time series data because a single time step i can have a high relative error, while the whole time series is well approximated. This also means that the whole time series should be in focus, not a single point in time. For the comparison, also the mean of the absolute error $\mu(\epsilon s)$ and the $L^2(\epsilon s)$ error norm are useful, defined as,

$$\mu(\epsilon s) := \frac{1}{\tau}\sum_{i=1}^{\tau} \epsilon s_i \ , \qquad (4.20) \qquad L^2(\epsilon s) := \sqrt{\frac{1}{\tau}\sum_{i=1}^{\tau} \epsilon s_i^2} \ , \qquad (4.21)$$

and their relative counterparts,

$$\mu(\epsilon s_{\mathrm{rel}}) := \frac{1}{\tau}\sum_{i=1}^{\tau} \epsilon s_{i,\mathrm{rel}} \ , \qquad (4.22) \qquad L^2(\epsilon s_{\mathrm{rel}}) := \sqrt{\frac{1}{\tau}\sum_{i=1}^{\tau} \epsilon s_{i,\mathrm{rel}}^2} \ . \qquad (4.23)$$

4.7.4 Summary of surrogate models

With the COSM, there is now a concept available that can be used for real-time decision-making under uncertainty. Using the COSM in UQ simulations allows to speed up the propagation phase of two to three orders of magnitude depending on the runtime of the original model, which allows to obtain the results very fast: e.g. in seconds to minutes compared to hours.

Once the closed observables are constructed by sampling the original model, they can easily be saved into a file. By loading the closed observables from a file, they can be used in UQ simulations as a surrogate for the original model. The big advantage is the fast evaluation and the possibility to use the closed observables even if the probability distribution or its parameters changes, without the need of recreating the closed observables.

With the defined error measures, it is possible to determine the quality of the COSM to decide whether the loss of accuracy is still good enough to derive decisions from the results.

The COSM proved its efficiency in [28] and in the case study of Section 6.5.

5 The UQEF software framework

During this thesis, the uncertainty quantification execution framework (UQEF) has been developed for forward uncertainty quantification. The programming language is Python[1], and the offered API for users, therefore, is also in Python. The framework relies on Chaospy but could be—with some refactorings—opened to other underlying UQ toolboxes. UQEF further has, among others, dependencies to joblib [47] and mpi4py [113].

UQEF addresses some open points when using Chaospy directly that are outlined in Section 2.4.2 and the required efficiency aspects that are defined in Section 2.5. In Chaospy, the propagation phase is very much up to the developer, and UQEF closes this gap by providing different scheduling and scalability features. Its focus is on scalability and portability from development systems to compute (cluster) systems. For that, it implements the runtime prediction and automatic scheduling mechanisms described in Chapter 4. This enables the users of UQEF to apply these features for their own UQ simulations without focusing on the computing system environment and having strong support for the propagation phase (comp. Figure 2.2) in their UQ simulations.

UQEF is extensively used in all scenarios of Chapters 6 and 7 of this thesis: It proved its functionality with the academic test models (Section 3.1), with Vadere (Section 6.1) a pedestrian dynamics simulator, and with the water balance model simulator LARSIM (Section 7.1).

This chapter gives an overview on the software pieces of UQEF, shows its usage and its functionality, demonstrates how to do the configuration, and how to extend it with custom behaviour as well as extend it to other computing systems.

5.1 Overview and software architecture

In this section, a general overview of and a gentle introduction to the UQEF software framework is given. The following list gives an overview of the UQEF feature categories. Each feature category is described in detail in a corresponding section.

- Custom models (Section 5.2)
- Custom statistics (Section 5.3)
- Parametrisation of a UQ simulation (Section 5.4)
- Support for different UQ methods (Section 5.5)
- Scheduling and solver strategies (Section 5.6)
- Support for parameter studies (Section 5.7)
- Automatic runtime measurements, predictions, and optimised scheduling (Section 5.8)

UQEF is hierarchically organised as a Python package with separate modules for each different aspect. This object-oriented approach allows the separation of concerns and provides the opportunity to extend UQEF easily. Figure 5.1 illustrates the architecture of UQEF as a UML package and class diagram. The classes are explained in detail in the subsequent sections. *UQsim* is the main class for the users, which acts as a facade (the facade pattern is described in [48, p. 185–194]). It uses all other packages and relies on the interfaces and not on the concrete subclasses. The *simulation* package contains the *Simulation* interface, which is the base class for concrete simulation types, e.g. the Monte Carlo method (*McSimulation*) or the stochastic

[1]At the start of this thesis, the Python version was 2. But all sources are now written with Python version 3 and the backward compatibility to Python 2 is not tested anymore.

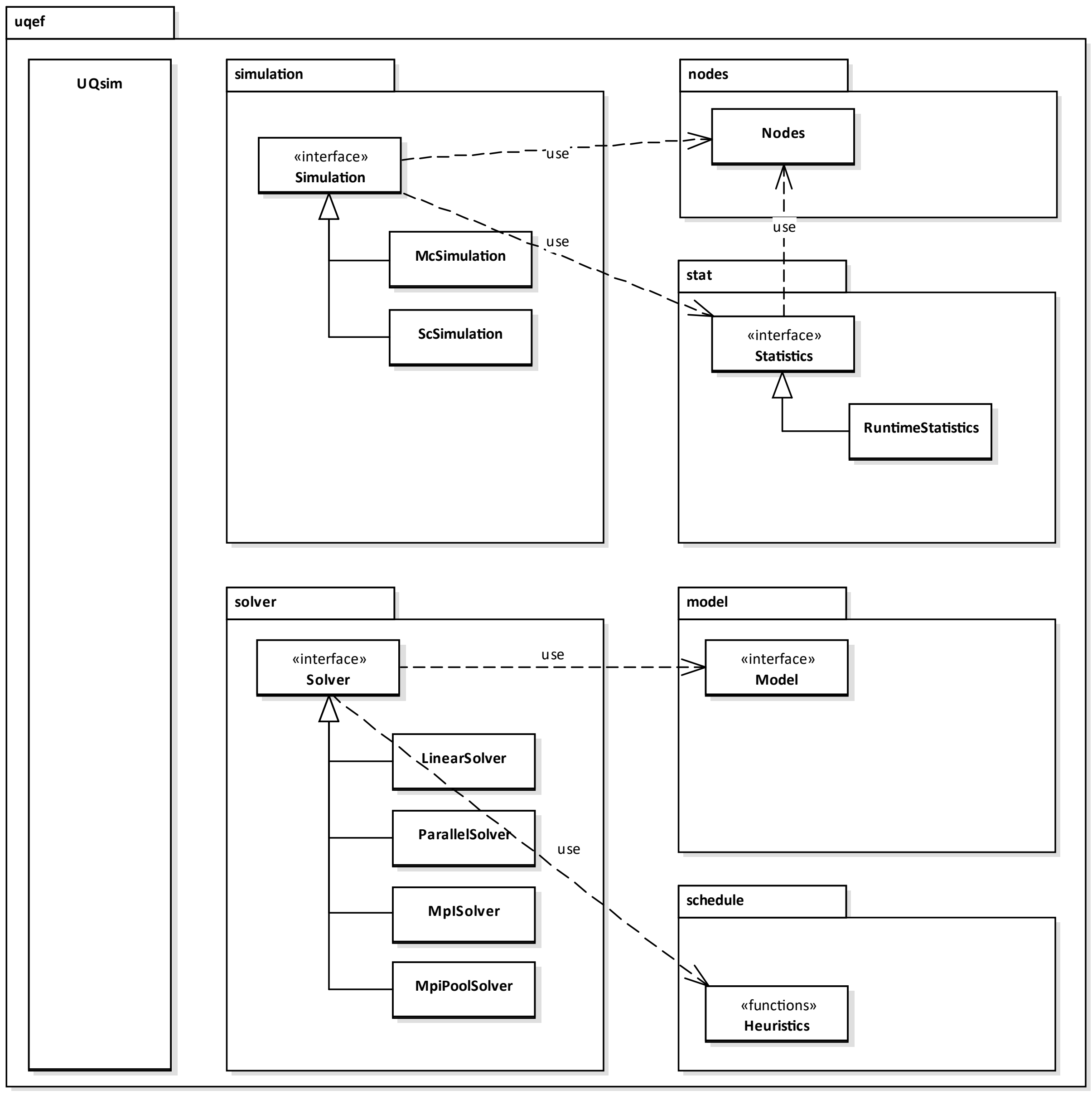


Figure 5.1: Illustration of the UQEF architecture: Combined UML package and class diagram that shows the main structure and elements with the dependencies of the UQEF framework.[2]

collocation with pseudo-spectral approach (*ScSimulation*). Inside the *nodes* package, there is the *Nodes* class which is responsible for configuring and generating nodes for the UQ simulation. The *Statistics* interface is inside the *stat* package and is derived by concrete statistic classes. The *solver* package contains the *Solver* interface with its concrete implementations: *LinearSolver*, *ParallelSolver*, *MpiSolver*, and *MpiPoolSolver*, which controls the scheduling of the individual model runs. The *Solvers* use the *Model* interface from the *model* package for the propagation, which is the extension point for users to include their own models. Finally, the *schedule* package contains the *Heuristics* file with functions for common scheduling heuristics (comp. Section 4.6.1).

Figure 5.2 illustrates a typical program architecture for UQEF simulation codes. Inside a

[2]For a better overview, the usage arrows from the *UQsim* class to all other packages (interfaces) where not drawn. Further, the diagram does not contain the functions and members of the classes, these are in detail explained in the subsequent sections.

`uq_simulation.py` Python script, there is one instance of a *UQsim* class. The object of the *Nodes* class, which is provided by the *UQsim* object, contains parameter information for each uncertain and fixed model parameter. *ConcreteModel* is a placeholder for a concrete class that is derived from the *Model* interface. It contains the actual model source code directly, or the system calls to perform a single run of the external model. The *ConcreteStatistics* object is again a placeholder for a concrete class derived from the *Statistics* interface. It is the counterpart of the *ConcreteModel* class which contains all the source code to evaluate the VoIs statistically and to generate the resulting numbers for the QoI(s) as well as the generation of tables and plots.

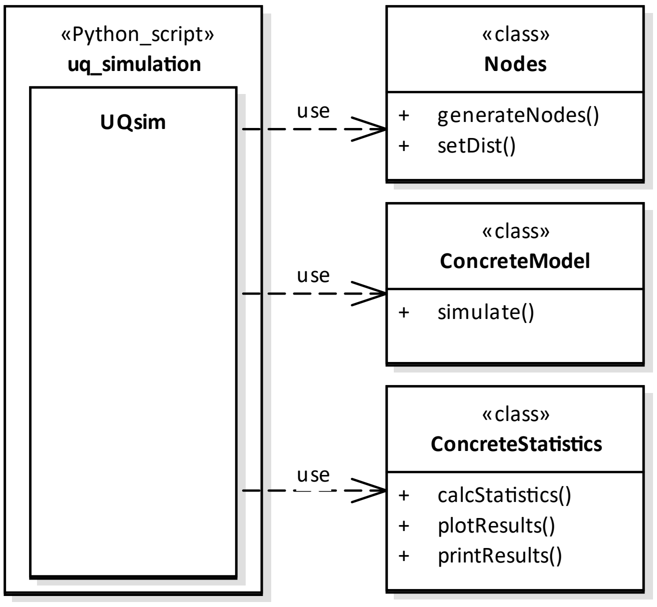


Figure 5.2: Illustration (UML class diagram) of a common simulation program architecture when using the UQEF framework.

Listing 5.1 represents a first, common example of a UQ simulation script (`uq_simulation.py`) using UQEF. Lines 1–2 include Chaospy and UQEF. In line 4, an instance (`uqef`) of the *UQsim* class is created. If nothing specific is configured (as in this example), *UQsim* uses per default stochastic collocation with the pseudo-spectral approach (see Section 2.2.2) and a *TestModel* (which simply adds all parameter values, or it can be interpreted as the identity function in case of one parameter) with the corresponding *TestModelStatistics*. Line 8 defines q, the number of collocation points for each uncertain parameter, and line 9 the order P which specifies the highest order of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE Equation (2.7). The distributions for the uncertain parameters are defined in lines 12–15: first, the names for the parameters `"uncertain_param_1"` and `"uncertain_param_2"` are specified. Then, the distributions are set, here, a normal (Gaussian) $N(0, 1)$ distribution for `"uncertain_param_1"`, and a uniform $U(-1, 1)$ distribution for `"uncertain_param_2"`. With line 18, all the objects within the `uqef` object are initialised and prepared for the actual UQ simulation (propagation) which is started in line 21. All the statistical calculations for the QoIs, the printing and plotting, and the saving of the results is done in lines 24–27. Finally, line 30 gives all the objects the chance to do some cleanup and tear down, before the `uq_simulation.py` script ends.

This is already a powerful basis for own UQ simulations with UQEF. In the following sections, it is explained how to add custom models and corresponding statistical evaluation source codes, as well as adding advanced configurations and parametrisations, and how to use and configure the automatic runtime prediction and scheduling features.

```python
import chaospy as cp
import uqef

# instantiate UQsim
uqsim = uqef.UQsim()

# args
uqsim.args.sc_q_order = 3 # order (q) number of colloc. points for each param.
uqsim.args.sc_p_order = 1 # order (P) highest order of orth. poly. of the gPCE

# initialise uncertain parameters:
if uqsim.is_master():
    uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
    uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(0, 1))
    uqsim.simulationNodes.setDist("uncertain_param_2", cp.Uniform(-1, 1))

# setup
uqsim.setup()

# start the simulation
uqsim.simulate()

# statistics:
uqsim.calc_statistics()
uqsim.print_statistics()
uqsim.plot_statistics()
uqsim.save_statistics()

# tear down
uqsim.tear_down()
```

Listing 5.1: First illustrative source code example (`uq_simulation.py`) that shows the basic usage of UQEF using the stochastic collocation with the pseudo-spectral approach (see Section 2.2.2).

5.2 Custom models: embedding model codes

An essential aspect is to use custom models with UQEF. For that, the *Model* interface exists, which has to be derived. It is recommended to derive a subclass per model that contains all the source code to run the model code.

Method signature	Abstract	Return value
`def normaliseParameter(self, parameter):`	yes	List of scalar values
`def assertParameter(self, parameter):`	yes	`None`
`def prepare(self):`	yes	`None`
`def run(self, i_s, parameters):`	yes	List of tuple(VoI, runtime)
`def timesteps(self):`	yes	List of timesteps

Table 5.2: Functions of the UQEF *Model* interface class inside the *model* package.

Table 5.2 lists the main functions of the *Model* interface with their return values. The "Abstract"-column indicates that the method is abstract and it has to be overwritten by the subclasses.

In the following list, the functions of the *Model* interface are explained:

`def normaliseParameter(self, parameter):`

Can be used to normalise or correct values in `parameter`. The `parameter` is a list that contains a scalar value for each defined parameter in `uqsim.simulationNodes` (*Nodes* class). This function is called from the active *Solver* (see Section 5.6) before the `assertParameter()` function is called.

Remark: The order of the parameter values in `parameter` corresponds to the order of the added parameters in the `uqsim.simulationNodes` object within the *UQEF* class object (e.g. lines 13–15 in Listing 5.1).

`def assertParameter(self, parameter):`

Can be used to assert that the values in `parameter` are valid for the model. Because the parameter values are usually sampled from a probability distribution or generated from the quadrature scheme, it can happen, that non-valid values for the model parameter are generated, which can be detected here. For a developer, it is recommended to use the Python `assert` statement to perform the check, which throws an *AssertionError* if a non-valid parameter value is detected. This function is called from the active *Solver* before the parameters are distributed to the workers.

`def prepare(self):`

Can be used for custom actions to prepare the model (e.g. copy files). This function is called from the active *Solver* before the `run()` function is called.

`def run(self, i_s, parameters):`

Is the actual method that starts the model run and returns the values of interest (VoIs) for every parameter set in `parameters`. Usually, `parameters` contains only one set of parameter values and in the `i_s` list the corresponding index. Each parameter set has its own index to identify a specific model run correctly. Some models, especially surrogate models like the COSM (see Section 4.7.2), perform better if they are loaded once and then proceed many runs directly one after another, which is the reason for this chunking mechanism. The index can also be used for models to create separate, identifiable folders with all the relevant input and output files for a specific model run. This function is called from the active *Solver*.

`def timesteps(self):`

Returns a list with the timesteps (scalar value) of the model. If only one timestep exists, a list with one element has to be returned. The return values of the `run()` function have to contain at least one value for each timestep.

Listing 5.2 contains a template for a custom model implementation. The functions `normaliseParameter()`, `assertParameter()`, and `prepare()` are only defined, but do not contain specific code. In the `run()` function, there is a loop over all `parameters` (and the corresponding indices `i_s`). Inside the loop, the model source code can be implemented directly, or an external model can be called via system calls to calculate the VoI. The VoI can be a single scalar value or a list of values, in the case of multiple VoIs, or one value for each timestep (additionally, a list of lists or a list of dictionaries is also supported). For the runtime measurements and optimisations, it is required that for every run, the runtime is determined and added to the `results`. `timesteps()` returns a list with one scalar value which means the *CustomModel* only has one timestep.

```python
import uqef
import time


class CustomModel(uqef.model.Model):
    def __init__(self):
        uqef.model.Model.__init__(self)

    def normaliseParameter(self, parameter):
        return parameter

    def assertParameter(self, parameter):
        pass

    def prepare(self):
        pass

    def run(self, i_s, parameters):
        results = []
        for ip, parameter in zip(i_s, parameters):
            start = time.time()

            # calc value of interest (VoI) with parameter set
            # or call external model application with parameter values
            value_of_interest = 0  # retrieve value of interest (VoI) from model

            # measure runtime
            runtime = time.time() - start

            # collect the output (voi and runtime)
            results.append((value_of_interest, runtime))

        return results

    def timesteps(self):
        return [1]
```

Listing 5.2: Example template for a custom model implementation with UQEF.

To use the *CustomModel* in own scripts, e.g. as in the UQEF source code example (Listing 5.1), only small extensions are required. Listing 5.3 shows how to choose the *CustomModel* (line 2) and to register it at the `uqsim` object (line 5). These lines should be added before `uqsim.setup()` is called. The `uqsim` object holds all known models in a dictionary, which is identified by a model name, here `"custom_model"`. The `model` dictionary gets a lambda function as its value because, for each run, a separate object of the model (*CustomModel*) is created. If required, the constructor of *CustomModel* can be used for further initialisation purposes of the model.

```python
1 # choose custom_model
2 uqsim.args.model = "custom_model"
3
4 # register model
5 uqsim.models.update({"custom_model": lambda: CustomModel()})
```

Listing 5.3: Selection and registration of a custom model (*CustomModel*) on an object (`uqsim`) of a *UQsim* instance with UQEF.

5.3 Custom statistics: computation of model-specific statistics

The *Statistics* interface is the counterpart to the *Model* interface. It represents the technical part of the certification phase, including the evaluation of the QoI as well as the generation of the coefficients c_j for gPCE based methods. A general implementation for the generation of the QoI statistics is currently not possible, because of the broad spectrum of different types and shapes of the data of different models. Therefore, the coding of the QoI evaluation, the generation of the numbers, data tables and the plots is up to the user. But, the *Statistics* interface is already provided by UQEF, for a clear structure and to support load/save mechanisms as well as, e.g. redo plotting without a recalculation of the QoI. For the generation of the QoI, the whole Chaospy functionality is available as described in Section 2.4.2. In Table 5.4, the main functions with their return values are listed.[3]

Method	Abstract	Return value
`def calcStatisticsForMc(...):`	no	`None`
`def calcStatisticsForSc(...):`	no	`None`
`def printResults(...):`	no	A string that represent the results.
`def plotResults(...):`	no	`None`
`def plotAnimation(...):`	no	`None`

Table 5.4: Functions of the UQEF *Statistics* interface class inside the *stat* package.[3]

In the following enumeration, the functions of the *Statistics* interface are explained:

```
def calcStatisticsForMc(self, rawSamples, timesteps,
                        simulationNodes, numEvaluations, order,
                        regression, solverTimes,
                        work_package_indexes,
                        original_runtime_estimator=None):
```

This function can be overwritten by concrete *Statistics* classes to support Monte Carlo statistic calculations. Here, the whole Numpy and the Chaospy functionality for the certification phase (see Section 2.4.2) can be used. The goal is to calculate the required QoIs (statistical moments) here because this depends on the UQ method and to store the data (as members) in the *Statistics* object in a way, such that it can be used later in the print and plot functions, which should not depend on a particular UQ method. This offers code reuse, especially if different UQ methods are used for the same model.

`rawSamples` is a list and contains all the VoIs that are returned from the *Model* during the propagation. The order of the VoIs is already restored and matches the original order of the generated nodes from the *Nodes* class. The `timesteps` list contains all timesteps that the model used for the propagation. For each timestep, the `rawSamples` list should contain one value for each VoI. The parameter `simulationNodes` contains all generated nodes from the *Nodes* class. `numEvaluations` is a scalar value with the number of evaluations that have been performed. UQEF supports the common point collocation method (Section 2.2.3), with the `order` parameter (P) for the highest order of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE. If the `regression` parameter is `True`, then point collocation is used instead of Monte Carlo.[4] In `solverTimes` all runtimes are stored (one value for each performed single run). Because of the scheduling features of UQEF, `work_package_indexes` is a list of lists with the indices that are grouped into work packages. If a runtime predictor is

[3]Because of the many function parameters these are not shown in Table 5.4.

[4]Because the common point collocation method (Section 2.2.3) relies on the same idea of sampling the probability distribution to generate sample points for the propagation as the Monte Carlo method, it is realised with the `regression` parameter at this point.

loaded from the `uqsim` object, then `original_runtime_estimator` contains the loaded gPCE object of the runtime rp_N. Further details are documented in Section 5.8. This function is called from the *UQsim* class when `uqsim.calc_statistics()` is called.

```
def calcStatisticsForSc(self, rawSamples, timesteps,
                        simulationNodes, order,
                        regression, solverTimes,
                        work_package_indexes,
                        original_runtime_estimator=None):
```

This function can be overwritten by concrete *Statistics* classes to support stochastic collocation statistic calculations. The meaning of the parameters is the same as for the `calcStatisticsForMc()` function, except the `simulationNodes` are generated with a quadrature rule, and the `rawSamples` contains only the VoIs that are propagated using the `simulationNodes`. If `regression == True`, then the probabilistic collocation approach (Section 2.2.3) is used. This function is called from the *UQsim* class when `uqsim.calc_statistics()` is called.

```
def printResults(self, timestep=-1):
```

This function can be overwritten by concrete *Statistics* classes to generate text that represents the results. For that, the member variables that contain the QoI values that are created by the `calcStatisticsForMc()` or the `calcStatisticsForSc()` function should be used. The return value is a string with the results, usually in tabular form. If `timestep == -1`, then the results for all timesteps should be printed, otherwise only the results for the specified timestep.

```
def plotResults(self, timestep=-1, display=False,
                fileName="", fileNameIdent="", directory="./",
                fileNameIdentIsFullName=False, safe=True):
```

This function can be overwritten by concrete *Statistics* classes to generate graphical plots that represent the results. If `timestep == -1`, then the results for all timesteps should be plotted, otherwise only the results for the specified timestep. With `display == True`, the plots are directly displayed on the screen, which is useful for the development phase. The parameters `fileName`, `fileNameIdent`, `directory`, and `fileNameIdentIsFullName` contain information about the resulting filenames for the plots. If `safe == True`, then the plots are saved permanently into files.

To generate a full path for a plot, the `generateFileName()` function of the *Statistics* class can be used:

```
fileName = self.generateFileName(fileName, fileNameIdent,
                                 directory, fileNameIdentIsFullName)

pngFileName = fileName + ".png" #for a png
pdfFileName = fileName + ".pdf" #for a pdf
```

```
def plotAnimation(self, timesteps, display=False,
                  fileName="", fileNameIdent="", directory="./",
                  fileNameIdentIsFullName=False, safe=True):
```

This function can be overwritten by concrete *Statistics* classes to generate animations (videos) that represents the results. The `timesteps` parameter is a list with the timestep values that should be used for creating the animation.

In the following Listing 5.4, an example for a custom statistics object is shown that supports the stochastic collocation with the pseudo-spectral approach.

```
 1 import numpy as np
 2 from tabulate import tabulate
 3 import matplotlib.pyplot as plotter
 4 import chaospy as cp
 5 import uqef
 6
 7
 8 class Samples(object):
 9     def __init__(self, rawSamples, numTimeSteps):
10         self.voi = []
11
12         for sample in rawSamples:
13             self.voi.append(sample)
14
15         self.voi = np.array(self.voi)
16
17
18 class CustomStatistics(uqef.stat.Statistics):
19     def __init__(self):
20         uqef.stat.Statistics.__init__(self)
21
22     def calcStatisticsForSc(self, rawSamples, timesteps,
23                             simulationNodes, order, regression, solverTimes,
24                             work_package_indexes, original_runtime_estimator):
25         nodes = simulationNodes.distNodes
26         weights = simulationNodes.weights
27         dist = simulationNodes.joinedDists
28
29         self.timesteps = timesteps
30         self.numTimeSteps = len(self.timesteps)
31
32         # extract VoI samples from rawSamples
33         samples = Samples(rawSamples, self.numTimeSteps)
34
35         OP = cp.orth_ttr(order, dist)  # creates orthogonal polynomials
36         if regression is True:
37             # generate gPCE via regression
38             self.gPCE = cp.fit_regression(OP, nodes, samples.voi)
39         else:
40             # generate gPCE via quadrature
41             self.gPCE = cp.fit_quadrature(OP, nodes, weights, samples.voi)
42
43         # extract the statistics
44         self.E_qoi                  = cp.E(self.gPCE, dist)
45         self.Var_qoi                = cp.Var(self.gPCE, dist)
46         self.StdDev_qoi             = cp.Std(self.gPCE, dist)
47         self.P10_qoi, self.P90_qoi = cp.Perc(self.gPCE, [10, 90], dist, 10**5)
48         #...
```

Listing 5.4: Example of a custom statistics implementation using UQEF.

It is used together with the *CustomModel* (see Listing 5.3). If *CustomStatistics* also implements `calcStatisticsForMc()`, then it can be used for Monte Carlo as well as stochastic collocation UQ simulations. This is very helpful if multiple methods should be evaluated. Lines 1–5 import all required packages. The *Samples* class (lines 8–15) is a small helper class to extract the VoI from the `rawSamples`, which is especially helpful for unifying multiple VoIs or in the case when not every model run used the same timesteps. `calcStatisticsForSc()` does the actual work and uses *Samples* to access the VoI easily. Lines 36–41 show how to use the `regression` parameter to decide which method should be used for the gPCE generation. The extraction of the QoI statistics on lines 44–47 is done with the generated `gPCE`. With the saved

QoI values in the member variables `E_qoi`, `Var_qoi`, `StdDev_qoi`, `P10_qoi`, and `P10_qoi`, the `printResults()` and `plotResults()` functions can do their work by using these variables.

Together with the *CustomModel* from the previous section, the *CustomStatistics* can be used in own scripts: Line 8 in Listing 5.5 shows how to register the *CustomStatistics* at the `uqsim` object. Now, a full working example with a custom model and the model-specific statistics exists.

```
1  # choose custom_model
2  uqsim.args.model = "custom_model"
3
4  # register model
5  uqsim.models.update({"custom_model": lambda: CustomModel()})
6
7  # register statistics
8  uqsim.statistics.update({"custom_model": lambda: CustomStatistics()})
```

Listing 5.5: Selection and registration of a custom model (*CustomModel*) and a corresponding *CustomStatistics* on an object (`uqsim`) of a *UQsim* instance with UQEF.

5.4 Parametrisation of a UQ simulation

The flexibility and portability of UQEF is due to the possibility to parametrise a `uqsim` instance of a *UQsim* class. The parameters can be set within a simulation script as well as from the command line as script arguments. It is also allowed to combine both variants: specify some of the parameters from the command line and some within the simulation script. For this parametrisation mechanism, the Python `argparse`[5] package is used and the `uqsim.args` object gives access to the parsed and set parameter values. In Table 5.5 all available parameters in the *UQsim* class are described, with parameter name, data type, and default value.

Parameter	Type	Default	Description
UQsim load/restore			
`"--uqsim_file"`	string	`"uqsim.saved"`	File where UQsim object is (re)stored
`"--uqsim_store_to_file"`	bool	False	Enable store UQsim object to file
`"--uqsim_restore_from_file"`	bool	False	Enable restore UQsim object from file
`"--disable_recalc_statistics"`	bool	False	Disable the recalculation of statistics
`"--disable_statistics"`	bool	False	Disable all statistical calculations including plots
Model and result directories			
`"--inputModelDir"`	string	`"."`	Folder for the input files of the model
`"--outputModelDir"`	string	`"."`	Folder for the output files of the model
`"--outputResultDir"`	string	`"."`	Folder for the statistics results (plots, tables, . . .)
Model settings			
`"--model"`	string	`"testmodel"`	Name of the model
`"--model_variant"`	int	1	Variant of the chosen model

Table 5.5: Supported parameters of the *UQsim* class that are additionally available as command line arguments to configure the behaviour of a UQ simulation.

[5]https://docs.python.org/3/library/argparse.html

Parameter	Type	Default	Description
UQ method and uncertain parameter settings			
`"--uncertain"`	string	`"all"`	Uncertain setting: can be evaluated to choose different probability distributions and their parameter values
`"--uq_method"`	string	`"sc"`	Define the UQ method: `"sc"` or `"mc"`
`"--regression"`	bool	False	Use of regression (point collocation) method
`"--mc_numevaluations"`	int	27	Number of Monte Carlo evaluations
`"--sc_q_order"`	int	2	Number of collocation points in each direction (q)
`"--sc_p_order"`	int	1	Highest order (P) of orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE
`"--sc_sparse_quadrature"`	bool	False	Enable sparse quadrature
`"--sc_quadrature_rule"`	string	`"G"`	Define the quadrature rule (see [80] for details)
`"--config_file"`	string		Config file where the parameter settings are taken from
Solver settings			
`"--parallel"`	bool	False	Enable parallelisation with threading
`"--num_cores"`	int	`cpu_count()`	Number of parallel cores per node
`"--mpi"`	bool		Enable MPI
`"--mpi_method"`	string	`"MpiPoolSolver"`	Choose MPI solver: `"MpiSolver"` or `"MpiPoolSolver"`
`"--mpi_combined_parallel"`	string	False	Enable hybrid parallelisation: MPI and threading
Chunk parameters			
`"--chunksize"`	int	1	Number of runs that are chunked into a group
`"--mpi_chunksize"`	int	1	Number of runs that are sent as a package via MPI
Runtime analysis and optimisation parameters			
`"--analyse_runtime"`	bool	False	Enable the runtime analysis
`"--opt_runtime"`	bool	False	Enable the runtime optimisation
`"--opt_runtime_gpce_Dir"`	string	`"."`	Define the folder for the runtime data
`"--opt_algorithm"`	string	`"LPT"`	Define the opt algorithm: `"FCFS"`, `"LPT"`, `"SPT"`, or `"MULTIFIT"`
`"--opt_strategy"`	string	`"DYNAMIC"`	Define the opt strategy: `"FIXED_ALTERNATE"` or `"FIXED_LINEAR DYNAMIC"`

Table 5.5 Supported parameters of the *UQsim* class that are additionally available as command line arguments to configure the behaviour of a UQ simulation.

Some of the parameters are used to control the usage of different UQ methods (Section 5.6), to support parameter studies (Section 5.7) and to enable automatic runtime measurements, predictions, and optimisations (Section 5.8) features, as well as to support the scalability from development systems to computing systems.

Listing 5.6 shows how to access parameters through the `uqsim.args` object of *UQsim*, for the example of setting the orders for q and P (cf. Listing 5.1 for further explanations).

```python
uqsim.args.sc_q_order = 6 # order (q) number of colloc. points for each param.
uqsim.args.sc_p_order = 2 # order (P) highest order of orth. poly. of the gPCE
```

Listing 5.6: Specify arguments within a Python UQ simulation script, before calling `uqsim.setup()`.

Listing 5.7 shows how to set the parameters from the command line. The Python interpreter (`python3`) is called with the script name (`uq_simulation.py`) and the arguments (parameters) for the script. Here, the same values for q and P as in Listing 5.6 have been specified.

Remark: If both variants are used, and the same parameters are used inside the script and as command line arguments, then the parameters inside the script overwrite the values from the command line.

```
python3 uq_simulation.py --sc_q_order 6 --sc_p_order 2
```

Listing 5.7: Start a UQ simulation (`uq_simulation.py`) and specify some script parameters from command line.

For each uncertain parameter, the probability distribution with its specific parameters has to be configured, which is indicated in Listing 5.8 (see the description of Listing 5.1 for further details).

```python
# initialise uncertain parameters:
if uqsim.is_master():
    uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
    uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(0, 1))
    uqsim.simulationNodes.setDist("uncertain_param_2", cp.Uniform(-1, 1))
```

Listing 5.8: Specify uncertain parameters within a Python UQ simulation script, before calling `uqsim.setup()`.

It is also possible to specify the uncertain parameters through a config file (`config.json`). The config file uses the JSON [82] format and works as shown in Listing 5.9.

```json
{
  "parameters" : [
    {
      "name"        : "uncertain_param_1",
      "distribution": "Normal",
      "mu"          : 0,
      "sigma"       : 1
    },
    {
      "name"        : "uncertain_param_2",
      "distribution": "Uniform",
      "lower"       : -1,
      "upper"       : 1
    }
  ]
}
```

Listing 5.9: Example of how to specify uncertain parameters via a config file (`config.json`) for UQEF.

By setting the `config_file` parameter (from command line or within the simulation script), the loading of the parameters from the specified filename is activated. In line 1 of (Listing 5.10), the loading is activated by setting the filename to `config.json`. The actual loading is done, when `uqsim.setup()` is called.

```python
uqsim.args.config_file = "config.json"
```

Listing 5.10: Example of how to activate a config file (`config.json`) for the uncertain parameters with UQEF.

Remark: All Chaospy distributions are supported: The names and parameters are the same as defined in Chaospy, and determined automatically via Python reflection mechanisms, which requires that the name of the distributions and the function arguments matches exactly the ones from Chaospy. The advantage of this mechanism is that automatically all Chaospy distributions are available (also new ones, without changing UQEF).

If for some parameters that are defined in a *Model* class, no probability distribution can be specified, then a fixed scalar value can be set instead. In Listing 5.11 on line 5, 1 is set as a scalar value for `"uncertain_param_2"`. The *Nodes* class inserts, thus, this fixed value in every parameter set when generating the nodes.

Remark: This allows to define many *Model* parameters without always to define these as uncertain parameters. This is very useful in cases where only the influence of one parameter after another should be tested, or in cases where the most important uncertain parameters are identified through a sensitivity analysis and then with this subset, the UQ simulation is repeated.

```python
1  # initialise uncertain parameters:
2  if uqsim.is_master():
3      uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
4      uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(0, 1))
5      uqsim.simulationNodes.setValue("uncertain_param_2", 1)
```

Listing 5.11: Example of how to specify fixed parameter values within a Python UQ simulation script, before calling `uqsim.setup()`.

It is also possible to define fixed parameters through a config file, as can be seen in Listing 5.12. For that, `"None"` is set as the `distribution` name, and the value is set on the `default` property.

```json
{
  "parameters" : [
    {
      "name"        : "uncertain_param_1",
      "distribution": "Normal",
      "mu"          : 0,
      "sigma"       : 1
    },
    {
      "name"        : "uncertain_param_2",
      "distribution": "None",
      "default"     : 1
    }
  ]
}
```

Listing 5.12: Example of how to specify fixed parameter values through a config file (`config.json`) for UQEF.

5.5 Support for different UQ methods

As described in Section 2.4.2, there are differences between Monte Carlo (Section 2.2.1), point collocation (Section 2.2.3), and the stochastic collocation (Section 2.2.2) UQ methods in the technical phases of the assimilation and certification when using Chaospy. The *Simulation* interface is an abstraction for concrete UQ simulation methods (see Figure 5.3), e.g. Monte Carlo (*McSimulation*) or the stochastic collocation with the pseudo-spectral approach (*ScSimulation*). Inside the concrete *Simulation* classes, the correct functions with their required parameters are called on the *Nodes* and *Statistics* classes. Table 5.7 lists the main functions of the *Simulation* interface.

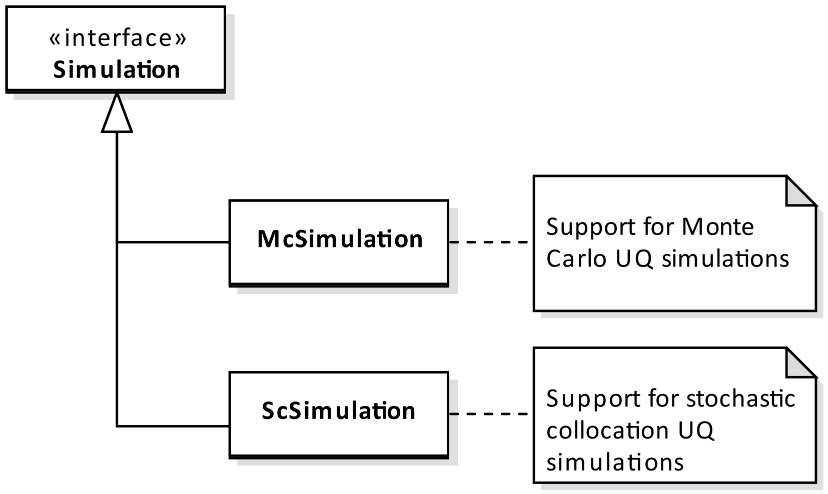

«interface»
Simulation
McSimulation
Support for Monte Carlo UQ simulations
ScSimulation
Support for stochastic collocation UQ simulations

Figure 5.3: Illustration of the supported UQEF simulation methods (UML class diagram).

Method signature	Abstract	Return value
`def getSetup(self):`	yes	A string that describes the simulation.
`def generateSimulationNodes(self, simNodes):`	yes	`None`
`def prepareSolver(self):`	no	`None`
`def calculateStatistics(self, statistics, simNodes, runtime_estimator):`	yes	`None`

Table 5.7: Functions of the UQEF *Simulation* interface class inside the *simulation* package.

In the following enumeration, the functions of the *Simulation* interface are explained:

`def getSetup(self):`

This function is called from a *UQsim* object to print the UQ simulation (method) type. `getSetup()` returns a string with the UQ simulation (method) and the specific parameter values.

`def generateSimulationNodes(self, simulationNodes):`

This function is called from a *UQsim* object during the setup (`setup()` function), to prepare the simulation. Because the generation of the simulation nodes depends on the UQ method, it is important that every *Simulation* implementation correctly utilises the `simulationNodes` object (*Nodes* class) with its specifics.

```
def prepareSolver(self):
```

This function is called from a *UQsim* object before the simulation is started. Its default implementation forwards the call to the `prepare()` method of the current *Solver* (see *Solver* interface in Section 5.6). A *Solver* can then do some initialisation before the actual propagation starts.

```
def calculateStatistics(self, statistics, simulationNodes,
                        original_runtime_estimator=None):
```

This function prepares the raw data (VoI) returned from a *Model* for the calculation of the statistics, and calls the correct function (e.g. `calculateStatisticsForMc()` or `calculateStatisticsForSc()`) on the actual `statistics` object (*Statistics* class) with the given `simulationNodes` (*Nodes* class) and an `original_runtime_estimator` (if one is loaded; see Section 5.8 for more details). The `original_runtime_estimator` is an object that represents a runtime predictor rp_N (see Equation (4.8)).

Through this interface design, UQEF is open for further UQ methods, just by deriving the *Simulation* interface. Furthermore, most of the UQEF classes are decoupled to a large extend, which improves the separation of concerns (SOC) [71], minimises the code that UQEF users have to write, and enhances the source code reuse inside UQEF. With this, one *Model* implementation (Section 5.2) is sufficient and can be used for different UQ methods without any code changes on the *Model*. The UQEF users only have to consider the concrete UQ method when subclassing the *Statistics* (Section 5.3) interface.

The UQ method for a simulation script is specified with the `uqsim.args.uq_method` parameter. For the Monte Carlo method, `"mc"` is set, as can be seen in line 2 of Listing 5.13. The number of random samples for the propagation is defined with the `uqsim.args.mc_numevaluations` parameter (line 3).

```
1  # args
2  uqsim.args.uq_method = "mc"
3  uqsim.args.mc_numevaluations = 1e3 # number of individual Monte Carlo samples
```

Listing 5.13: Example of how to activate the Monte Carlo method (see Section 2.2.1) in UQEF simulation scripts.

For stochastic collocation with pseudo-spectral approach, `"sc"` is set on `uqsim.args.uq_method` (see Listing 5.14, line 2). As already described on Listing 5.1, the orders q and P have to be set, as can be seen on lines 3–4.

```
1  # args
2  uqsim.args.uq_method = "sc"
3  uqsim.args.sc_q_order = 3 # order (q) number of colloc. points for each param.
4  uqsim.args.sc_p_order = 1 # order (P) highest order of orth. poly. of the gPCE
```

Listing 5.14: Example of how to activate the stochastic collocation with the pseudo-spectral approach (see Section 2.2.2) in UQEF simulation scripts.

The parameter `uqsim.args.sc_quadrature_rule` defines the quadrature rule for computing the coefficients c_j (Equation (2.14)). This parameter is directly given to Chaospy; therefore all quadrature rules from Chaospy[6] are supported. The parameter `sc_sparse_quadrature` enables sparse quadrature based on the chosen quadrature rule (`sc_quadrature_rule`). The sparse quadrature rule feature of Chaospy is used for that.

[6] List of supported quadrature rules in Chaospy: `https://chaospy.readthedocs.io/en/master/quadrature.html`

Remark: If the sparse quadrature feature is enabled in Chaospy, it is recommended to use this only with nestable quadrature rules (such “Leja”, “Gauss-Petterson”, or “Clenshaw-Curtis”). Otherwise, it ends up with non-optimal or more quadrature nodes as expected.

To enable the point collocation method, set `uqsim.args.uq_method = "mc"` together with `uqsim.args.regression == True`, as it is shown in Listing 5.15. Additionally, the p order (line 4) has to be defined. Implementations of the *Statistics* interface have to consider the `regression` parameter (cf. Section 5.3).

```
1 # args
2 uqsim.args.uq_method = "mc"
3 uqsim.args.mc_numevaluations = 1e6 # number of individual samples
4 uqsim.args.sc_p_order = 3          # highest order of orth. poly. of the gPCE
5 uqsim.args.regression = True       # enables regression method
```

Listing 5.15: Example of how to activate the point collocation (see Section 2.2.3) method in UQEF simulation scripts.

For the probabilistic collocation method, set `uqsim.args.regression == True` when using `uqsim.args.uq_method = "sc"` (Listing 5.16).

```
# args
uqsim.args.uq_method = "sc"
uqsim.args.sc_q_order = 3 # order (q) number of colloc. points for each param.
uqsim.args.sc_p_order = 1 # order (P) highest order of orth. poly. of the gPCE
uqsim.args.regression = True # enables regression method
```

Listing 5.16: Example of how to activate the probabilistic collocation (see Section 2.2.3) method in UQEF simulation scripts.

5.6 Scheduling and solver strategies

To support different scheduling and solver strategies, the *Solver* interface and its implementations (Figure 5.4) exist in UQEF. This is the heart of UQEF and enables the scalability features with the ability to move from a development PC to a production environment (compute cluster) without any code changes on the *Model* nor on the *Statistics*. This can be achieved by specifying the right *UQsim* parameters (Section 5.4), to setup an appropriate *Solver*. A *Solver* implements one or more scheduling strategies (see Section 4.4). The *LinearSolver* has no parallelisation and runs one model run after another. This can be used, if a single core CPU is used, or to analyse errors and debug UQ simulations. The *ParallelSolver* is a good choice to speed-up the propagation on a development PC, by starting multiple model runs in parallel. It uses the Python `joblib` [47] package to set up a thread pool, which utilises each core on a CPU with a worker thread. With the *MpiSolver*, it is possible to use static work packages (Sections 4.4.2 and 4.4.3) with the message passing interface (MPI). It can be used on development PCs and large compute systems to utilise all acquired nodes. The *MpiSolver* uses the `mpi4py` [113] Python package to access the MPI functions. The *MpiPoolSolver* sets up a worker pool on MPI level (Section 4.4.4). It uses the `mpi4py.futures` Python package with the *MPICommExecutor* to achieve the pooling functionality across computing nodes.

In Table 5.9, the main functions of the *Solver* interface are listed. The *Solver* class design follows the strategy design pattern ([48, p. 315–324]). New compute environments and scheduling strategies can be implemented by subclassing the *Solver* interface and implementing these methods.

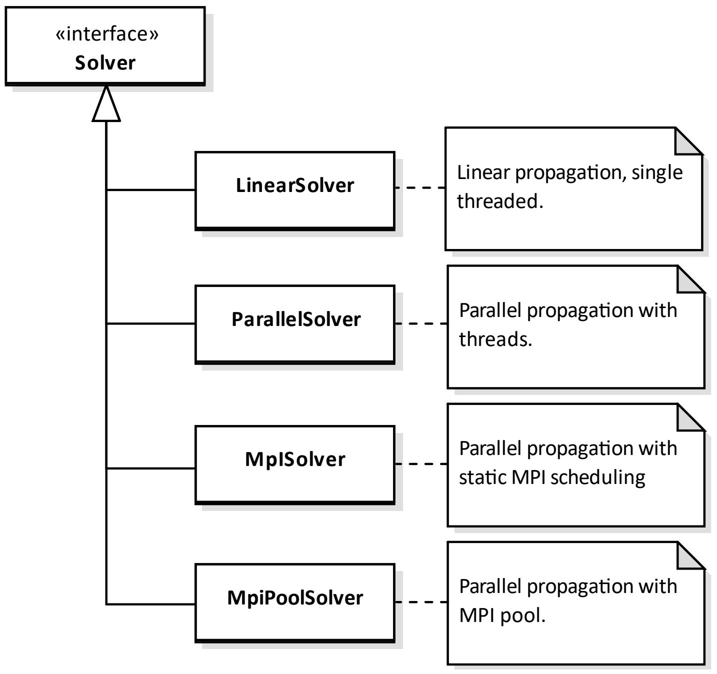

«interface»
Solver
LinearSolver
ParallelSolver
MpiSolver
MpiPoolSolver
Linear propagation, single threaded.
Parallel propagation with threads.
Parallel propagation with static MPI scheduling
Parallel propagation with MPI pool.

Figure 5.4: Illustration of the implemented UQEF solvers (UML class diagram).

Method signature	Abstract	Return value
`def getSetup(self):`	yes	A string that describes the solver.
`def init(self):`	yes	`None`
`def tearDown(self):`	yes	`None`
`def prepare(self, parameters):`	yes	`None`
`def solve(self, runtime_estimator=None, chunksize=1, algorithm=schedule.Algorithm.FCFS, strategy=schedule.Strategy.FIXED_LINEAR):`	yes	`None`

Table 5.9: Functions of the UQEF *Solver* interface class inside the *solver* package.

In the following list, the functions of the *Solver* interface are explained:

```
def getSetup(self):
```

This function is called from a *UQsim* object to print the *Solver* type. `getSetup()` returns a string with the concrete *Solver* and the specific setup details.

```
def init(self):
```

This function is called from a *UQsim* object at the very beginning of the `simulate()` function. This gives a *Solver* the chance to do some solver specific initialisations, e.g. starting the environment.

```
def tearDown(self):
```

This function is called from a *UQsim* object through its `tear_down()` function. *Solvers* can here shutdown their environment.

```
def prepare(self, parameters):
```

This function is called from the current *Simulation* object. An implementation of `prepare()` should forward this to the *Model* and save the `parameters` as a member variable.

```
def solve(self, runtime_estimator=None, chunksize=1,
          algorithm=schedule.Algorithm.FCFS,
          strategy=schedule.Strategy.FIXED_LINEAR):
```

In the `solve()` function, the whole propagation with the *Solver* specific scheduling is realised. A concrete *Solver* can use the `runtime_estimator` rp_N, if one is given, to predict the runtime for each `parameter` set. This information can then be used together with the `algorithm` (*UQsim* parameter: `opt_algorithm`) and `strategy` (*UQsim* parameter: `opt_strategy`) to schedule the individual *Model* runs. The `schedule.Algorithm` enum allows the following values: `FCFS` (first come, first served), `LPT` (longest process time first), `SPT` (shortest process time first), and `MULTIFIT`. With the `schedule.Strategy` enum, it can be defined, how the parameter sets are arranged into work packages: `FIXED_ALTERNATE` takes one parameter set after another and places it into the next work package; `FIXED_LINEAR` takes one parameter set after another and places it into the first work package, after it is full, it proceeds with the next work package; With `DYNAMIC`, the work packages evolve automatically during the propagation by giving the next parameter set to the next free worker.

With the *Solver* interface and its implementations, UQEF can support different scheduling strategies, which can be seen in Table 5.11.

Name	Threading	MPI	Scheduling	Solver
Single-threaded (linear) propagation			FCFS	*LinearSolver*
Thread pool parallelisation	X		DWP	*ParallelSolver*
Static work packages		X	SWP	*MpiSolver*
Static work packages with thread pool on node level	X	X	SWPT	*MpiSolver*
Dynamic work packages		X	DWP	*MpiPoolSolver*

Table 5.11: Overview of supported scheduling strategies (usage scenarios) with UQEF.

The following list describes the parameters to use and configure the different standard scheduling strategies:

Single-threaded (linear) propagation

For this scheduling strategy, the *LinearSolver* is used. There is no parallelisation mechanism used, that is why it is usable for very fast models, and for debugging purposes to find and fix source code errors.

Listing 5.17 shows how to use the *LinearSolver* from command line. Only the script name `uq_simulation.py` has to be given to the Python interpreter `python3`.

```
python3 uq_simulation.py
```

Listing 5.17: Start a UQ simulation with single-threaded (linear) propagation (*LinearSolver*) using UQEF.

Thread pool parallelisation

The thread pool parallelisation is a good choice for development PCs or on cluster systems when only one node is used. It uses a thread pool to utilise the CPU cores. This is equivalent to DWP scheduling.

Listing 5.18 shows how to set the parameters to use thread pool parallelisation and how to configure the number of threads. By specifying `--parallel`, the *ParallelSolver* is used. With `--num_cores`, the number of cores (threads) that should be used within the thread pool is specified. If `--num_cores` is not set, then UQEF determines automatically how many CPU cores are available and uses this value.

```
python3 uq_simulation.py --parallel    \
                         --num_cores=2
```

Listing 5.18: Start a UQ simulation with the multithreaded *ParallelSolver* on two CPU cores (threads) using UQEF.

Alternatively, also work package configurations with the parameters `--opt_strategy` and `--opt_algorithm` are available if needed (although it is not very common to use this together with the *ParallelSolver*).

Static work packages

For the static work packages, the *MpiSolver* is used. It allows the usage of a whole set of cluster computing nodes. On each CPU core, one MPI process can be started. With MPI `scatter()`, the parameter sets are distributed to the MPI processes, and with MPI `gather()`, the results are collected. This is the SWP scheduling, as described in Section 4.4.2.

With `--mpi` and `--mpi_method`, as can be seen in Listing 5.19, the SWP scheduling is configured, and the *MpiSolver* is used. By specifying `--opt_strategy` and `--opt_algorithm`, a fine-grained configuration of how the work packages are prepared can be obtained.

```
mpiexec -n 4 python3 uq_simulation.py                                 \
                     --mpi --mpi_method "MpiSolver"                    \
                     --opt_strategy "FIXED_LINEAR" --opt_algorithm "FCFS"
```

Listing 5.19: Start a UQ simulation with MPI and SWP scheduling (*MpiSolver*) using UQEF.

Static work packages with thread pool on node level

The *MpiSolver* additionally supports the static work packages with thread pool on node level (SWPT) scheduling. By setting `--mpi_combined_parallel`, a thread pool with Python `joblib` is started on each MPI process. The `--num_cores` parameter determines the number of threads in each thread pool. If `--num_cores` is not specified, then UQEF automatically determines the number of cores on the CPU and uses this value.

Listing 5.20 contains an example of how to configure SWPT. In this example, four MPI processes are created, and each MPI process uses a thread pool with four threads.

```
mpiexec -n 4 python3 uq_simulation.py                                 \
                     --mpi --mpi_method "MpiSolver"                    \
                     --mpi_combined_parallel --num_cores=4             \
                     --opt_strategy "FIXED_LINEAR" --opt_algorithm "FCFS"
```

Listing 5.20: Start a UQ simulation with MPI and SWPT scheduling (*MpiSolver*) using UQEF.

Dynamic work packages

To use dynamic work packages on a cluster, the *MpiPoolSolver* is highly recommended.

Through the *MPICommExecutor* in the `mpi4py` package, an MPI pool is started, to distribute the work across the MPI processes.

As an example, Listing 5.21 enables DWP by setting `"MpiPoolSolver"` as the `--mpi_method`. It starts an MPI pool on the master MPI process (rank 0), and three MPI worker processes.

```
mpiexec -n 4 python3 uq_simulation.py                                   \
                     --mpi --mpi_method "MpiPoolSolver"                 \
                     --opt_strategy "DYNAMIC" --opt_algorithm "FCFS"
```

Listing 5.21: Start a UQ simulation with dynamic work packages on MPI (*MpiPoolSolver*) using UQEF.

Remark: UQEF is designed to easily switch between these scheduling strategies, without changing any *Model* or *Statistics* source code. Only the parameters, when starting the UQ simulation script, have to be specified differently. This is very comfortable for the users–but this requires additional work within UQEF and the *UQsim* class. Especially if MPI is used, it has to be considered that the main script is started several times. It is important to know, which code is executed on the master MPI process (rank 0) or on a worker MPI process, not to do things twice or to miss some initialisations on the MPI workers. The `is_master()` function of the *UQsim* class is used internally to make such decisions. However, it can also be used in custom scripts to make such decisions.

5.7 Support for parameter studies

If UQ is considered as a loop around a model, then a parameter study for the UQ method is an outer-loop around UQ. UQEF, with its *UQsim* class, also allows such outer-loop scenarios. Listing 5.22 comprises such an outer-loop example. There are different orders defined for q and P (line 5). A *UQsim* object can be instantiated within that loop (line 9), and the orders q and P are taken from the outer-loop variables (lines 12–13). This works with all described scheduling and solver strategies (see Section 5.6).

```
 1  import chaospy as cp
 2  import uqef
 3
 4  #          q  p    q  p    q  p    q  p
 5  orders = [(3, 1), (4, 1), (5, 2), (6, 2)]
 6
 7  for q_order, p_order in orders:
 8      # instantiate UQsim
 9      uqsim = uqef.UQsim()
10
11      # args
12      uqsim.args.sc_q_order = q_order # number of colloc. points for each param.
13      uqsim.args.sc_p_order = p_order # highest order of orth. poly. of the gPCE
14
15      # initialise uncertain parameters:
16      if uqsim.is_master():
17          uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
18          uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(0, 1))
19          uqsim.simulationNodes.setDist("uncertain_param_2", cp.Uniform(-1, 1))
20
21      # ... setup, simulate, statistics, tear down
```

Listing 5.22: Outer-loop support example for UQ method parameter studies using UQEF.

Another outer-loop example, as shown in Listing 5.23, is the variation of the probability distribution parameters. Often, the influence of different values is under investigation. To this end, these values are stored in a list (line 5) and used for the outer-loop. The parameter values are used to initialise the distributions within the loop (lines 18–19).

```python
1  import chaospy as cp
2  import uqef
3
4  #                  p1_1    p2_1        p1_2    p2_2
5  dist_params = [((0, 1), (-1, 1)), ((5, 2), (-3, 3))]
6
7  for dp1, dp2 in dist_params:
8      # instantiate UQsim
9      uqsim = uqef.UQsim()
10
11     # args
12     uqsim.args.sc_q_order = 3 # number of colloc. points for each param.
13     uqsim.args.sc_p_order = 2 # highest order of orth. poly. of the gPCE
14
15     # initialise uncertain parameters:
16     if uqsim.is_master():
17         uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
18         uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(*dp1))
19         uqsim.simulationNodes.setDist("uncertain_param_2", cp.Uniform(*dp2))
20
21     # ... setup, simulate, statistics, tear down
```

Listing 5.23: Outer-loop support example for probability distribution parameter studies using UQEF.

5.8 Automatic runtime measurements, predictions, and optimised scheduling

The automatic runtime measurements, predictions, and the optimised scheduling strategies of Section 4.6 are already fully implemented within the UQEF software framework. A UQEF user only has to enable some parameters (comp. Section 5.4) to activate them. These features could be generally implemented and work with any *Model* implementation, as long as the *Models* are implemented correctly. It is a requirement for all *Models* that the results must, besides the VoI, additionally contain the runtime measurements for each single model run (see Section 5.2). By default, the runtime measurements are saved in an object of type *SolverTimes*, which is part of the `solver` package, but the data are not further analysed by default.

When setting the *UQsim* parameter `analyse_runtime = True`, then UQEF automatically uses the *RuntimeStatistics* class that is already built-in in UQEF, to create a runtime predictor rp_N (see Equation (4.8)) when `calc_statistic()` is called. This can be seen in the example of Listing 5.24 on line 8. The created runtime predictor rp_N can also be saved into a file, when `save_statistics()` (line 30) is called on a `UQsim` object. The file is saved into a directory that is specified with the `outputResultDir` parameter. The filename can be specified with the `uqsim_file` parameter (not shown in the example listing, the default filename is `"uqsim.saved"`). When the example runs the first time, the runtime predictor rp_N is created and saved into a file, from where it can be loaded again.

```python
1  import chaospy as cp
2  import uqef
3
4  # instantiate UQsim
5  uqsim = uqef.UQsim()
6
7  # args
8  uqsim.args.analyse_runtime = True     # enables creation of rp_N
9  uqsim.args.opt_runtime = True         # loads and uses an existing rp_N
10 uqsim.args.opt_runtime_gpce_Dir = "." # directory to load an existing rp_N
11
12 # initialise uncertain parameters:
13 if uqsim.is_master():
14     uqsim.setup_nodes(["uncertain_param_1", "uncertain_param_2"])
15     uqsim.simulationNodes.setDist("uncertain_param_1", cp.Normal(0, 1))
16     uqsim.simulationNodes.setDist("uncertain_param_2", cp.Uniform(-1, 1))
17
18 # setup
19 uqsim.setup()
20
21 # start the simulation
22 uqsim.simulate()
23
24 # statistics:
25 uqsim.calc_statistics()
26 uqsim.print_statistics()
27 uqsim.plot_statistics()
28 uqsim.save_statistics()
29
30 # tear down
31 uqsim.tear_down()
```

Listing 5.24: Example of how to create, save, and use the runtime prediction rp_N to optimise the scheduling using UQEF.

The creation of the rp_N works with any of the standard scheduling strategies (Section 5.6). But the ability to create the rp_N depends on the UQ method: All UQ methods that are based on gPCE, including the regression methods, can be used. Table 5.12 shows which *Simulation* class is suitable for creating an rp_N.

	McSimulation	***ScSimulation***
Without regression		X
With regression	X	X

Table 5.12: Overview of supported UQ methods to create a runtime predictor rp_N with UQEF.

If an existing rp_N should be used for optimising the scheduling behaviour, the *UQsim* parameter `opt_runtime = True` has to be used. With the `opt_runtime_gpce_Dir` parameter, the directory of the saved rp_N file can be specified. Listing 5.24 contains an example of setting these parameters (lines 9–10). When the example script is started the second time, it can load the rp_N from the file and use it to predict the runtime of each individual model run. Based on the chosen *Solver* and scheduling strategy (see Section 5.6), the nodes (parameter sets) for the individual model runs are sorted and grouped into work packages. All the sorting and grouping into work packages is done inside the specific *Solver* implementations because this is very individual for each scheduling strategy. Additionally, all the resorting of the results after the propagation is also done by the *Solvers*. This has the advantage of being very comfortable for the users that write custom *Statistics* implementations because they can rely on the correct order of the nodes (parameter sets) and the corresponding results (`rawResults`). The optimised scheduling can be used with any of the provided *Solvers* in UQEF and is usable with all UQ methods, including Monte Carlo.

6 Case study: Efficient uncertainty quantification in pedestrian dynamics

In the field of pedestrian dynamics (see [159, 199] for an introduction) it is investigated how pedestrians behave in different situations. Simulating the behaviour of pedestrians is very challenging because the motion is two- or even three-dimensional, it can contain counterflows and interactions in the flow. Additionally, many different environments (e.g. a car, a train, a building, a city, a hill, . . .) exist where pedestrians reside, and at entrances and exits, jamming or clogging phenomena can occur. By simulating pedestrian dynamics, a lot of pedestrian properties such as culture, age, or gender have to be considered. Usually, pedestrians are not alone, and therefore, families and even small to large groups with and without relations among the pedestrians must be taken into account. This is useful for various applications, especially in understanding how and why certain phenomena evolve and to improve safety and security aspects in buildings, places, or up to very large events.

In this thesis, three different scenarios in the field of pedestrian dynamics have been investigated. For all of the three scenarios, non-intrusive forward UQ simulations have been performed, which covers the different efficiency aspects that have been developed in this thesis. Table 6.2 lists the three scenarios, and in Table 1.1, the investigated features and techniques that are used in each scenario can be seen. All simulations for the three scenarios have been performed on the Linux-Cluster CoolMUC2 of the Leibniz Supercomputing Centre [115] using the already existing pedestrian dynamics simulator Vadere.

Scenario	Description	Section
Scenario 1	Evacuation of a train station.	Section 6.3
Scenario 2	Evacuation of a building with separated families	Section 6.4
Scenario 3	Utilisation of a campus	Section 6.5

Table 6.2: Overview of pedestrian dynamics case study scenarios.

Before diving into the three scenarios, the pedestrian dynamics simulator Vadere is explained in Section 6.1, followed by an explanation of the challenges of quantifying the uncertainty in pedestrian dynamics in Section 6.2.

6.1 Vadere: a pedestrian dynamics simulator

In this thesis, the microscopic open-source framework Vadere [17, 91] is used as a pedestrian dynamics simulator. Vadere is actively developed at the Department of Computer Science and Mathematics at the Munich University of Applied Sciences in the research group of Prof. Dr Gerta Köster. Through the microscopic approach, each pedestrian is considered individually. Vadere is written in Java and runs therefore on Windows, Mac OS X, and on Linux. Besides the graphical user interface of Vadere, a command line interface is also available, which is used in this thesis to perform the black-box model runs.

The movement (locomotion) of a pedestrian can be modelled in different ways. Vadere supports already some locomotion models such as the optimal steps model [167, 168, 166, 171], the gradient navigation model [26], and the social force model [64, 99]. Additional behavioural models can be combined with the basic locomotion models. This enables not only to simulate the dynamics of pedestrians but also cars and granular flow can be introduced into the framework.

Vadere works with the concept of scenarios, which is a description file in the JSON format containing all required information to perform a single pedestrian dynamics simulation. An overview of the main input and output files is illustrated in Figure 6.1. A scenario contains the information about the topography, e.g. the shape of a building, the pedestrians and their individual behaviour (e.g. speed and position) and the combination of a basic locomotion model with some additional behavioural models. It is further configured, how many time steps with the time step length should be performed and which measures (values of interest) should be taken and how they should be saved into result files. The output files (also JSON or optionally in CSV format) can be specified to contain the position for every pedestrian at every time step, which allows a detailed analysis of the trajectories. Another way is to introduce measure zones that automatically calculate the number of pedestrians within this measure zone and to save this information for every time step into an output file. This allows determining densities of pedestrians over some regions.

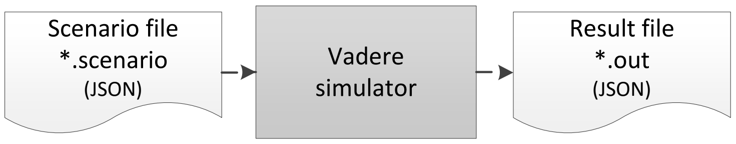


Figure 6.1: Illustration of the main input and output files of the Vadere simulator.

Listing 6.1 shows an example of a Vadere console invocation. `vadere-console.jar` contains the actual Vadere console simulator and the `evacuation.scenario` contains the scenario description. Depending on the configuration in `evacuation.scenario`, the output files are written into the `output/` folder of the current working directory.

```
java -Xms256m -Xmx1600m -Xss512m \
     -jar vadere-console.jar scenario-run -f evacuation.scenario
```

Listing 6.1: Example of calling the Vadere console simulator with the `evacuation.scenario` file and specific memory `-Xm*` settings for the java runtime environment.

When performing multiple parallel runs of the Vadere console on a computing node, it is important to specify the available memory boundaries for each java runtime environment (JRE), because otherwise out-of-memory errors can occur. On the Linux-Cluster CoolMUC2, the memory settings in Table 6.3 are used to perform parallel Vadere black-box runs. This has proven to be a good choice to allow full parallelisation on all CPU cores without having out-of-memory errors.

Parameter	Description	Memory
`-Xms`	Set initial Java heap size	256 MiB
`-Xmx`	Set maximum Java heap size	1,600 MiB
`-Xss`	Set java thread stack size	512 MiB

Table 6.3: Memory settings for Vadere console runs on the Linux-Cluster CoolMUC2 of the Leibniz Supercomputing Centre [115].

6.2 Challenges of quantifying uncertainty using Vadere

Because it is already very difficult to perform pedestrian dynamics simulations, it is even more challenging to quantify the uncertainty of such simulations. The different aspects are in detail explained in separate paragraphs below: stochastic dimensionality, unknown parameter distributions, verification and validation problems, long simulation runtimes for large or detailed scenarios, and real-time decisions. The discussed points consider Vadere as the simulation framework, but most of the points are not limited to Vadere and do also hold for other pedestrian dynamics simulation frameworks.

Stochastic dimensionality

All the challenges concerning the properties of pedestrians such as culture, age, or gender do increase the number of parameters for a scenario. Common parameters are the number of pedestrians, their goals and behaviour, the movement speed, and the timing settings (e.g. time step length, total simulation time). Depending on the kind of simulation that has to be performed, a lot of possible input parameters can be defined as unknowns, and therefore, increase the stochastic dimensionality. Additionally, the basic locomotion models like the OSM do also have various parameters[1] which could also be defined to be uncertain. It is also possible to consider the topography as uncertain parameters when quantifying the uncertainty.

Here, it is important to define what has to be investigated exactly. Defining as many parameters equally for all pedestrians and scenarios should be in focus. Otherwise, the simulation suffers heavily from the curse of dimensionality: When every pedestrian should have specific uncertain settings, this would end up in $M = \#num\_parameters \times \#num\_pedestrians$ random variables in $\boldsymbol{\zeta}$. A good choice is to start with as much default parameters as possible and keep this fixed. Only the essential ones should be defined to be uncertain. A possible list with such default parameters can be found in [204, 154], which contains mean speeds, mean densities on different places, or capacities of pedestrian systems for different kind of pedestrians under different situations.

Unknown parameter distributions

Once it is identified which parameters in a scenario should be modelled as uncertain variables ζ, the next challenge arises: Which probability distribution with its specific parameter values can be applied in which context? Again, [204, 154], can help for a starting point, but usually many interviews with specialists (e.g. psychologists) in the field have to be conducted, as it is described in [174, 172].

Qualitative and quantitative validation problems

To assess the quality of a pedestrian dynamics simulation, it has to be validated. In [174], there is a discussion on the validation problematic, and it is distinguished between qualitative and quantitative validation.

Qualitative validation compares the behaviour of the simulation results with the described behaviour of certain scientific areas of different researchers. The problem is that people do not always behave in the same way and act differently in stress situations or among different kind of groups. In quantitative validation, the produced simulation data are compared with measurable data from reality. At a first glance, this sounds easy. But performing experiments with pedestrians is not easy, because pedestrians may behave differently in experimental scenarios compared to reality, and ethical standards play a considerable role and often prohibit such experiments

[1]An overview of the OSM parameters with suitable default values can be found in [175, 171, 166].

from gathering data for comparison. For example: How often should a building be evacuated to get the required data?

The problem is still prominent in UQ simulations for pedestrian dynamics. However, UQ can help to validate a pedestrian dynamics model by producing many data with the statistical QoI measures that give more information about the general behaviour of the simulated pedestrians under uncertainty of the not exactly known parameter values.

Long simulation runtimes for large or detailed scenarios

Using Vadere for pedestrian dynamics simulations, the runtime of a single run depends on many things, especially the chosen scenario with its topography, the number of pedestrians, and the simulated timespan have a strong influence. The runtime of a single run can range from a few seconds up to 10 minutes or even more for larger scenarios. For UQ simulations where many simulations have to be performed, large computing systems such as HPC systems are required to get the results in a reasonable time.

Additionally, the runtime can depend on the values of the uncertain parameters, which means that every single run within the propagation phase of a UQ simulation, can have a significantly different runtime. This could already be observed in [175, 172] together with Vadere. In [5, 38], similar variations in the runtime for other UQ simulations could be observed. This is in detail, investigated in scenario 2 (Section 6.4).

Real-time decisions

In cases where it is required to get the UQ simulation results within seconds to make decisions in real-time, it is not suitable to use a microscopic simulator such as Vadere, because of the long simulation runtimes discussed above. In [28], it has been proposed to use the closed observables surrogate model (COSM) (Section 4.7) for a transport hub scenario to quantify the uncertainty in real-time. With this technique, it is possible to obtain the QoIs in seconds—but with limited accuracy. Section 6.5 also uses the COSM, but for a larger scenario, where the utilisation of a campus is considered, and the runtime advantage is even more visible.

6.3 Scenario 1: Evacuation of a train station

The motivation for this scenario is the emergency of the London bombings on 7th Juli 2005. In the London underground, three bombs detonated simultaneously at 8:50 AM in different tube lines. Because of the explosions, the London underground lost electricity, and it was dark on the trains. The passengers could not know if there will be additional detonations, while the emergency services could not reach them for some time. More than 700 passengers were injured, and 52 passengers were killed during the explosions. The passengers started to evacuate themselves, and the passengers without or with small injuries started to help the badly injured passengers to evacuate safely.

Exactly this helping behaviour is the subject for this scenario[2], which is described in the following Section 6.3.1 and further investigated with UQ methods. The corresponding UQ simulation setup is described in Section 6.3.2, and the numerical results can be found in Section 6.3.3. Section 6.3.4 summarises the scenario.

[2]This scenario is the result of a collaborative work with Dr Isabella from Sivers which has been originally published in [175]. For the writing of this thesis, all the simulations have been repeated with the developed UQEF containing more numerical results.

6.3.1 Helping behaviour and social identity in pedestrian dynamics

The helping behaviour of passengers in emergencies is investigated through studies with many interviews by psychologists in [30, 32, 31]. While the passengers in public transportation usually act independently, it is known that in emergencies they act somehow as a group [111] and help each other according to the social identity [189] and self-categorisation theory [198].

This social identity and helping behaviour is implemented by the social identity model application (SIMA) that is developed in [174] and additionally, in detail described in [175]. SIMA is a behavioural model implemented in Vadere which can be used with any locomotion model.

In this scenario, the helping behaviour with the social identity is investigated under uncertainty. For that, Vadere is used with the existing SIMA implementation. For a first prototype, one car out of a whole train with a simplified safe zone is modelled. Figure 6.2 illustrates the initial scenario configuration. A total of 60 pedestrians (blue and light blue circles) are inside the car. The grey star indicates the position of the bomb and the 14 pedestrians (light blue circles) that are near the grey star are likely to be injured. Vadere starts the simulation exactly after the bomb explosion, and the passengers immediately start to evacuate. Some pedestrians without or with small injuries start to help the badly injured pedestrians and move together to the safe zone (yellow striped rectangle), while the remaining not injured pedestrians move alone to the safe zone.

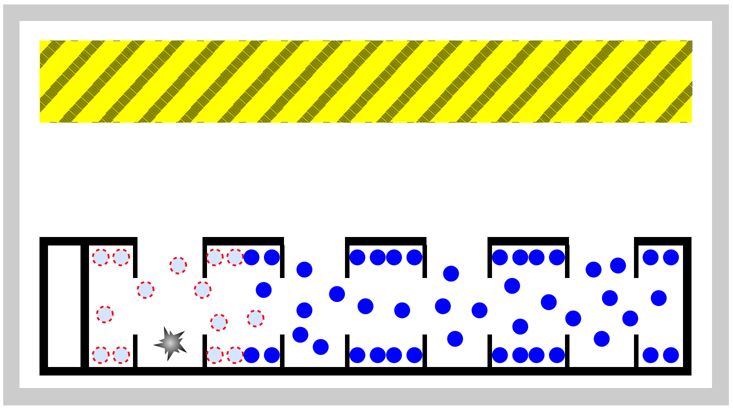


Figure 6.2: Illustration of one car for scenario 1: evacuation of a train station. Inside the car, there are 60 pedestrians (blue and light blue circles). The bomb is visualised with the grey star and the 14 pedestrians (light blue circles) near the grey star are likely to be injured. The safe zone is indicated with the yellow striped rectangle (source: [175]).

6.3.2 UQ simulation setup

According to [30], not all pedestrians share a social identity which is modelled with the $perc_{sharingSI}$ parameter, and a $U(0.6, 1.0)$ probability distribution is applied. How many pedestrians are going to be injured within an emergency is hard to predict. For an evacuation scenario, usually, only the badly injured pedestrians have to be considered differently. In this work, therefore, the percentage of badly injured pedestrians is modelled with the $perc_{injPeds}$ parameter using a $U(0.1, 0.3)$ probability distribution. The moving speed of that badly injured pedestrians is set to zero, which means they cannot evacuate themself. When a helping pedestrian assists them, the speed is modelled with a $U(0.4, 0.8)$ probability distribution in the v_{inj} parameter. The values for the probability distributions in Table 6.5 consider the reported details in [30, 4, 78, 197, 146] and the generated nodes n_i are directly given to the SIMA through a Vadere scenario file in the UQ propagation phase. The base locomotion model is the OSM, and exactly the same parameter values that are documented in [175] are used.

Parameter	Description	Distribution
$perc_{sharingSI}$	Pedestrians sharing a social identity [%]	$U(0.6, 1.0)$
$perc_{injPeds}$	Percentage of injured pedestrians [%]	$U(0.1, 0.3)$
v_{inj}	Speed of a helper with an injured pedestrian $[m/s]$	$U(0.4, 0.8)$

Table 6.5: List of uncertain parameters and their distributions for scenario 1: evacuation of a train station. The parameters are uniformly distributed between the specified minimum and maximum values (cf. [175]).

The maximum evacuation time ev_t until all pedestrians have reached the safe zone is defined to be the output of interest. For a fine-grained analysis of the evacuation scenario and the SIMA, additionally, the number of pedestrians np inside the car and the safe zone are of interest.

To quantify the uncertainty, the stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is used in this scenario. A highest order of $P = 7$ is used for the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE (Equation (2.7)), and $q = 21$ collocation points per dimension are used, which results in $Q = 21^3 = 9126$ collocation points z_i. The values for P and q are determined via parameter studies until the statistical moments have been stabilising. For a comparison, Monte Carlo simulations with $\mathcal{M} = 100{,}000$ number of samples have been performed, which enables a coarse validation of the stochastic collocation with the pseudo-spectral approach.

Figure 6.3 illustrates as a high level view the UQ simulation pipeline for this scenario. *Generate Scenario* reads a `Master.scenario` file and generates a Vadere scenario file for each collocation point z_i with the specific parameter values for each of the three parameters (cf. Table 6.5). The other (non-uncertain parameter) values are used as they are specified in the `Master.scenario`. *Propagate* performs the actual black-box runs and calls the Vadere console once (similar to Listing 6.1), for each generated Vadere scenario file. *Parse results* reads the output files of each Vadere console run and extract the values of interest. These VoIs are then processed by *Calculate Statistics*, which calculates the statistics like the mean μ_{u_N} (Equation (2.17)) and visualises them by generating appropriate plots and tables for the interpretation of the results afterwards.

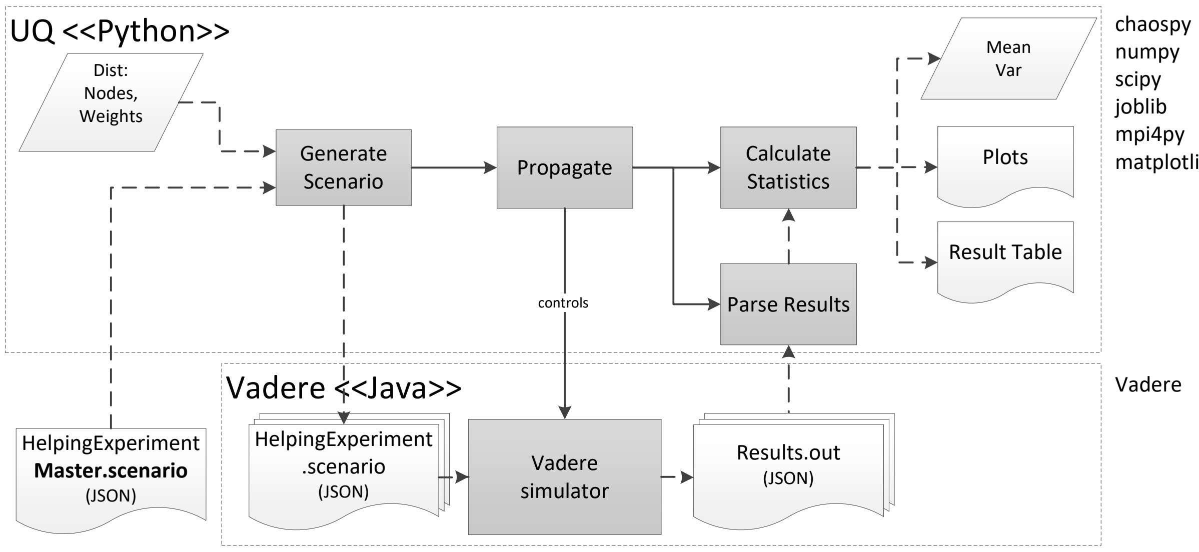


Figure 6.3: Illustration of the UQ simulation pipeline for scenario 1: evacuation of a train station. The pipeline shows the interaction of the UQ parts with the Vadere simulator. The used software is listed on the right side.

To perform this UQ simulation, UQEF is used. The derived classes for this scenario can be seen in Figure 6.4. The Vadere console simulator is applied in the *TrainEvacuationModel* class, which is derived from the UQEF *Model* interface (Section 5.2). All the statistical evaluations for the QoI is implemented in the *TrainEvacuationStatistics* class, which is derived from the UQEF *Statistics* interface (Section 5.3).

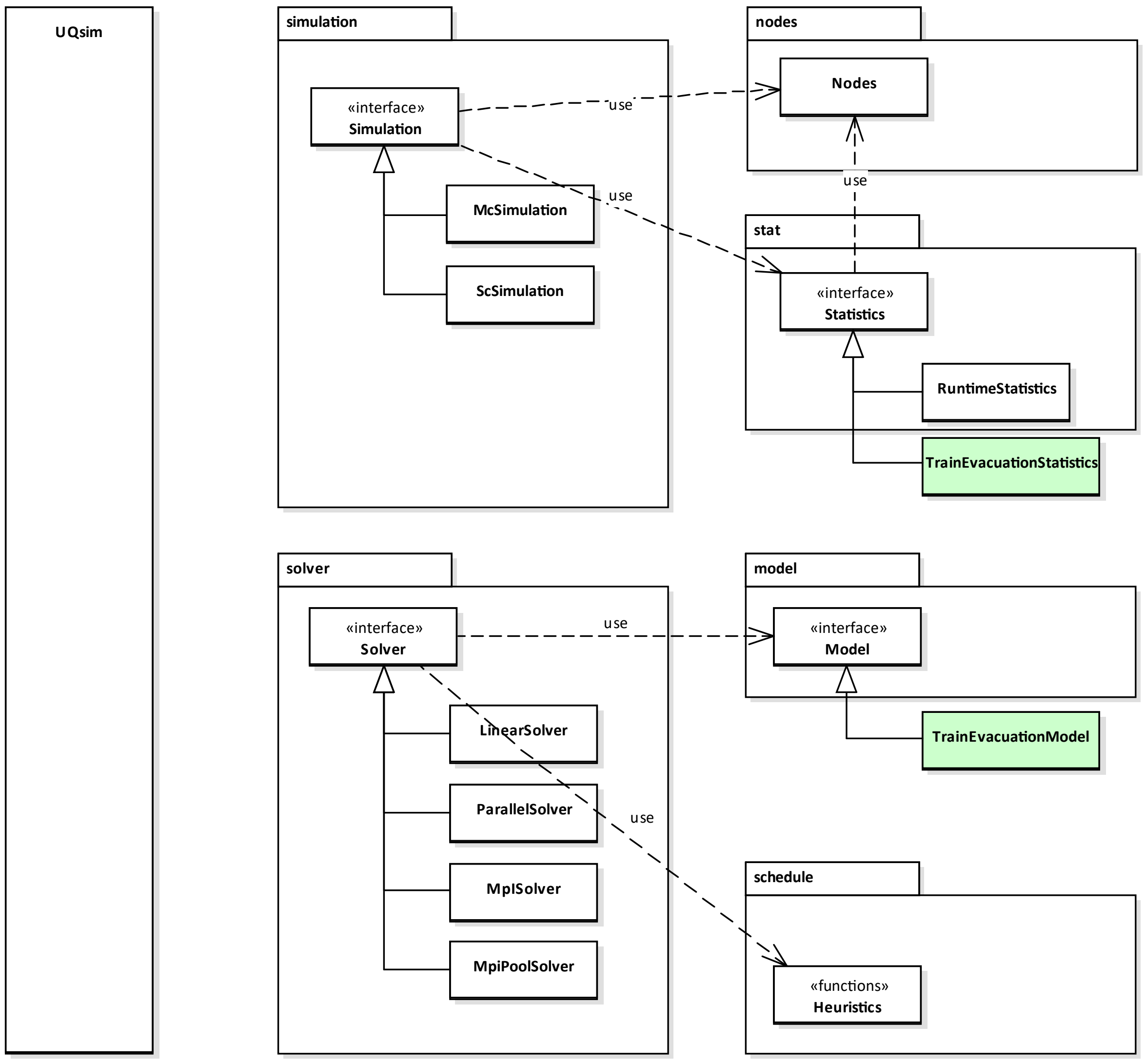


Figure 6.4: Illustration of derived classes for scenario 1: evacuation of a train station. The light green rectangles indicate the implemented custom classes for this scenario. See Section 5.1 for a detailed description of the overall UQEF architecture.

The implemented simulation program architecture based on UQEF is visible in Figure 6.5. A SLURM batch script submits a batch job with the *train evacuation* Python script to the SLURM scheduler. UQEF is configured to use DWP scheduling with four cluster nodes. The *train evacuation* Python script instantiates the *UQsim* class and registers the *TrainEvacuationModel* as a custom model, and *TrainEvacuationStatistics* as a custom statistics object. *TrainOutputProcesses* acts as a helper class for the *TrainEvacuationModel* to extract the VoIs from the Vadere output files. Additionally, all the values for the probability distributions for the uncertain parameters according to Table 6.5 are set in the *train evacuation* Python script. With that, the whole software is set up and ready to perform the UQ simulation for this scenario. The corresponding results are documented in the next section.

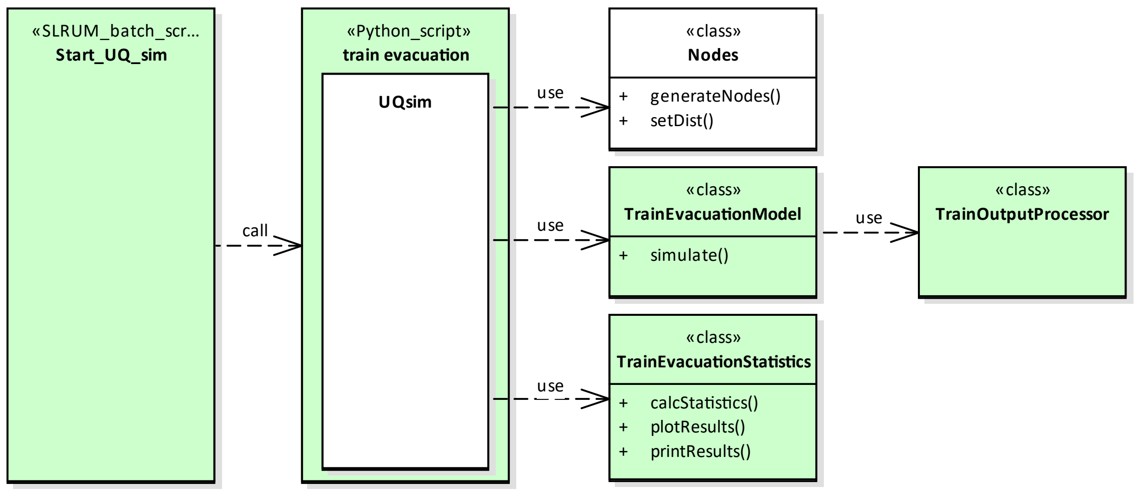


Figure 6.5: Illustration of the simulation program architecture for scenario 1: evacuation of a train station. The light green rectangles indicate the custom implementation parts for this scenario. The white rectangles indicates the UQEF classes that are instantiated and used directly, but are not changed.

6.3.3 Numerical results

To analyse the influence of each uncertain parameter without any other influence, three UQ simulations have been performed where only one parameter was defined to be uncertain, and the others are kept fixed with predefined values (the average between their minimum and maximum value which are defined in Table 6.5). The fourth simulation, with all of the three uncertain parameters, has been additionally performed to analyse the overall uncertainty and possible interaction between the parameters.

Figure 6.6(a) shows the results when only $perc_{sharingSI}$ is defined to be uncertain.

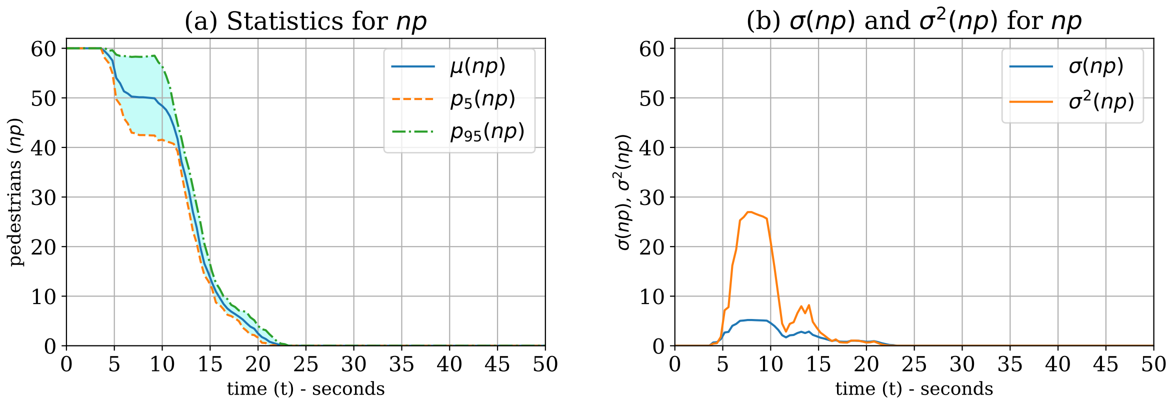


Figure 6.6: QoI results for the number of pedestrians np remaining in the car of the train with *pedestrians sharing a social identity* ($perc_{sharingSI}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

As can be seen on the percentiles ($p_5(np)$ and $p_{95}(np)$) and on the variance $\sigma(np)$ as well as the standard deviation $\sigma^2(np)$ in Figure 6.6(b), the uncertainty has a strong influence at the beginning of the simulation and decreases in the end. This seems to be plausible because as fewer pedestrians share a social identity, the more pedestrians can evacuate themselves quickly. But as long as enough pedestrians are willing to help the badly injured pedestrians, all pedestrians can reach the safe zone. In the end, the values for the maximum evacuation time ev_t varying only a little bit with a $\sigma(ev_t)$ (Equation (2.17)) of 0.9 seconds around the $\mu(ev_t)$ (Equation (2.19)) of 26.9 seconds.

Figure 6.7 contains the results for *percentage of injured pedestrians* ($perc_{injPeds}$) as the uncertain parameter. Within the first seconds, the parameter has not much influence on the number of pedestrian np remaining in the car. This is again because the unharmed pedestrians who do not share a social identity near the doors evacuate themself quickly without considering the badly injured pedestrians. But then—which is highly expected—it has a very strong influence. This is also confirmed by ev_t with a $\mu(ev_t)$ of 20.7 seconds and a $\sigma(ev_t)$ of 4.3 seconds.

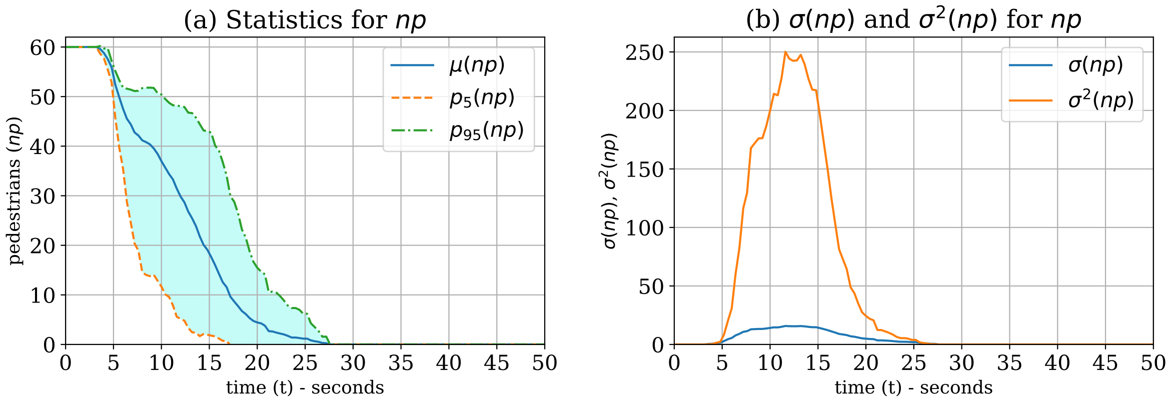


Figure 6.7: QoI results for the number of pedestrians np remaining in the car of the train with *percentage of injured pedestrians* ($perc_{injPeds}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

An additional visualisation of the uncertain influence of the *percentage of injured pedestrians* ($perc_{injPeds}$) parameter is visualised in Figure 6.8. Figure 6.8(a) contains a plot of the probability density function $PDF(ev_t)$ and the corresponding statistical moments $\mu(ev_t)$, $\sigma(ev_t)$, $p_5(ev_t)$, and $p_{95}(ev_t)$. For that, the QoI distribution (Section 3.5) is reconstructed by the usage of the gPCE (Equation (2.7)). It can be seen that there is a shift in the probability on the right side of the mean $\mu(ev_t)$. In Figure 6.8(b), a 3D plot of the reconstructed distribution for the number of pedestrians np remaining in the car shows the whole influence and the shape for each time step in the simulation.

The results for *speed of a helper with an injured pedestrian* (v_{inj}) as the uncertain parameter are visualised in Figure 6.9. As the statistics show, at the beginning of the simulation, there is almost no influence on the number of pedestrians np that have already been evacuated. This is because a helper needs some time to reach a person and then evacuate with the reduced speed v_{inj} until the first pair can reach the safe zone. The other non harmed pedestrians near the doors which do not share a social identity can immediately reach the safe zone. Starting from about 8 seconds, the v_{inj} is going to influence np, because then the aided evacuation speed is more significant. The maximum evacuation time ev_t ranges with a $\sigma(ev_t)$ of 2.9 seconds around the $\mu(ev_t)$ of 21.8 seconds.

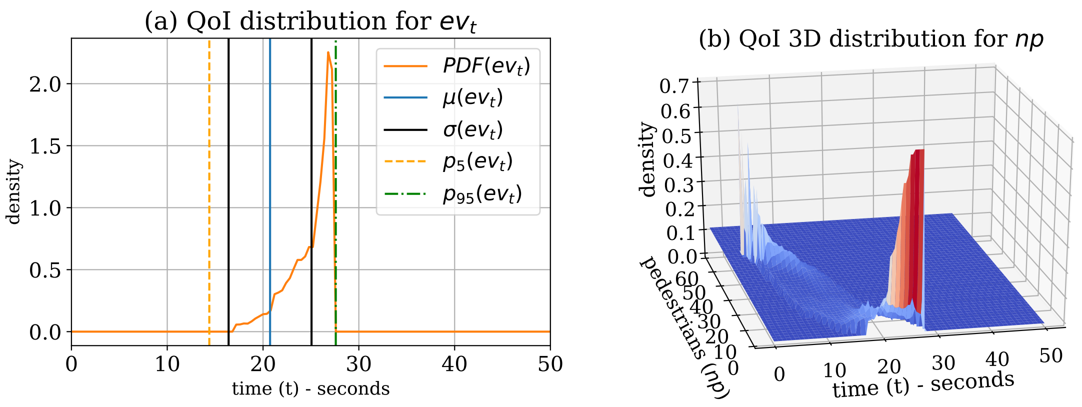


Figure 6.8: QoI distribution reconstruction for *percentage of injured pedestrians* ($perc_{injPeds}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) contains the plots of the reconstructed QoI distribution of the maximum evacuation time ev_t. (b) visualises the reconstructed QoI 3D distribution of the number of pedestrians np in the car for every time step in the simulation.

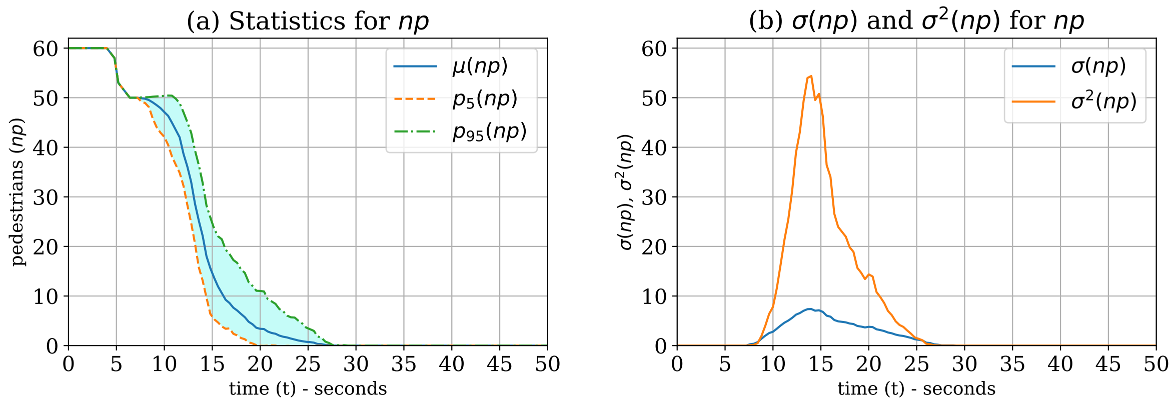


Figure 6.9: QoI results for the number of pedestrians np remaining in the car of the train with *speed of a helper with an injured pedestrian* (v_{inj}) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

To understand the influence on the evacuation scenario with all of the three uncertain parameters (Table 6.5) defined to be uncertain, a fourth simulation has been performed, whose results are visualised in Figure 6.10. Now, the accumulated influence of all parameters is strongly visible from the very beginning of the simulation. The $\mu(ev_t)$ of 21.1 seconds ranges about $\sigma(ev_t)$ 5.6 seconds for the maximum evacuation time ev_t.

Table 6.7 contains a summary of the maximum evacuation times ev_t for all performed UQ simulations. It can be observed that the standard deviation $\sigma(ev_t)$ is only a little bit higher for simulation number four than for the simulations where only one parameter is defined to be uncertain. Therefore, the uncertainty does not seem to be fully cumulative, which needs to be further investigated.

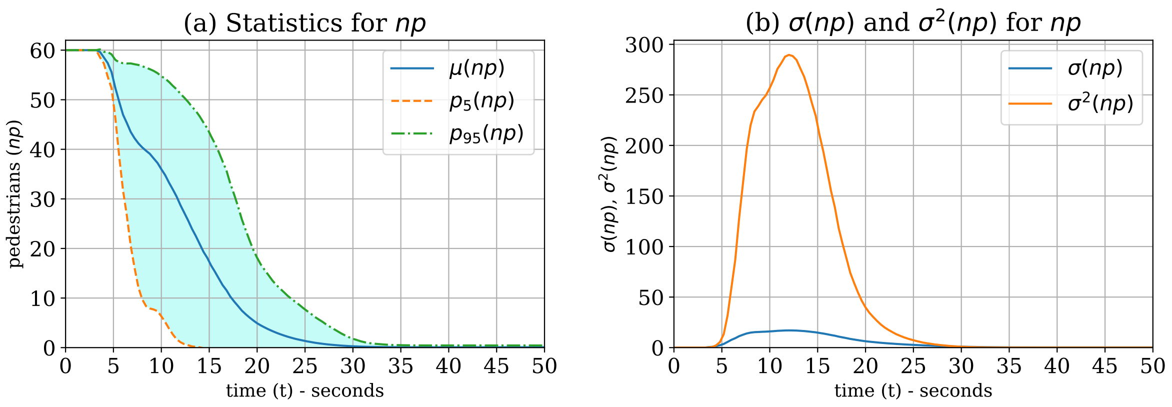


Figure 6.10: QoI results for the number of pedestrians np remaining in the car of the train with all of the three uncertain parameters (Table 6.5) for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

Nr.	Uncertain parameter(s)	QoI: Maximum evacuation time			
		$\mu(ev_t)$	$\sigma(ev_t)$	$p_5(ev_t)$	$p_{95}(ev_t)$
1	Pedestrians sharing a social identity ($perc_{sharingSI}$)	20.9	0.5	20.5	21.9
2	Percentage of injured pedestrians ($perc_{injPeds}$)	20.7	4.3	14.3	27.5
3	Speed of a helper with an injured pedestrian (v_{inj})	21.8	2.9	18.0	26.7
4	All three parameters	21.1	5.6	12.1	30.8

Table 6.7: QoI results of the maximum evacuation time ev_t for scenario 1: evacuation of a train station. The values can be interpreted as seconds and have been rounded down to one digit after the decimal point (cf. [175]).

Another way of determining the influence of a single parameter is the generation of sensitivity indices, as described in Section 2.3. The first-order sensitivity indices can be seen in Figure 6.11(a): $si\_perc_{sharingSI}$ (blue curve) shows no influence at the start but has a very strong influence starting from about five seconds of the simulation time. As $si\_perc_{injPeds}$ (green curve) shows, the $perc_{injPeds}$ parameter only has a strong influence in the beginning and then goes nearly down to zero. For v_{inj} it is confirmed by si_v_{inj} (orange curve) that there is a significant influence in the middle of the simulation time. si_int indicates a strong interaction of the uncertain parameters during the end of the simulation. The total-order sensitivity indices can be seen in Figure 6.11(b), which shows that the total influence of the uncertain parameters is in the following order: $perc_{injPeds}$, v_{inj}, and lastly $perc_{sharingSI}$. Important to see on $st\_perc_{sharingSI}$ is that both, the first-order and the total-order sensitivity indices are containing essential information: If the focus is only on the first-order sensitivity indices, then one might carelessly drop the uncertain $perc_{sharingSI}$ parameter to reduce the dimensionality of the problem—but then valuable information might get lost.

With the constructed gPCE (Equation (2.7)), it is further possible to investigate the behaviour of the maximum evacuation time ev_t by using the gPCE surrogate for it and plot the function values for varying numbers and different combinations of the uncertain parameters. The result for that can be seen in Figure 6.12. For all combinations of the three uncertain parameters (Table 6.5), it is now visible how the underlying function (scenario 1: evacuation of a train station) of ev_t behaves for the support of the input parameters.

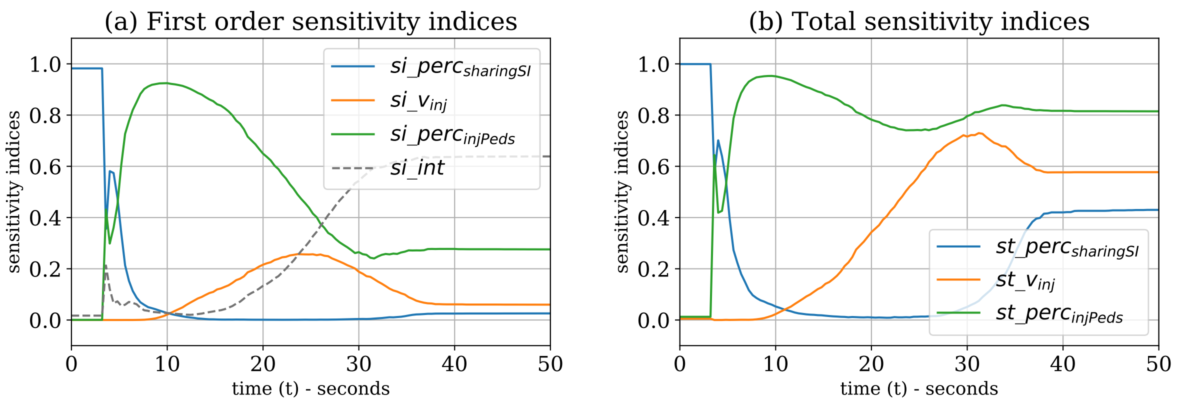


Figure 6.11: Sensitivity indices for the number of pedestrians np remaining in the car of the train with all of the three uncertain parameters (Table 6.5) for scenario 1: evacuation of a train station. (a) shows the first-order sensitivity indices and (b) shows the corresponding total-order sensitivity indices.

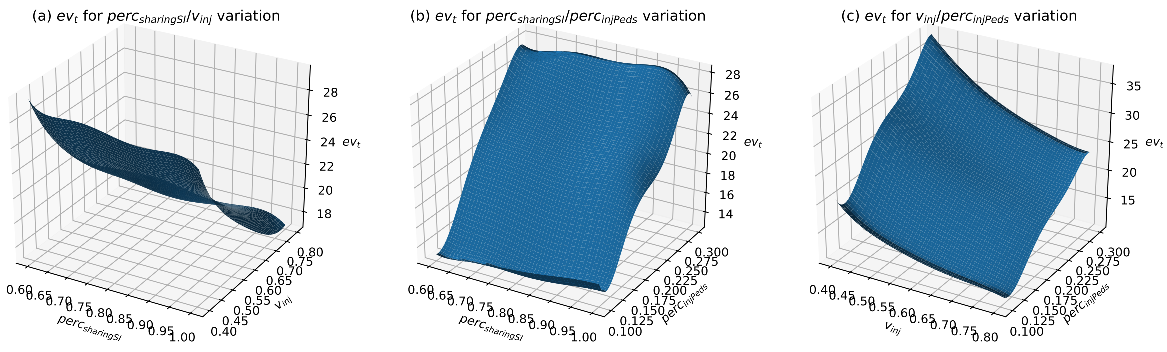


Figure 6.12: Visualisation of the generated gPCE (Equation (2.7)) for the maximum evacuation time ev_t with all of the three uncertain parameters (Table 6.5) for scenario 1: evacuation of a train station. (a) contains the resulting ev_t values for the $perc_{sharingSI}/v_{inj}$ parameter variation, (b) for $perc_{sharingSI}/perc_{injPeds}$, and (c) for $v_{inj}/perc_{injPeds}$.

For a comparison to the figures above, Appendix A.5 contains plots for a Monte Carlo UQ simulation with 100,000 samples for the same scenario. The results look qualitatively similar to the one of the stochastic collocation with the pseudo-spectral approach in this section, which nicely shows that the choice of a suitable UQ method helps to efficiently quantify the uncertainty by reducing the number of required black-box model runs significantly.

As can be seen in this analysis of the influence of the uncertainty, it is of great value to not only focus on the standard statistical moments in UQ. Using the VoI for every time step and trying to find valuable plots to visualise the statistical moments can help the pedestrian dynamics application engineers a lot in understanding the scenarios and their algorithms. Especially the information about the influence of a single parameter, its combination with the others, and the function reconstruction with the gPCE helps them to interpret the results efficiently.

6.3.4 Summary

In this scenario, the evacuation of a train during an emergency with helping behaviour among the pedestrians is investigated under uncertainty using the pedestrian dynamics simulator Vadere and the social identity model application (SIMA). To quantify the uncertainty, the efficient stochastic collocation with the pseudo-spectral approach (Section 2.2.2) UQ method is used.

Three parameters are identified to be uncertain (see Table 6.5). When all of the three parameters are defined to be uncertain at the same time, a strong uncertainty in the maximum evacuation time ev_t becomes visible (cf. Table 6.7). This shows the relevance of the use of UQ methods when investigating such scenarios. The importance of each parameter could be identified with two different approaches: First, three UQ simulations with only one uncertain parameter have been performed, while the others are kept fixed at predefined values. Second, a global sensitivity analysis (Section 2.3) is performed on top of the UQ simulation with all of the three uncertain parameters, which gives additional information about the importance and the interaction with the other parameters.

For the analysis of the QoI, the number of persons np remaining in the car of the train are additionally taken as an OoI and plotted for each time step of the simulation as proposed in Section 3.5. This gives additional insights about the evacuation behaviour and the expected evacuation time ev_t.

For a more detailed analysis of the QoI, the QoI distribution is reconstructed with the implemented *SampleDist* functionality (Sections 3.4 and 3.5): This is used for the maximum evacuation time ev_t, and the resulting QoI distribution is used to visualise the probability density function and its statistical moments. Furthermore, this technique is used to reconstruct the QoI distribution of the number of pedestrians np remaining in the car for each time step of the simulation. By plotting this within a 3D plot allows to understand the progress of the distribution during the simulated evacuation. With the technique of the reconstruction of the model function of the evacuation time ev_t via the constructed gPCE, the ev_t is visualised as a 3D plot for each of the three uncertain parameter value variations. The combination of the different visualisation techniques allows analysing the influence of the uncertainty on this scenario efficiently.

The simulation codes have been implemented with UQEF (Chapter 5) on a development PC with a reduced number of collocation points and with a coarse time step for the Vadere scenario for a fast and rapid development cycle. For the production runs, the Linux-Cluster CoolMUC2 with four cluster nodes is used. Due to the scaling support of UQEF (Section 5.6), only some parameters have to be changed to run the UQ simulation on the cluster efficiently. No code changes in the model and statistics implementation are required.

Because this UQ simulation setup was the first with Vadere and stochastic collocation with the pseudo-spectral approach, an additional Monte Carlo (Section 2.2.1) simulation with 100,000 number of $\mathcal{M}$ samples have been performed for comparison reasons. The results of the Monte Carlo simulation compared to the stochastic collocation with the pseudo-spectral approach with $Q = 9126$ number of collocation points z_i look qualitatively similar (cf. Section 6.3.3 and Appendix A.5), and gives more reliability when stochastic collocation with the pseudo-spectral approach is further used for UQ simulation scenarios with Vadere. Again, only a few parameters need to be changed when starting the simulation to switch to a different UQ method with UQEF (according to Section 5.5).

In this scenario, the DWP scheduling is used for the production runs on the Linux-Cluster CoolMUC2, but the scheduling and runtime behaviour is not further analysed. This is the subject of the next scenario.

6.4 Scenario 2: Evacuation of a building with separated families

The evacuation of buildings, e.g. in emergencies like fire alarms is an event that sporadically happens. The alarm is a surprising event for most of the pedestrians inside a building. Everyone knows that now the building should be left as fast as possible to a safe zone. Nevertheless, separated family members seem not to evacuate individually, they want to evacuate together as a group [16], especially if small children are among the family members.

The goal for this scenario[3] is to reproduce this group behaviour under uncertain conditions for separated families in a building when a fire alarm occurs. Additionally, the runtime and scheduling behaviour is analysed in detail for this scenario, because of the strong variation of the Vadere simulator runtime depending on the uncertain input parameters. The remainder for this scenario is as follows: Section 6.4.1 contains the scenario description, Section 6.4.2 the UQ simulation setup, Section 6.4.3 the numerical results, Section 6.4.4 the results for the runtime and scheduling behaviour, and Section 6.4.5 a summary of the scenario.

6.4.1 Family search strategy in pedestrian dynamics

In pedestrian dynamics, the group behaviour in evacuation scenarios has been the topic of active research [75, 3, 130, 196, 191] for many decades. Several models [130, 98, 119] have been developed to simulate such behaviour. For separated families with small children, the behaviour seems to be more difficult: Family members return into the danger zone [170, 122] to search for missing family members. Small children tend to freeze in such emergencies [107, 108], and the parents try to find them to evacuate together with their children. All this possibly increases the evacuation time, as indicated in [150].

Exactly this search behaviour is implemented in the family affiliation model (FAM), which is developed in [172], for the Vadere simulator. FAM is a behavioural model which can be used on top of a locomotion model. In FAM, adult pedestrians can search for their children. Figure 6.13 illustrates the scenario. It considers a building with one floor, multiple rooms and one combined entrance and exit. Inside the building, the scenario considers $np = 100$ pedestrians. Some are considered to be adults without children (blue circles), some are adults with a child (green stars), and some are the children (pink triangles). In this scenario, it is assumed that one adult pedestrian can have only one child, and every child has exactly one adult parent pedestrian.

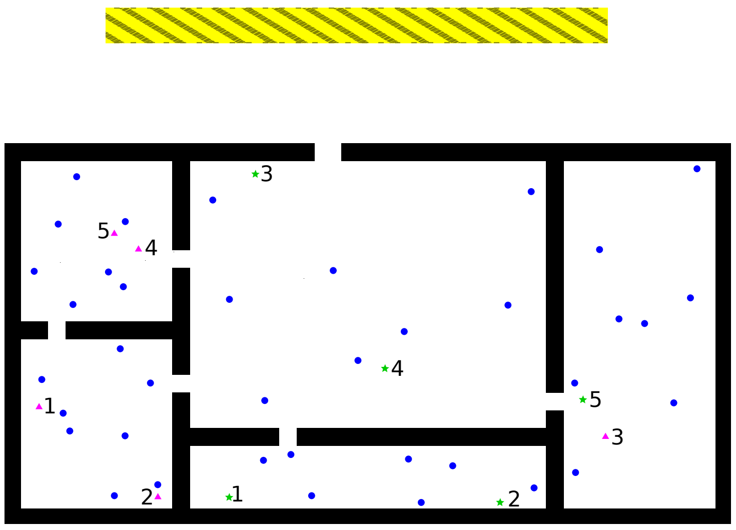


Figure 6.13: Illustration of the building for scenario 2: evacuation of a building with separated families. Inside the building, the adult pedestrians without a child are denoted by blue circles, the adult pedestrians with a child by green stars, and child pedestrians as pink triangles. The safe zone is indicated with the yellow striped rectangle. In this figure, the total number of pedestrians are reduced for visualisation purposes compared to the actual scenario setup (cf. [172, 103]).

[3]This scenario is the result of a collaborative work with Dr Isabella from Sivers which has been originally published in [172]. For the writing of this thesis, all the simulations have been repeated with the developed UQEF containing more numerical results.

The simulation directly starts at the point when the fire alarm occurs. All adults without children immediately start to evacuate themself to the safe zone (yellow striped rectangle), while the children tend to freeze. For that, their speed is set to zero. The adults with children actually do not know where their child is and start to search for them. Their search strategy is to always look in the nearest room first (see [173] for a description of how people search) until they have found it. Once they have found their child, they evacuate together with a reduced speed to the safe zone.

6.4.2 UQ simulation setup

The FAM has three relevant parameters: *percentage of family members* ($perc_{fam}$), *speed of parent-pedestrians searching their child-pedestrians* (v_{parent}), and *speed of the parent-child-pair* (v_{child}) which are defined to be uncertain. In [172], there is a discussion of the problematic of finding reasonable examples with corresponding values for uncertain parameters. Here, the same uniform distributions and parameters settings are used to perform the UQ simulation. The parameters with their distributions are listed in Table 6.8. This scenario uses OSM as the base locomotion model. The parameters for the OSM are set as defined in [172].

Parameter	Description	Distribution
$perc_{fam}$	percentage of family members [%]	$U(0.1, 0.5)$
v_{parent}	speed of parent-pedestrians searching their child-pedestrians [m/s]	$U(1.4, 1.8)$
v_{child}	speed of the parent-child-pair [m/s]	$U(0.8, 1.2)$

Table 6.8: List of uncertain parameters and their distributions for scenario 2: evacuation of a building with separated families. The parameters are uniformly distributed between the specified minimum and maximum values (cf. [172]).

As for scenario 1 (Section 6.3), the maximum evacuation time ev_t until all pedestrians have left the building and finally received the safe zone is defined to be the output of interest. The number of pedestrians np inside the building is taken as a second output of interest for a fine-grained analysis of the uncertain influence on this scenario.

To quantify the uncertainty, again, the stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is used in this scenario. A highest order of $P = 7$ is used for the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE (Equation (2.7)), and $q = 21$ collocation points per dimension are used, which results in $Q = 21^3 = 9126$ collocation points z_i. The values for P and q are determined through parameter studies until the statistical moments have been stabilising.

In Figure 6.14, the UQEF architecture with the derived classes for this scenario is visualised. *FamilyEvacuationModel* contains all the source code to call the Vadere console and *FamilyEvacuationStatistics* contains the statistical evaluation source code for the quantity of interests.

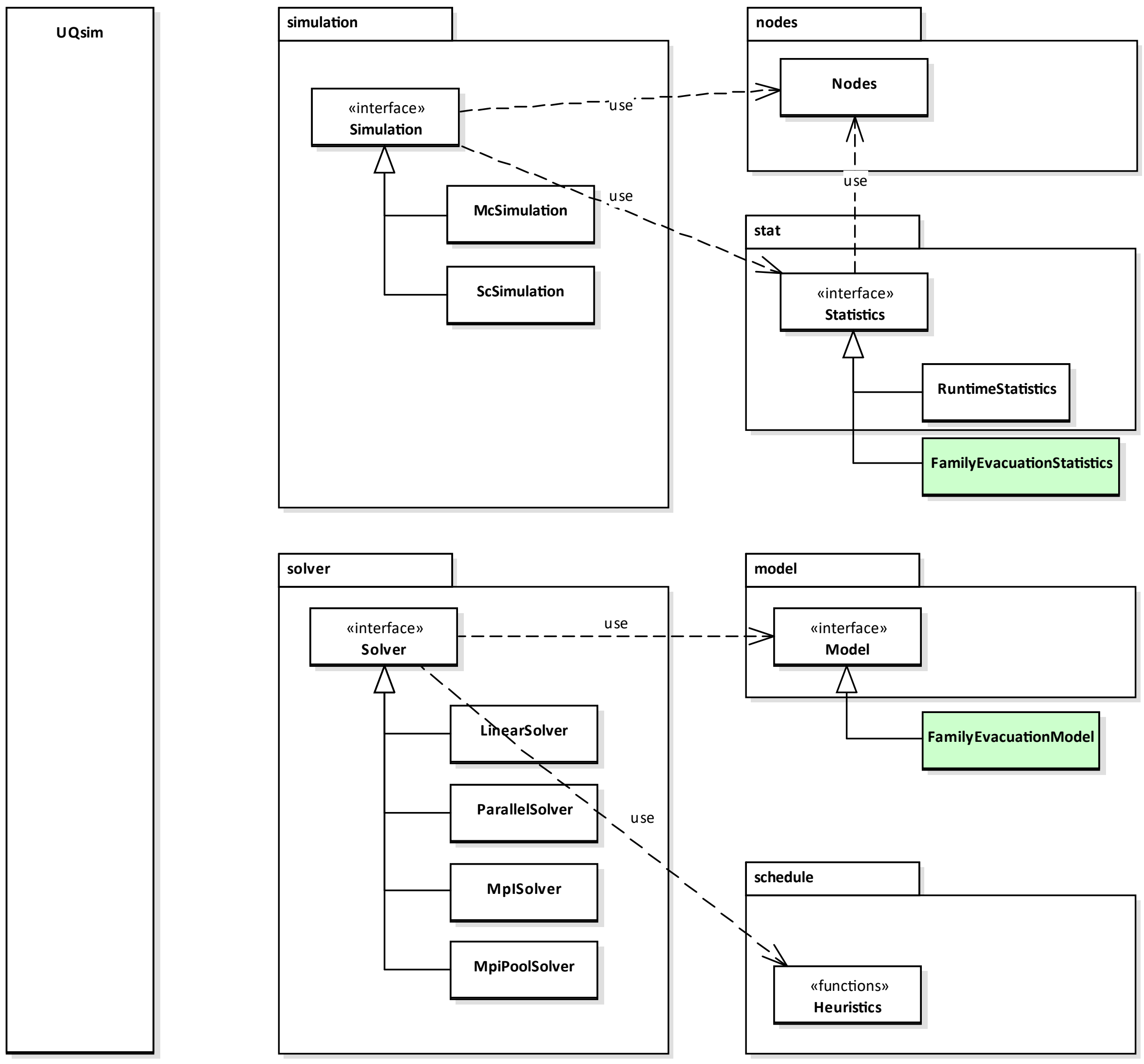


Figure 6.14: Illustration of derived classes for scenario 2: evacuation of a building with separated families. The light green rectangles indicate the implemented custom classes for this scenario. See Section 5.1 for a detailed description of the overall UQEF architecture.

Figure 6.15 shows the implemented UQEF simulation program architecture. It is similar to scenario 1 (Section 6.3)—but with different scenario-specific classes.

For the analysis of the runtime and scheduling behaviour, many additional UQ simulations have been performed. All standard scheduling strategies SWP (Section 4.4.2), SWPT (Section 4.4.3), and DWP (Section 4.4.4) as well as the developed optimised scheduling strategies SWP$_{OPT}$ (Section 4.6.1), SWPT$_{OPT}$ (Section 4.6.2), and DWP$_{OPT}$ (Section 4.6.3) using the runtime prediction mechanism (Section 4.5) are considered here. Similar to Chapter 4, the T_{Prop} is measured for a different number of collocation points $q = 4, 5, \ldots, 12$ and for a different number of cluster nodes $cn = 2, 3, 4, 5$. The statistical QoI runtime evaluation is performed in the *RuntimeStatistics* class (see Figure 6.14) of UQEF, which is generally available and does not need any changes for the usage in this scenario.

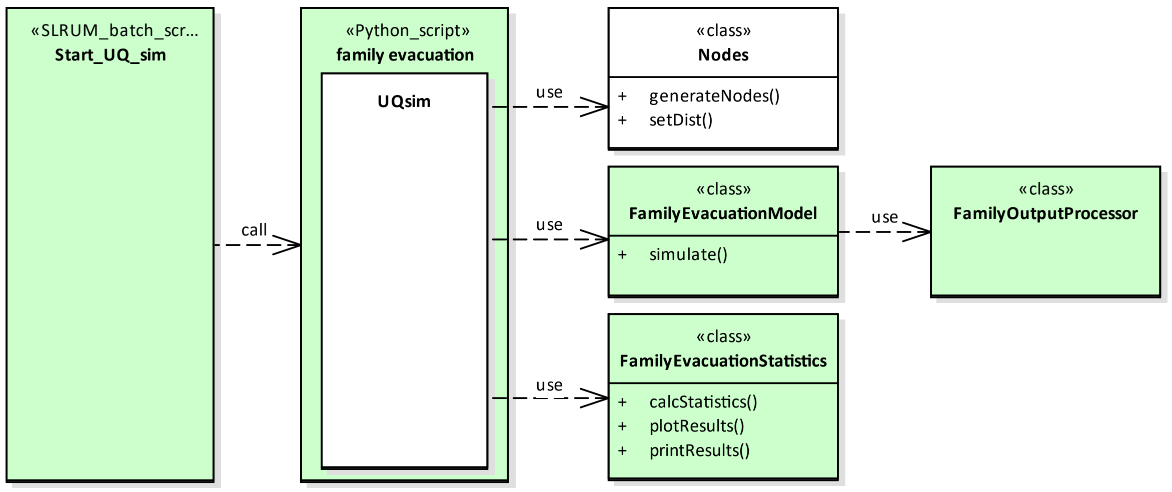


Figure 6.15: Illustration of the simulation program architecture for scenario 2: evacuation of a building with separated families. The light green rectangles indicate the custom implementation parts for this scenario. The white rectangles indicates the UQEF classes that are instantiated and used directly, but are not changed.

6.4.3 Numerical results

To analyse the influence of the uncertainty, several UQ simulations have been performed: First, three UQ simulations have been realised with only one of the parameters set to be uncertain. The others are kept fixed with predefined values (the average between their minimum and maximum value which are defined in Table 6.8). Second, a UQ simulation with all of the three parameters are defined to be uncertain is executed.

The first uncertain parameter is *percentage of family members* ($perc_{fam}$). As can be seen in Figure 6.16, the percentiles ($p_5(np)$ and $p_{95}(np)$), as well as the standard deviation $\sigma(pn)$ and the variance $\sigma^2(pn)$ indicates only a small influence of the uncertainty until the first 20–25 seconds in the number of remaining pedestrians np in the building. This is where the adult pedestrians without a children move individually to the save zone. Between 25–65 seconds, there is a significant influence of the uncertainty in the number of pedestrians np in the building. The more families are in the building, the more likely is it, that a parent pedestrian has to search in several rooms until its child is found. This is the reason for the strong influence of $perc_{fam}$ on np. The QoI for the maximum evacuation time ev_t results in a mean $\mu(ev_t)$ of 74.7 seconds and a standard deviation $\sigma(ev_t)$ of 5.8 seconds.

Remark: Note that the percentiles ($p_5(np)$ and $p_{95}(np)$) and the variance $\sigma^2(pn)$ (and the standard deviation $\sigma(pn)$) in Figure 6.16 have their maximum difference to the mean $\mu(np)$ on different points in time, since they contain only a different subset of the overall information of the resulting quantity of interest distribution. The range between the 5th and 95th percentile (other than the variance) contain about 90% of the values, whereas the variance $\sigma^2(pn)$ contains less (this depends on the resulting QoI probability distribution; for a normal distribution, the variance σ^2 indicates that about 68% of the values are inside the interval around the mean μ).

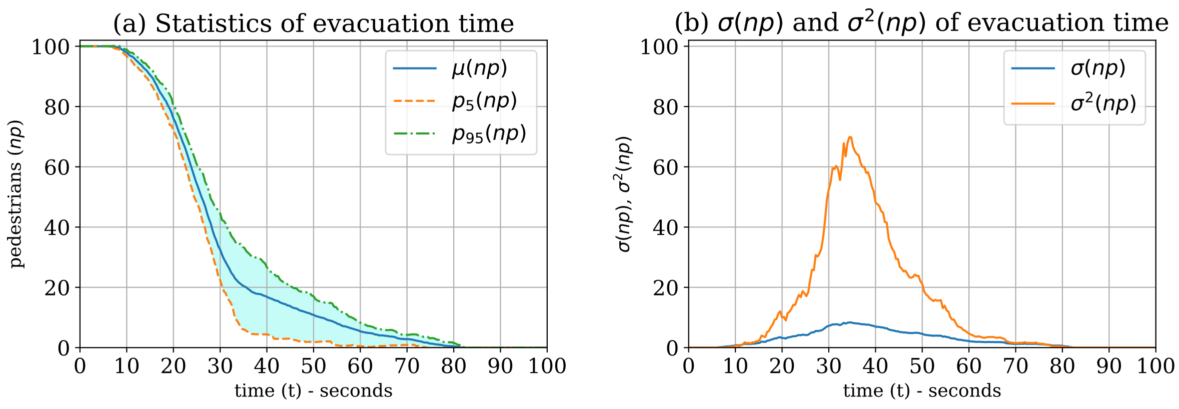


Figure 6.16: QoI results for the number of pedestrians np remaining in the building with *percentage of family members* ($perc_{fam}$) as the uncertain parameter for scenario 2: evacuation of a building with separated families. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

The results for the second parameter, the *speed of parent-pedestrians searching their child-pedestrians* (v_{parent}), are plotted in Figure 6.17. Until the first 30 seconds, almost no influence of the uncertainty is visible on the number of pedestrians np inside the building. To the end of the simulation, the influence is going to be slightly visible. This can be explained with the short time period where the parent pedestrians search their children. In this scenario, the largest part is the evacuation time when the parent moves together with its child to the safe zone. This results in a mean $\mu(ev_t)$ of 68.2 seconds with a small standard deviation $\sigma(ev_t)$ of 2.0 seconds for the maximum evacuation time ev_t.

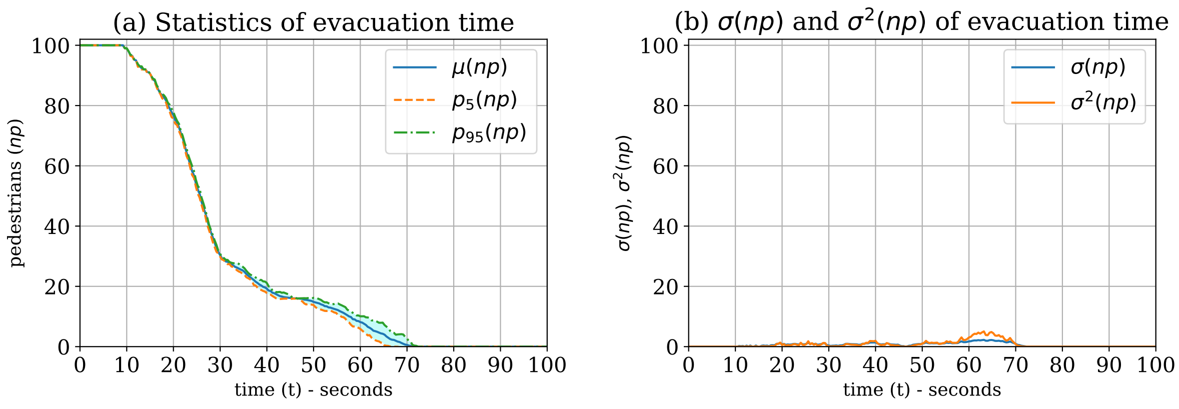


Figure 6.17: QoI results for the number of pedestrians np remaining in the building with *speed of parent-pedestrians searching their child-pedestrians* (v_{parent}) as the uncertain parameter for scenario 2: evacuation of a building with separated families. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

Finally, the results for the third uncertain parameter, the *speed of the parent-child-pair* (v_{child}) is visualised in Figure 6.18. Similar to the second uncertain parameter v_{parent}, almost no uncertainty is visible during the first 30 seconds on the number of pedestrians np remaining in the building. Then, there is an influence visible until the end of the simulation: The maximum evacuation ev_t with a mean $\mu(ev_t)$ of 68.5 seconds ranges with a standard deviation $\sigma(ev_t)$ of

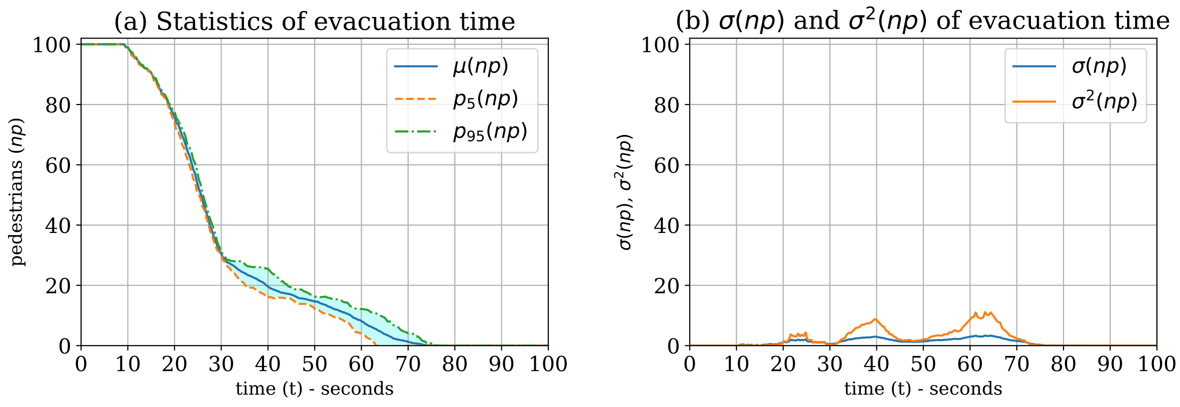


Figure 6.18: QoI results for the number of pedestrians np remaining in the building with *speed of the parent-child-pair* (v_{child}) as the uncertain parameter for scenario 2: evacuation of a building with separated families. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

4.0 seconds.

Figure 6.19 contains the results for the UQ simulation with all three parameters of Table 6.8 being uncertain. The first 10 seconds, no pedestrian is able to reach the safe zone. After that, the uncertainty is beginning to influence the number of pedestrians np remaining in the building. At about 25–30 seconds, the uncertainty is going to have a strong impact. This is when the first parent-child-pairs leave the building and reach the safe zone. The maximum evacuation time ev_t results in a $\mu(ev_t)$ of 75.2 seconds that ranges strongly with a $\sigma(ev_t)$ of 7.6 seconds.

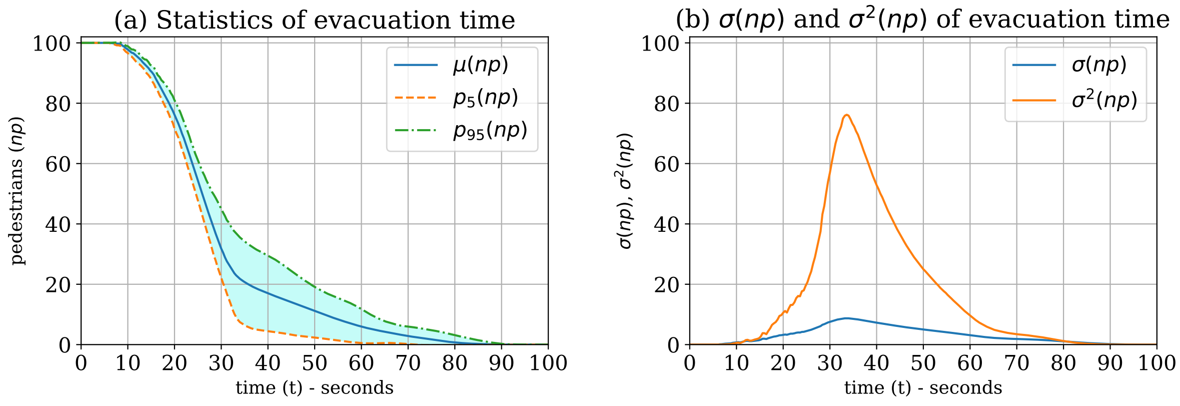


Figure 6.19: QoI results for the number of pedestrians np remaining in the building with all of the three uncertain parameters (Table 6.8) for scenario 2: evacuation of a building with separated families. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

A summary of the quantity of interest statistics of the maximum evacuation time ev_t is listed in Table 6.10. Looking at $\sigma(ev_t)$, $perc_{fam}$ has the strongest influence on ev_t, followed by v_{child}, and finally by v_{parent}. Considering the percentiles ($p_5(ev_t)$ and $p_{95}(ev_t)$), the same order in the influence as for the standard deviation $\sigma(ev_t)$ can be seen.

Nr.	Uncertain parameter(s)	QoI: Maximum evacuation time			
		$\mu(ev_t)$	$\sigma(ev_t)$	$p_5(ev_t)$	$p_{95}(ev_t)$
1	Percentage of family members ($perc_{fam}$)	74.7	5.8	62.7	80.6
2	Speed of parent-pedestrians searching their child-pedestrians (v_{parent})	68.2	2.0	65.1	71.2
3	Speed of the parent-child-pair (v_{child})	68.5	4.0	62.7	74.6
4	All three parameters	75.2	7.6	61.9	86.8

Table 6.10: QoI results of the maximum evacuation time ev_t for scenario 2: evacuation of a building with separated families. The values can be interpreted as seconds and have been rounded down to one digit after the decimal point (cf. [175]).

For the distinction of the influence on the uncertainty for the number of pedestrians np remaining in the building, a global sensitivity analysis (Section 2.3) has been performed. The first-order sensitivity indices are plotted in Figure 6.20(a). As can be seen on si_perc_{fam}, $perc_{fam}$ is the dominating parameter. At about 80 seconds, the other parameters also contribute to the varying np, where v_{child} is going to be the dominating parameter. v_{parent} has much less influence during the whole simulation time than the other two parameters. It can also be recognised on si_int that there is a significant interaction between the parameters at the very end. To evaluate the overall contribution of each parameter, the total-order sensitivity indices in Figure 6.20(b) help. Until the first 75 seconds, a similar behaviour as for the first-order sensitivity indices can be observed. Then, the picture changes: Besides v_{child}, v_{parent} also becomes very important. The reason for that is that the faster a parent pedestrian finds his child, the earlier they can start evacuating together, and v_{parent} has the most influence for the pedestrians that have to search in many rooms until they find their child.

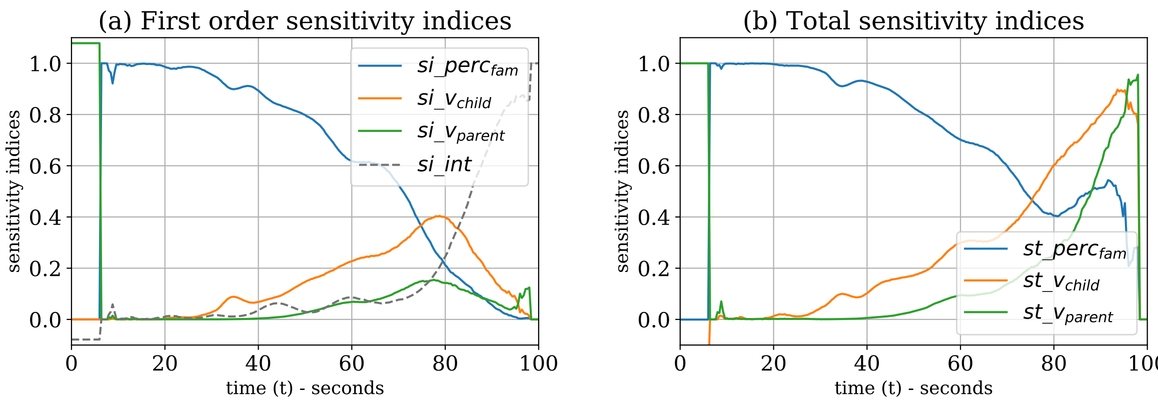


Figure 6.20: Sensitivity indices for the number of pedestrians np remaining in the building with all of the three uncertain parameters (Table 6.8) for scenario 2: evacuation of a building with separated families. (a) shows the first-order sensitivity indices and (b) shows the corresponding total-order sensitivity indices.

In Figure 6.21, the whole gPCE (Equation (2.7)) function values for the number of pedestrians np inside the building are plotted for the different combinations of the three uncertain parameters (Table 6.10). This gives an overview of the shape for the whole support of the three parameters and further helps to understand the FAM behaviour.

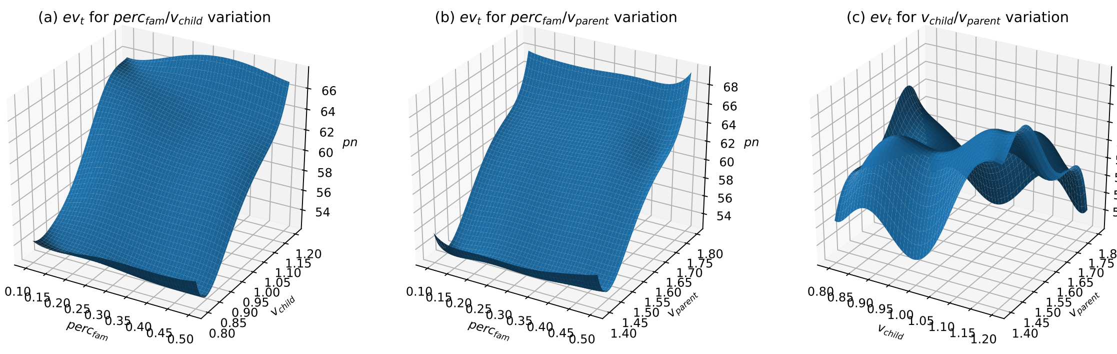


Figure 6.21: Visualisation (after 24 seconds of the simulation) of the generated gPCE (Equation (2.7)) for the number of pedestrians np with all of the three uncertain parameters (Table 6.8) for scenario 2: evacuation of a building with separated families. (a) contains the resulting np values for the $perc_{fam}/v_{child}$ parameter variation, (b) for $perc_{fam}/v_{parent}$, and (c) for v_{child}/v_{parent}.

Using UQ for such evacuation scenarios makes the uncertainty visible and shows that there is a strong need to take this into account. This first UQ simulations with the analysed QoI and the resulting plots can be used as a blueprint for further investigations and more concrete evacuation scenarios, e.g. when planning new buildings. As for scenario 1 (Section 6.3), it is of great value to use the VoI for every time step. This allows much more insights to the behaviour of the model as only looking on the maximum evacuation time ev_t.

6.4.4 Runtime and scheduling behaviour

The FAM tends to vary a lot in the runtime T_S^i depending on the values of the three input parameters. Additionally, after all pedestrians have been evacuated, the simulation is stopped. This stopping criterion seems to reinforce the variation of the runtime. Therefore, the runtime, its variation, and a possible optimisation are analysed in detail in this section.

First, the individual black-box model runs and their runtime T_S^i have been measured, which is plotted in Figure 6.22(a). As can be seen on the measurements (blue circles), the runtime

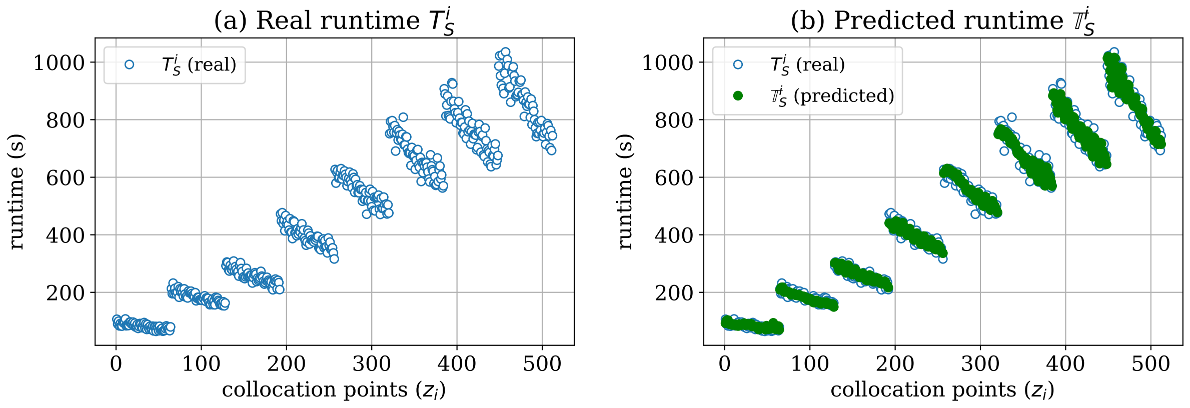


Figure 6.22: (a) Real (measured) runtime T_S^i (blue circles) for scenario 2: evacuation of a building with separated families. (b) contains the corresponding predicted runtime $\mathbb{T}_S^i$ (filled green dots) compared to the measured runtime T_S^i (blue circles). For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

T_S^i vary significantly between 64–1035 seconds and is somehow clustered in eight steps. Figure 6.22(b) shows the predicted runtimes $\mathbb{T}_S^i$ (filled green dots) using the rp_N (Equation (4.8)).

The predicted $\mathbb{T}_S^i$ does not exactly match the real measured runtimes T_S^i due to some prediction errors[4].

To further understand the prediction quality, the error between the measured T_S^i and the predicted runtimes $\mathbb{T}_S^i$ is determined. Figure 6.23(a) shows the absolute error ϵr_i (Equation (4.10)), which ranges mainly between 0–20 seconds and only a few absolute error values reaching up to about 84 seconds. The absolute errors ϵr_i are higher for the longer T_S^i runtimes. In Figure 6.23(b), the relative errors $\epsilon r_{i,\mathrm{rel}}$ (Equation (4.11)) are plotted. The values for $\epsilon r_{i,\mathrm{rel}}$ are mostly below 0.1. For the runs with the smaller runtimes T_S^i, the relative error $\epsilon r_{i,\mathrm{rel}}$ is higher and reaches values up to 0.28.

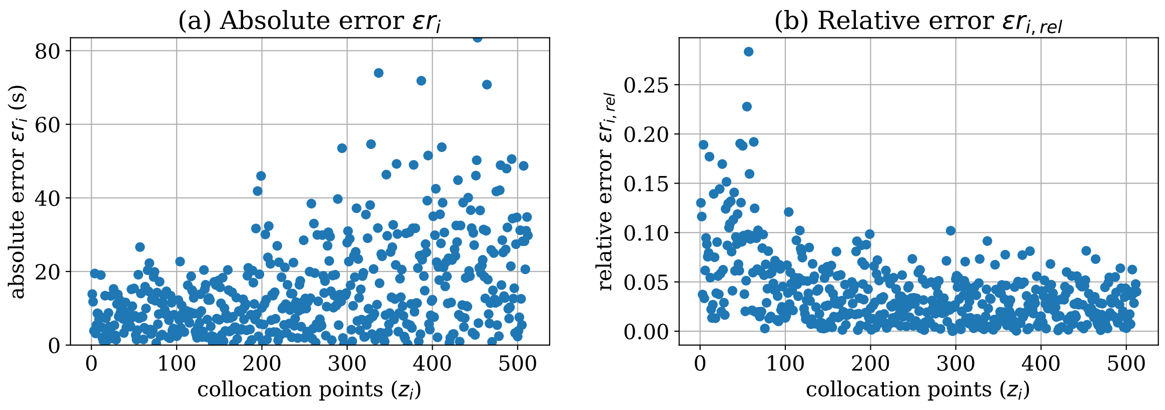


Figure 6.23: (a) Absolute error ϵr_i for scenario 2: evacuation of a building with separated families. (b) contains the corresponding relative error $\epsilon r_{i,\mathrm{rel}}$. For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

For an overview of the error trend with different $q = 4, 5, \dots, 12$, the statistics of the error is analysed in Figure 6.24. Because no analytical solution is available for this scenario, the predictions are compared against the UQ simulation with $q = 12$. Figure 6.24(a) contains the

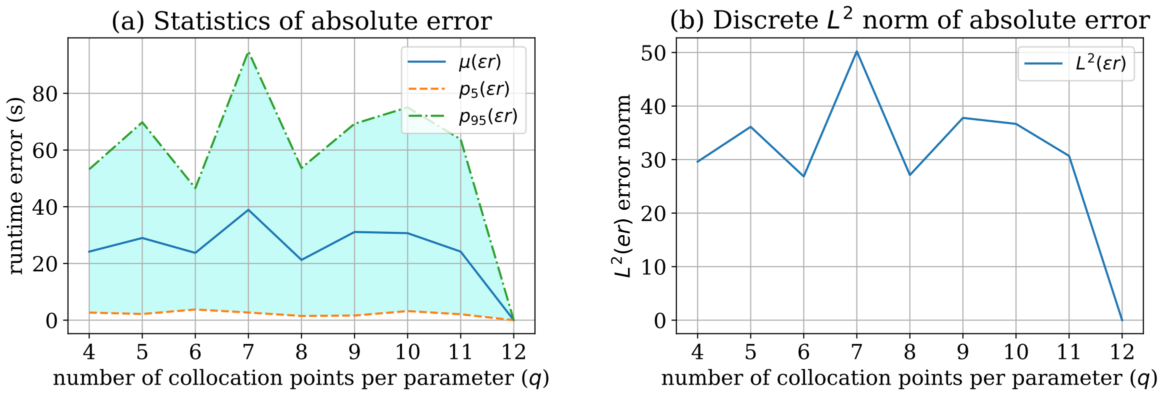


Figure 6.24: (a) Absolute error statistics with mean error $\mu(\epsilon r)$ as well as the 5th $p_5(\epsilon r)$ and the 95th $p_5(\epsilon r)$ percentile for scenario 2: evacuation of a building with separated families. (b) contains the corresponding $L^2(\epsilon r)$ error norm. For the UQ simulations, different number of collocation points $q = 4, 5, \dots, 12$ per parameter have been used.

mean error $\mu(\epsilon r)$ as well as the 5th $p_5(\epsilon r)$ and the 95th $p_5(\epsilon r)$ percentile. For all evaluated

[4]The stochastic collocation with the pseudo-spectral approach does not use interpolation and can therefore predict slightly different values at the collocation points z_i (cf. Section 2.2.2, Section 2.2.3, and [34]).

q, the mean error $\mu(\epsilon r)$ is high with values about 21.2–31.1 seconds. The percentiles ($p_{95}(\epsilon r)$ and $p_{95}(\epsilon r)$) do also indicate a high range in the values. The $L^2(\epsilon r)$ error norm is plotted in Figure 6.24(b). The values are also high and ranging about 29.6–50.2 seconds. It can be observed that with higher q, the $L^2(\epsilon r)$ is not decreasing, which is usually expected. This behaviour is assumed due to some intrinsic uncertainty inside the Vadere simulator, which randomly initialises the starting position of the pedestrians as well as the route planning. This intrinsic uncertainty is not controlled by one of the three FAM parameters (Table 6.8). It results therefore in slightly different runtimes T^i_S in subsequent black-box runs of the Vadere simulator. Additionally, other influences from the compute cluster and operating system specific properties can play a role there. These errors do only belong to the prediction of the T^i_S runtimes and not to the quality of the UQ analysis in Section 6.4.3.

The shape of the generated runtime predictor rp_N (Equation (4.8)) is visualised in Figure 6.25. This gives a good overview of the resulting runtimes $\mathbb{T}^i_S$ at the support of the input parameters for the variation of $perc_{fam}$ to v_{child} (Figure 6.25(a)), $perc_{fam}$ to v_{parent} (Figure 6.25(b)), and v_{child} to v_{parent} (Figure 6.25(c)). This indicates that $perc_{fam}$ is the dominating parameter because it produces the most changes in $\mathbb{T}^i_S$ for its support. v_{child} does also have some influence on $\mathbb{T}^i_S$, whereas v_{parent} seems to have the smallest impact.

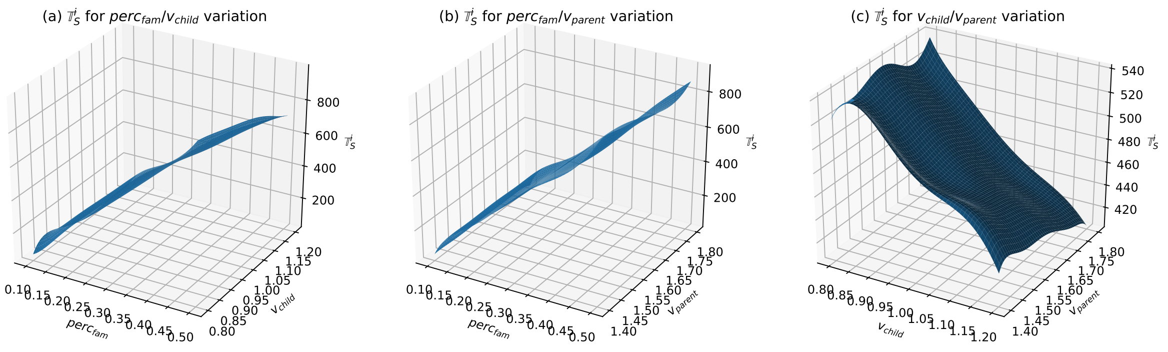


Figure 6.25: Visualisation of the generated runtime predictor rp_N (Equation (4.8)) for the predicted $\mathbb{T}^i_S$ runtime with all of the three uncertain parameters (Table 6.8) for scenario 2: evacuation of a building with separated families. (a) contains the resulting $\mathbb{T}^i_S$ runtimes for the $perc_{fam}/v_{child}$ parameter variation, (b) for $perc_{fam}/v_{parent}$, and (c) for v_{child}/v_{parent}. For the UQ simulations, $Q = 8^3 = 512$ collocation points have been used.

The whole runtime prediction $\mathbb{T}^i_S$ for the parameter support is now visible an can be further used to control the scheduling. The question is whether a good scheduling can be achieved despite the large $\mathbb{T}^i_S$ runtime prediction errors that could be seen above? To answer this, the three standard scheduling strategies (Section 4.4) and their optimised versions (Section 4.6) have been intensively tested.

The measured propagation runtimes T_{Prop} for the six scheduling strategies can be found in Figure 6.26. As can be seen in Figure 6.26(a) for $cn = 2$ cluster nodes and $q = 12$ collocation points per dimension, SWP takes with 29,060 seconds the longest propagation time T_{Prop}, SWPT with 23,686 seconds the second longest, and the others ranging between 14,619–17,002 seconds. The results for $cn = 5$ cluster nodes are visible in Figure 6.26(b). Again, SWP takes with 12,535 seconds longest, followed by SWPT with 11,121 seconds. SWP$_{OPT}$ could compete for $q \leq 11$ with the others, but for $q = 12$ the propagation runtime significantly increases up to 10,246 seconds. DWP, SWPT$_{OPT}$, and DWP$_{OPT}$ ranging about 5,814–6,391 seconds for $q = 12$. For a complete overview, all measured propagation runtimes T_{Prop} for all $cn = 2, 3, 4, 5$ in combination with all $q = 4, 5, \ldots, 12$ can be found in Table A.6.

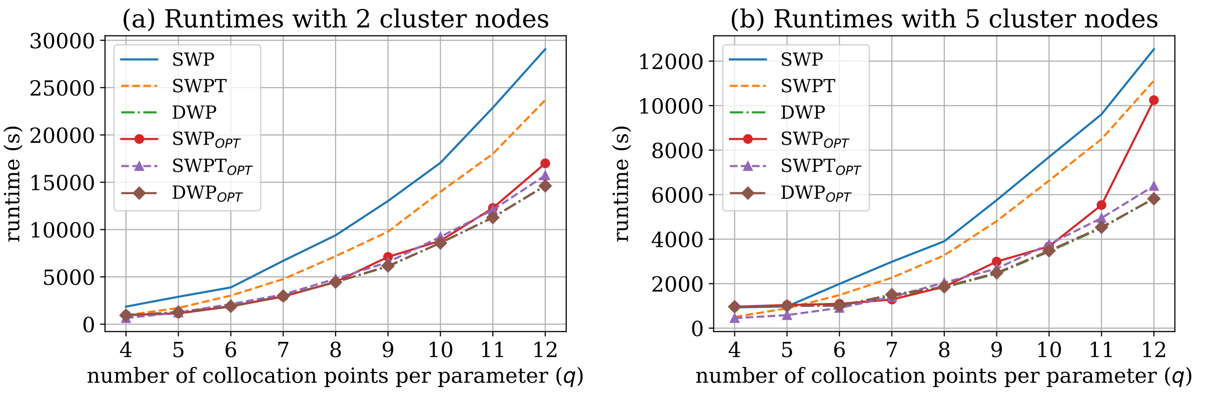


Figure 6.26: Measured propagation runtimes T_{Prop} for scenario 2: evacuation of a building with separated families for two cluster nodes ($cn = 2$) in (a) and for $cn = 5$ in (b) with a varying number $q = 4, 5, \dots, 12$ of collocation points per parameter. The UQ simulations have been performed for the three standard scheduling strategies (Section 4.4) and their optimised versions (Section 4.6).

The improvements in the propagation time T_{Prop} for SWP compared to the optimised scheduling strategies can be seen in Figure 6.27(a). The speed-up is calculated for all $cn = 2, 3, 4, 5$ and contains therefore 4 corresponding speed-up lines of identical colour and line style. For $q \geq 6$, the speed-up for SWP to SWP$_{OPT}$ is about 1.2–2.4, SWP to SWPT$_{OPT}$ about 1.7–2.3, and SWP to DWP$_{OPT}$ about 1.7–2.3. Figure 6.27(b) contains the speed-ups for the standard scheduling strategy with its optimised version. It shows speed-ups for $q \geq 6$ with SWP to SWP$_{OPT}$ of about 1.2–2.4, for SWPT to SWPT$_{OPT}$ of about 1.4–1.8, and for DWP to DWP$_{OPT}$ only about 1.0 (for $q \geq 8$). The speed-ups for all tested cn and q are additionally listed in Appendix A.4.

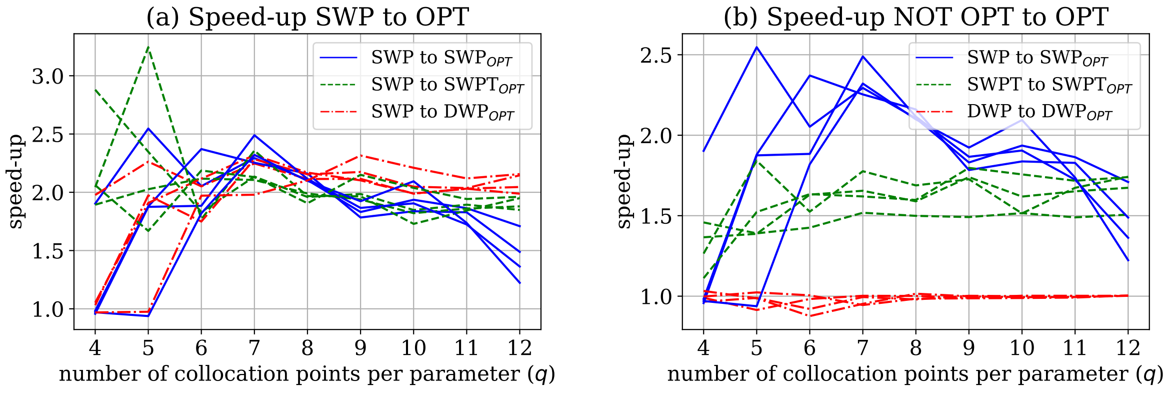


Figure 6.27: Speed-up for T_{Prop} for scenario 2: evacuation of a building with separated families. (a) contains the speed-ups of SWP compared to SWP$_{OPT}$, SWPT$_{OPT}$, and DWP$_{OPT}$. (b) contains the speed-ups for SWP to SWP$_{OPT}$, SWPT to SWPT$_{OPT}$, and DWP to DWP$_{OPT}$. Each plot contains 4 speed-up lines (of identical colour and line style) that corresponds to the different number of cluster nodes $cn = 2, 3, 4, 5$.

To definitely answer which uncertain parameter has the most influence on the individual black-box model runtime T_S^i and therefore on the propagation time T_{Prop}, the sensitivity indices (Section 2.3) have been calculated. As can be seen in Figure 6.28(a) for the first-order sensitivity indices, $perc_{fam}$ is the most important parameter for all tested q. v_{child} contributes more than v_{parent}, but both contributions are very small compared to $perc_{fam}$. This was already assumed in Figure 6.22 and Figure 6.25 because $perc_{fam}$ was already identified to be the dominating factor, but here, the relation and the impact of the parameters is much more visible. It can also be observed that there is almost no interaction between the parameters which is shown by si_int. For that reason, the total-order sensitivity indices in Figure 6.28(b) looks almost the same as the first-order sensitivity indices in Figure 6.28(a).

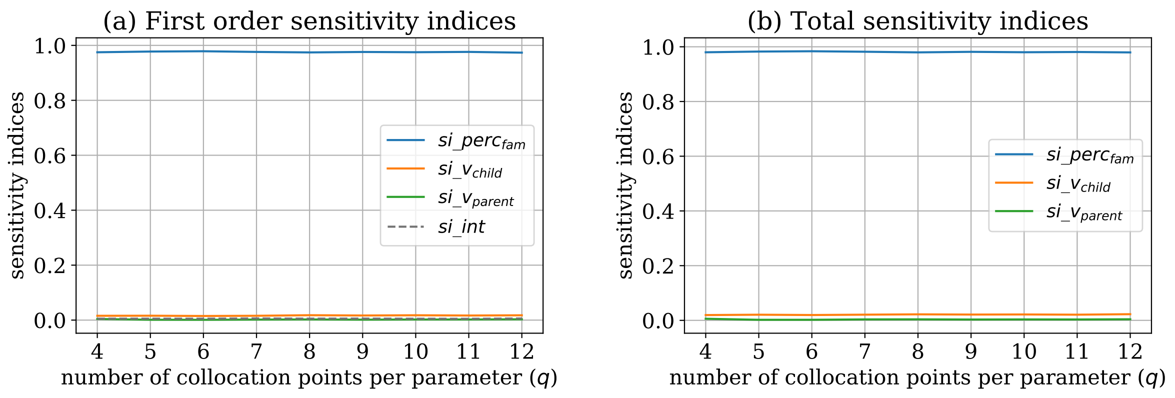


Figure 6.28: Sensitivity indices for the individual black-box model run runtimes T^i_S with all of the three uncertain parameters (Table 6.8) for scenario 2: evacuation of a building with separated families. (a) shows the first-order sensitivity indices and (b) shows the corresponding total-order sensitivity indices.

Despite the large absolute errors ϵr_i and relative errors $\epsilon r_{i,\text{rel}}$ in the prediction of $\mathbb{T}^i_S$, it is possible to significantly reduce the propagation time T_{Prop} with the dynamic and optimised scheduling strategies using the runtime predictor rp_N. SWPT$_{OPT}$, DWP, and DWP$_{OPT}$ work best for this scenario. For the scheduling, the prediction errors in the black-box model runs with the small runtimes $\mathbb{T}^i_S$ are not as essential. It is more important that the individual black-box model runs with the long runtimes $\mathbb{T}^i_S$ are identified correctly–but not necessarily predicted exactly! The reason why the scheduling strategies with the higher dynamic component performs better is most likely due to the large prediction errors in $\mathbb{T}^i_S$.

6.4.5 Summary

The subject of this scenario was the evacuation of a building with separated families under uncertain conditions. For the simulation of the scenario, the Vadere simulator with the family affiliation model (FAM) is used. To quantify the uncertainty, again, stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is used as the UQ method of choice.

The FAM has three parameters which are defined to be uncertain (Table 6.8), which leads to a 3D UQ problem. The UQ simulation results in a significant variation of the maximum evacuation time ev_t (Table 6.10) for the chosen uncertain parameters. This underlines the importance of a) the usage of the FAM to consider evacuation scenarios with separated families, and b) the use of UQ methods to quantify the uncertainty to obtain more realistic knowledge about values for the evacuation time ev_t.

A global sensitivity analysis (Section 2.3) provides detailed information about the importance and the interaction of the parameters for each time step of the simulation. Now, the individual contribution of each of the three uncertain FAM parameters can be determined, which is very valuable for the modeller of the FAM itself as well as for the users of the FAM.

The reconstruction of the model function (Section 3.5) of the maximum evacuation time ev_t using the generated gPCE gives, again, valuable insights of the possible output values with different combinations of the uncertain parameter values.

All simulations in the context of this scenario are performed with the UQEF (Chapter 5) on the Linux-Cluster CoolMUC2. The FAM shows significant differences in the runtime depending on the input parameter values. A detailed analysis of the runtime variation for the simulation scenario is performed under the given uncertainties. With the developed runtime prediction mechanism in Section 4.5, a runtime predictor rp_N (Section 4.5) is constructed for the FAM scenario, and its prediction quality is measured. The rp_N can predict the runtime of a single black-box model run, with relative errors mostly below 0.1.

For the runtime visualisation, the constructed rp_N is used to plot the predicted runtimes as 3D plots for the different parameter variations of the three uncertain parameters. This allows to see the predicted runtime propagation values for many different parameter value variations.

Furthermore, the propagation runtime T_{Prop} for the three standard scheduling strategies (Section 4.4) and the optimised versions (Section 4.6.1) is analysed for different number $q = 4, 5, \ldots, 12$ of collocation points per parameter and cluster nodes $cn = 2, 3, 4, 5$. The scheduling strategy has a huge impact on the propagation runtime T_{Prop}. The dynamic and the optimised versions can reduce the amount of idling and therefore reduce the propagation runtime T_{Prop} depending on the scheduling strategy. In this scenario, the DWP and DWP$_{OPT}$ scheduling have the fastest propagation time and should be preferred to the others for a fast and efficient UQ simulation. From the worst scheduling (SWP) to the best scheduling (DWP or DWP$_{OPT}$) a speed-up of about 1.7-2.3 is observed.

With a global sensitivity analysis using the constructed rp_N, one parameter is identified as the main cause for the runtime variation, while the others have only little influence. It is now possible to predict and to automatically optimise the propagation time T_{Prop} of the FAM scenario with the investigated scheduling strategies. All this can be achieved on different computing systems with a different number of computing nodes because UQEF automatically distributes the work—depending on the chosen scheduling strategy—among the computing nodes.

6.5 Scenario 3: Utilisation of a campus

In pedestrian dynamics, not only the utilisation of a single building [187, 188, 76] is of interest. Also, the utilisation of a whole campus, a combination of streets and several connected buildings can be relevant [22, 18, 33]. On a campus, like a university or a huge company, there is something like a daily routine of the pedestrians [116]. The pedestrians enter the campus, move between the buildings, and then leave the campus at the end of the day. Exactly this behaviour is the subject of this scenario and is further investigated under uncertain conditions, with the combination of the closed observables surrogate model (COSM) (Section 4.7.1), to efficiently (fast) quantify the uncertainties.

The details of this scenario[5] are described in Section 6.5.1. The setup for the UQ simulation can be found in Section 6.5.2, the construction of the COSM in Section 6.5.3, and the UQ results in Section 6.5.4 as well as the computational efficiency and error results in Section 6.5.5. The scenario is summarised in Section 6.5.6.

6.5.1 TUM campus utilisation

The goal of this scenario is to simulate the utilisation of a campus with several distributed buildings. The pedestrians follow a daily routine and move between the buildings, while the utilisation of each building is measured and analysed.

[5]This scenario is the result of a collaborative work with Dr Felix Dietrich. It is a continuation of the joint work that has been originally published in [28], where the COSM (Section 4.7.1) is used to support real-time decision systems in the context of the evacuation of a train station.

For that, the campus Garching of the Technical University of Munich (TUM) is modelled partially on a coarse level with a Vadere scenario, which is visualised in the left side of Figure 6.29. The daily routine of the considered pedestrians is as follows: They arrive at the campus by the underground and then move directly to the TUM Math & Informatics building (middle part of Figure 6.29). After working a while, they go to a canteen for lunch. On the campus, there is the TUM mensa and the IPP canteen, and the pedestrians can choose between both alternatives (right side of Figure 6.29). After lunch, they move back to work to the TUM Math & Informatics building. At the end of the working day, they walk to the underground and leave the campus.

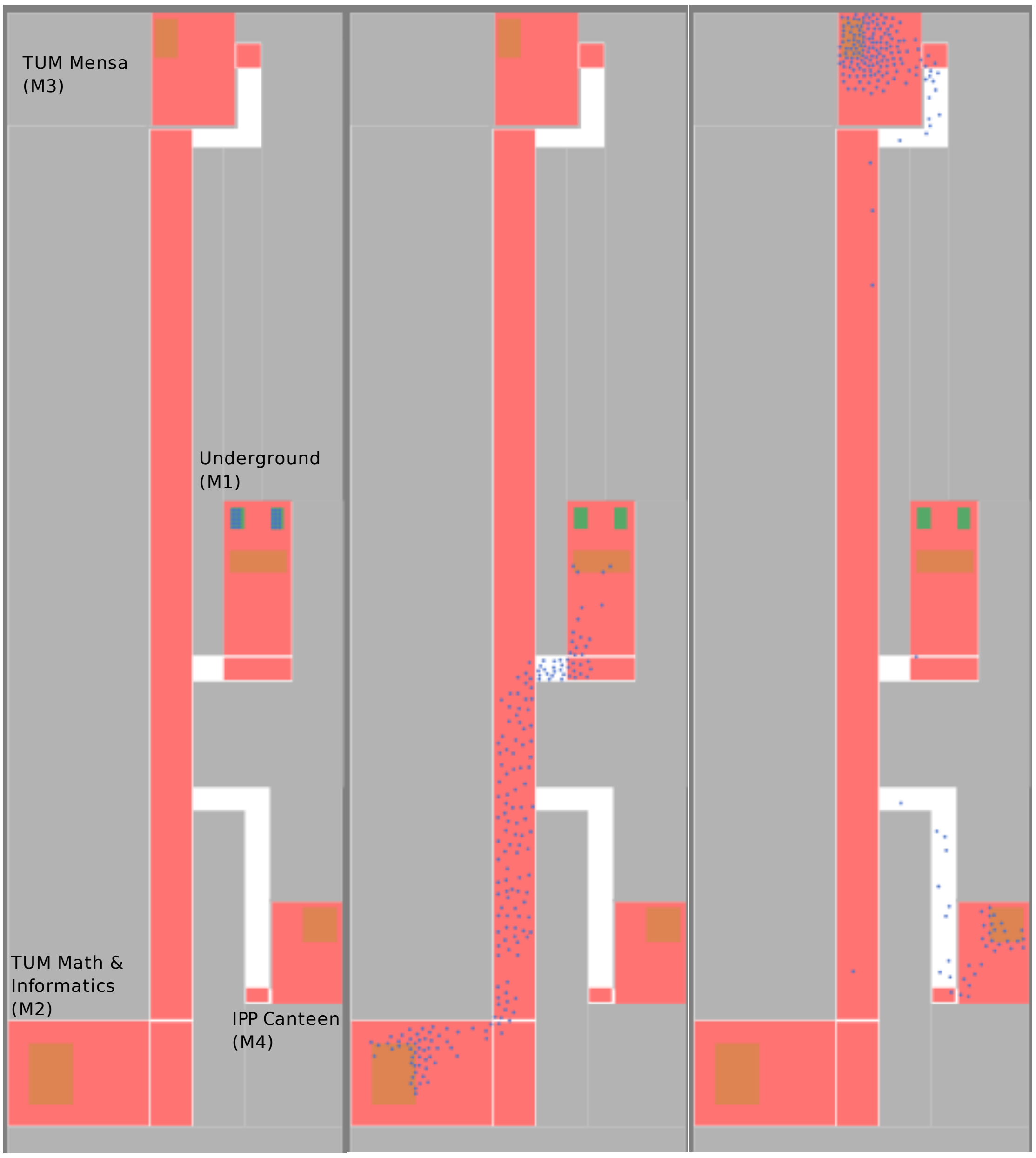


Figure 6.29: Illustration of campus for scenario 3: utilisation of a campus. On the left, the partially modelled TUM campus with the building names is visualised. On the middle, the pedestrians move from the underground to the TUM Math & Informatics building. The right visualisation shows the pedestrians in the TUM and IPP canteen.

For the sake of simplicity, the buildings are modelled with one floor, and there are 200 pedestrians in the scenario. The working time at the TUM Math & Informatics building is the same for all pedestrians, as a simplification. Some of the pedestrians do always go to the TUM mensa, while the others prefer the IPP canteen. In this setting, the pedestrians do not switch their preference during a single simulation run. There are several days simulated to see the cycles in the utilisation of the buildings that the daily routines of the pedestrians produce. The time the pedestrians work, eat, move, or are not present at the campus (during the night) is shortened to reduce the computational effort, which means there is no direct mapping between the time steps in the simulation and a real 24 hour day. After a one day period, the pedestrians arrive at the underground, and they are not removed from the scenario. Instead, they wait for some time in the underground and then a new cycle starts where the pedestrians move again to the TUM Math & Informatics building.

The time step size for the Vadere scenario is set to 0.4 and the finish time to 6,000, which ensures to produce several cycles. For the basic locomotion model, the OSM with its default values is used. The preference for a canteen is modelled with two different sources (green rectangles in Figure 6.29) for the pedestrians, where the cycle with the specific canteen is defined. Around each target (orange rectangles), measure zones (light red rectangles) are defined, which counts the number of pedestrians within this measure zone for every time step of the simulation. Table 6.11 lists the measure zones that are used for the values of interest.

Denotation	Measure zone
$M1(Underground)$	Underground
$M2(TUM)$	TUM Math & Informatics building
$M3(TUM_{mensa})$	TUM mensa building
$M4(IPP_{canteen})$	IPP canteen building

Table 6.11: Denotation of the measure zones for scenario 3: TUM campus utilisation.

6.5.2 UQ simulation setup

The scenario is modelled with four uncertain parameters which are listed in Table 6.12. Because each pedestrian has a different preference on the favourite canteen, this is defined to be uncertain and modelled with the IPP canteen ratio (ir) parameter. Because the TUM mensa has more capacity as the IPP canteen, a uniform distribution between 0.1 and 0.3 is assumed, which means that 70%–90% percent of the pedestrians prefer the TUM mensa and 10%–30% the IPP canteen. The pedestrians do eat for some time in the canteen—but the exact amount of time is not known[6]. This is modelled with two different normally distributed parameters, with i_{rtime} for residence time in the IPP canteen, and t_{rtime} for the residence time in the TUM mensa. The walking speed of pedestrians is modelled with the v parameter, and according to [154], a

Parameter	Description	Distribution
ir	IPP canteen ratio [%]	$U(0.1, 0.3)$
i_{rtime}	Residence time in IPP canteen [$minutes$]	$N(50, 1)$
t_{rtime}	Residence time in TUM mensa [$minutes$]	$N(60, 1)$
v	Walking speed of pedestrians [m/s]	$U(1.3, 1.8)$

Table 6.12: List of uncertain parameters and their distributions for scenario 3: utilisation of a campus. The parameters ir and v are uniformly distributed between the specified minimum and maximum values. For the residence times i_{rtime} and t_{rtime}, normal distributions are used.

[6]As a simplification, it is not considered that the pedestrians usually walk and eat in small groups and join and leave the canteen together. This could be modelled in future investigations with the SIMA (that is used in scenario 1 (Section 6.3)) or with the FAM (that is used in scenario 2 (Section 6.4)), depending on the scenario.

uniform distribution between 1.3–1.8 $[m/s]$ is assumed.

To analyse the utilisation of the buildings under these uncertain conditions, the four measure zones $M1(Underground)$, $M2(TUM)$, $M3(TUM_{mensa})$, and $M4(IPP_{canteen})$ that are defined in Table 6.11 are used for the value of interest. They are the subject of the uncertainty quantification. The measured values are taken for every time step in the simulated scenarios, to reconstruct a fine-grained utilisation for each building.

The UQ method of choice is the stochastic collocation with the pseudo-spectral approach (Section 2.2.2). For a detailed analysis, $q = 10$ number of collocation points per dimension are used. This results in $Q = 10^4 = 10{,}000$ number of collocation points and, hence, 10,000 number of black-box model runs in the propagation phase. The highest order of the orthogonal polynomials $\Phi_j(\zeta)$ in the gPCE (Equation (2.7)) is set to $P = 5$ to ensure stable numerical moments for the QoI. The truncation value is determined via a parameter study.

The scenario-specific classes that are derived from UQEF are visualised in Figure 6.30. The

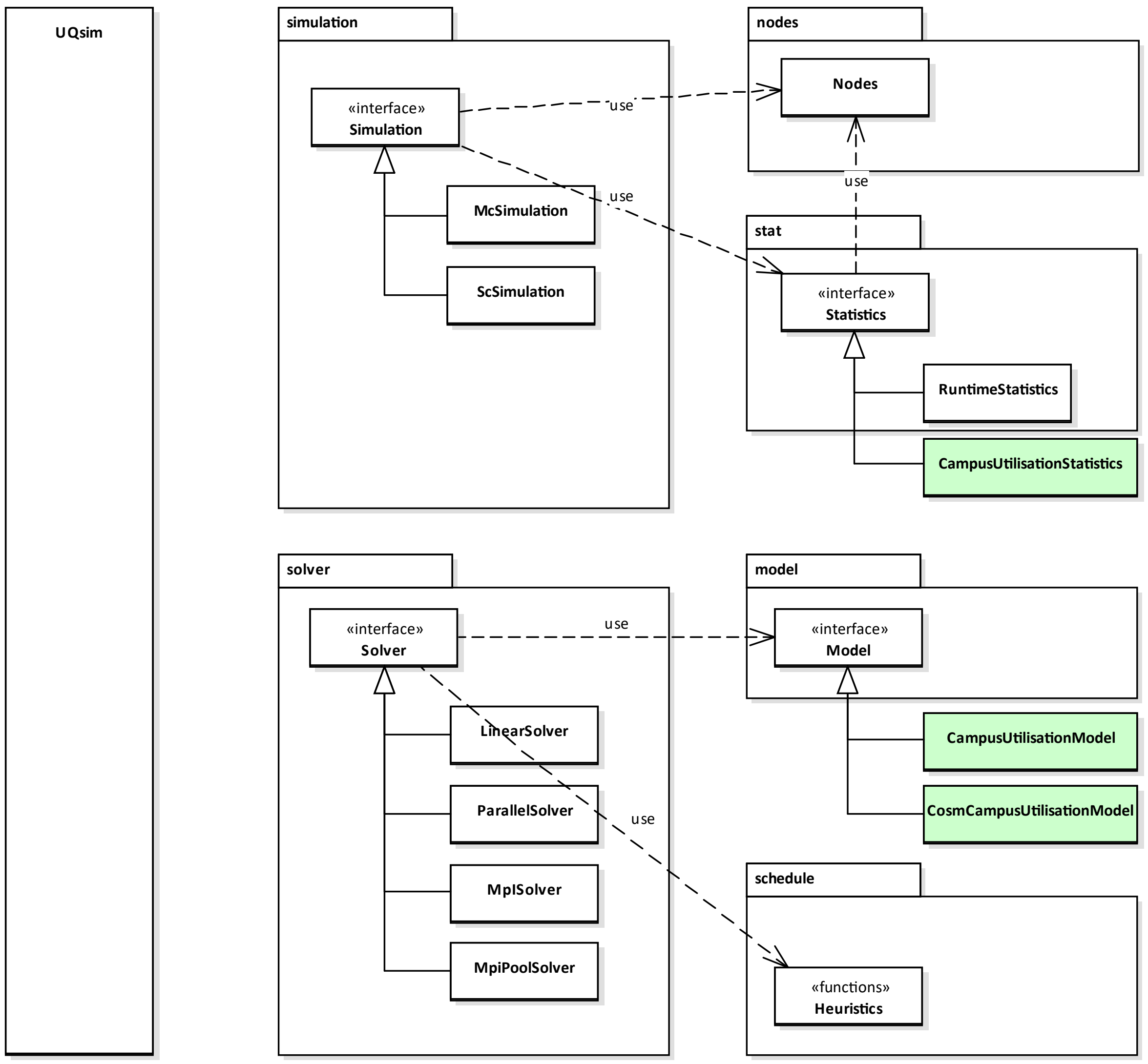


Figure 6.30: Illustration of derived classes for scenario 3: utilisation of a campus. The light green rectangles indicate the implemented custom classes for this scenario. See Section 5.1 for a detailed description of the overall UQEF architecture.

CampusUtilisationModel class contains all the source code to call the Vadere simulator. Because the execution of the COSM simulator differs significantly from calling the Vadere simulator, a separate implementation exists in the *CosmCampusUtilisationModel* class. Both *Model* implementations are implemented in such a way that they return the values of interest in the same style and order. Therefore, a single *CampusUtilisationStatistics* class is suitable to calculate the QoI statistics for both *Model* implementations.

Figure 6.31 shows the corresponding UQEF simulation program architecture. Both *Model* implementations are used in the *campus utilisation* Python script but registered under different model names. As explained in Section 5.2 and Section 5.4, it is possible to choose the model by using the *UQsim* `"--model"` command line parameter. The *CampusUtilisationStatistics* implementation is registered two times, for each model implementation, respectively. This allows to reuse the *campus utilisation* Python script and further reduces the amount of source code for the scenario.

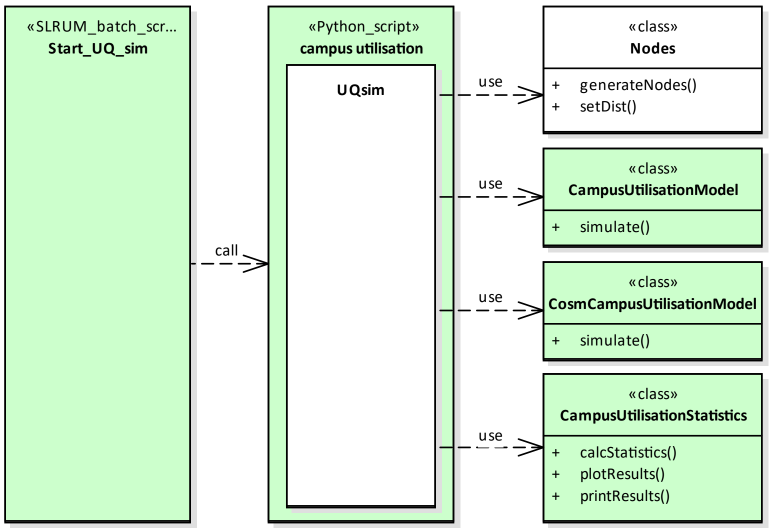


Figure 6.31: Illustration of the simulation program architecture for scenario 3: utilisation of a campus. The light green rectangles indicate the custom implementation parts for this scenario. The white rectangles indicates the UQEF classes that are instantiated and used directly, but are not changed.

The focus on this scenario is the runtime improvement that can be achieved by using the generated surrogate model (COSM) instead of using the Vadere simulator. Following the advice in Table 4.9 on how to choose a suitable scheduling strategy, DWP is selected as the scheduling strategy, because it automatically distributes the workload relatively evenly across the computing nodes, without the knowledge of the T_S^i runtime being required.

To assess the quality of the generated COSM when using it to quantify the uncertainty, several UQ simulations with a varying $q = 2, 3, \ldots, 10$ have been performed. To get the results in a reasonable time, different cluster nodes $cn = 1, 2, 4, 6, 8$ are used. Table 6.13 shows for each q the number of used cluster nodes cn. An important argument for using a surrogate model is to

q	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Q	16	81	256	625	1,296	2,401	4,096	6,561	10,000
cn	1	2	2	4	4	4	6	6	8

Table 6.13: List of performed simulations for scenario 3: utilisation of a campus. For each q, the resulting number of collocation points Q and the used number of cluster nodes cn is listed.

achieve faster propagation times T_{Prop}. Therefore, the propagation runtime T_{Prop} is measured for all performed UQ simulations and the COSM runtimes are compared to the Vadere runtimes.

6.5.3 Construction of the COSM

In the offline phase of the COSM (Section 4.7.2), it is required to sample the original microscopic model—the Vadere simulator—several times. The sampling is based on a full grid of 5 values per parameter. This leads to $N_{ir} \cdot N_{i_{rtime}} \cdot N_{t_{rtime}} \cdot N_v = 5^4 = 625$ number of parameter sets for the sampling of Vadere. Table 6.15 lists the corresponding values for each parameter. An equidistant sampling between a predefined range is chosen. The range is based on the values of Table 6.12 for the probability distributions of the parameters.

Parameter	Values				
ir	0.1	0.15	0.2	0.25	0.3
i_{rtime}	40.0	45.0	50.0	55.0	60.0
t_{rtime}	50.0	55.0	60.0	65.0	70.0
v	1.3	1.425	1.55	1.675	1.8

Table 6.15: Sampling values for the parameters ir, i_{rtime}, t_{rtime}, and v to create the COSM. This results in $N_{ir} \cdot N_{i_{rtime}} \cdot N_{t_{rtime}} \cdot N_v = 5^4 = 625$ number of parameter sets in the full grid sampling.

The observed values of interest (see Table 6.11) are very noisy, because the four parameters ir, i_{rtime}, t_{rtime}, and v significantly influence the individual movements of the pedestrians within the Vadere simulator, and therefore the utilisation of the buildings. To obtain smoother values for the construction of the COSM, the following aspects have been considered:

- To simplify the dynamic of the data to that of the slow manifold [87] when setting up the COSM, the first 8,000 time steps of the simulation runs are not taken into account. By comparing the UQ results with Vadere to that of the COSM, the same time steps (or even some of the time steps to match the time series data again) of Vadere also have to be skipped. This allows a direct comparison of the measure zone data with detailed error results as described in Section 4.7.3.
- The values of interest do change in a different velocity for the different measure zones, which has the consequence that the resulting values of the COSM do slightly diverge. To eliminate this behaviour, the change of the values (the acceleration) is additionally taken into account as a separate dimension in the input data to generate the closed observables.
- The resulting number of dimensions for the closed observables is 6. With a data file of about 23.99 MiB, all data for the closed observables are stored on a disk.
- For the interpolation of the sampled input data from Vadere to the output map of the COSM, the nearest-neighbour interpolation of the `scipy.interpolate.NearestNDInterpolator` Python package is used.

6.5.4 Numerical results

This section presents the numerical results of the UQ simulation that is gathered with the constructed COSM as the surrogate model. For the quantification of the uncertainties, $q = 10$ number of collocation points have been used for each uncertain parameter. Hence, $10^4 = 10{,}000$ simulation runs with the COSM have been performed.

To analyse the general utilisation of the four modelled buildings (cf. Table 6.11) in this scenario, the mean utilisation values $\mu_{i,M1}(Underground)$, $\mu_{i,M2}(TUM)$, $\mu_{i,M3}(TUM_{mensa})$, and $\mu_{i,M4}(IPP_{canteen})$ are plotted in Figure 6.32 for every time step. As can be seen, the daily cycles are nicely visible for the whole simulated time period: After the arrival in the underground and the first working time at the TUM Math & Informatics building in the mid-morning, the pedestrians move to one of both canteens, return to work, and finally move to the underground. On the light blue area around each line (5th to 95th percentile) the influence of the uncertainty becomes evident.

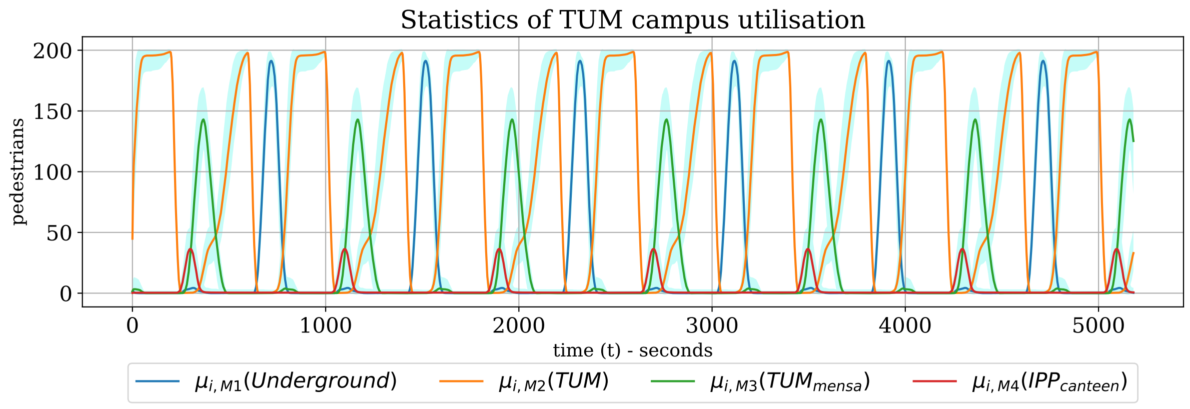


Figure 6.32: QoI results of the mean utilisation for scenario 3: utilisation of a campus. For each of the four measure zones (see Table 6.11), the mean utilisation for every time step is plotted. The light blue area around each line is the 90% interval of the values (5th to 95th percentile).

Figure 6.33 shows the corresponding standard deviations $\sigma_{i,M1}(Underground)$, $\sigma_{i,M2}(TUM)$, $\sigma_{i,M3}(TUM_{mensa})$, and $\sigma_{i,M4}(IPP_{canteen})$ for each measure zone on each time step. The values indicate a high standard deviation for each measure zone. Especially in the periods when the first pedestrians enter and leave the buildings, the deviations are higher. For both canteens, the standard deviation is also high for the residence time of the pedestrians, because the residence time is comparably short, and pedestrians constantly enter and leave the buildings.

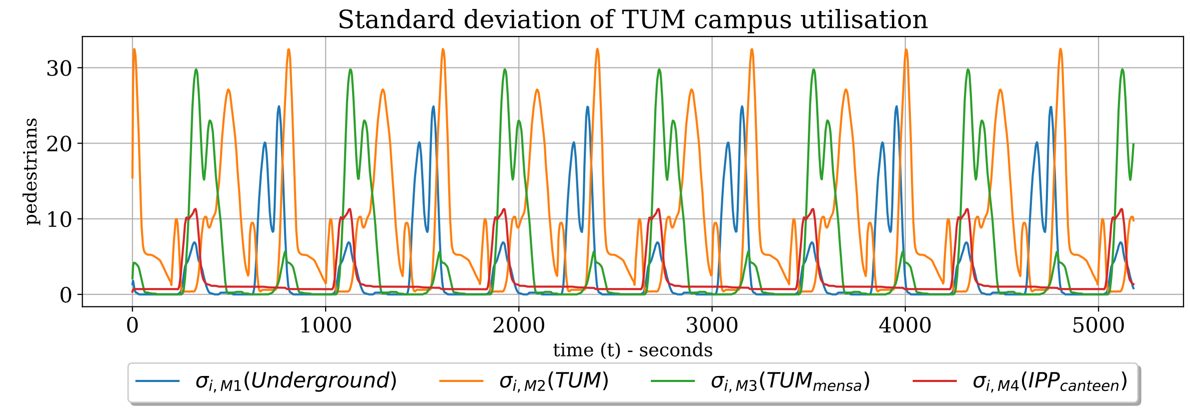


Figure 6.33: Standard deviation of the utilisation for scenario 3: utilisation of a campus. For each of the four measure zones (see Table 6.11), the standard deviation for every time step is plotted.

With the sensitivity indices, the influence of each parameter on the resulting utilisation can be determined. Figure 6.34 shows the first-order sensitivity indices for each measure zone for each time step. As expected, the influence changes heavily over the daily time period. For all of the four buildings, the utilisation is highly influenced by the speed parameter v.

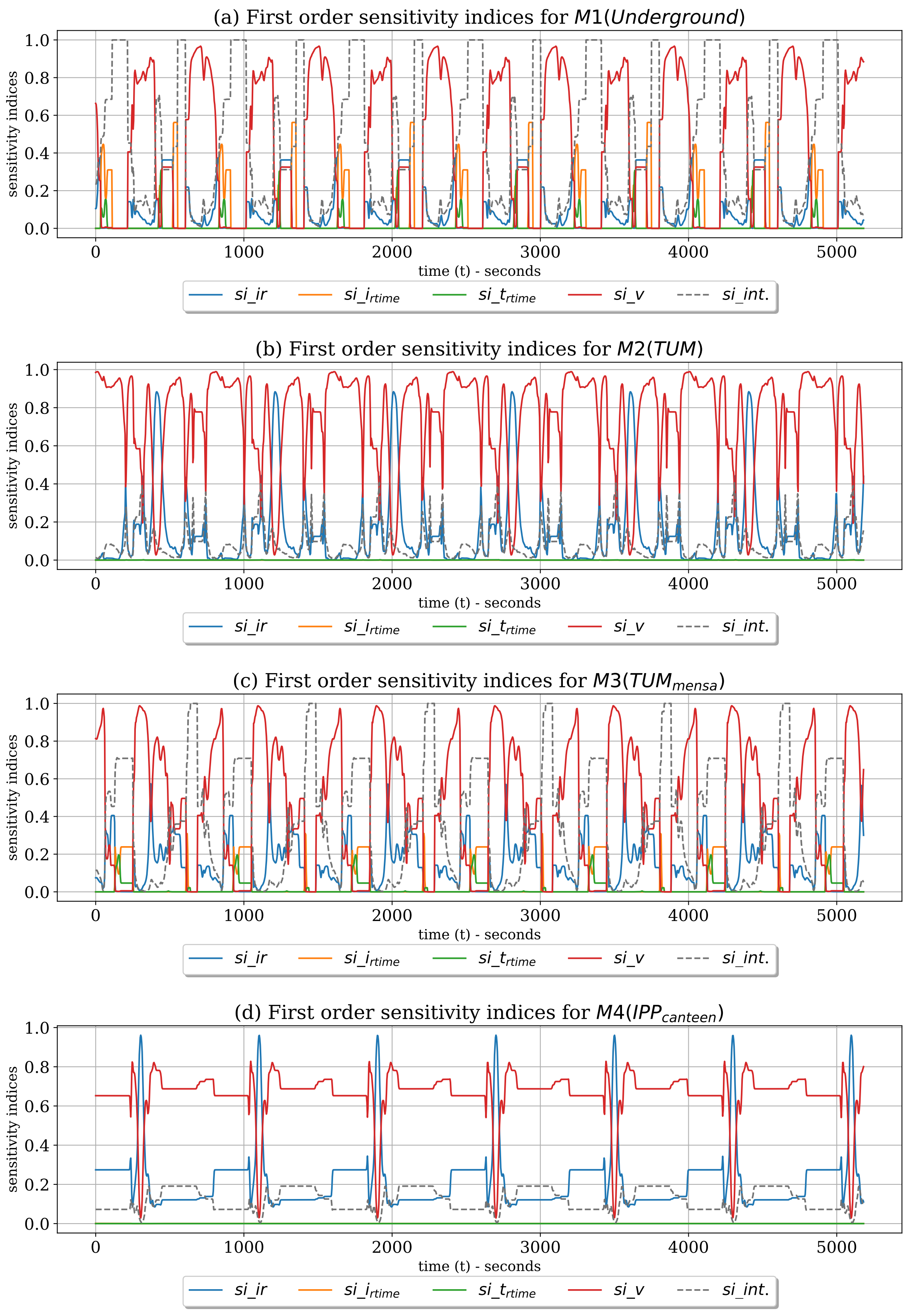


Figure 6.34: First-order sensitivity indices for the utilisation of the buildings with all of the four uncertain parameters (Table 6.12) for scenario 3: utilisation of a campus. (a) contains the values for $M1(Underground)$, (b) for $M2(TUM)$, (c) for $M3(TUM_{mensa})$, and (d) for $M4(IPP_{canteen})$.

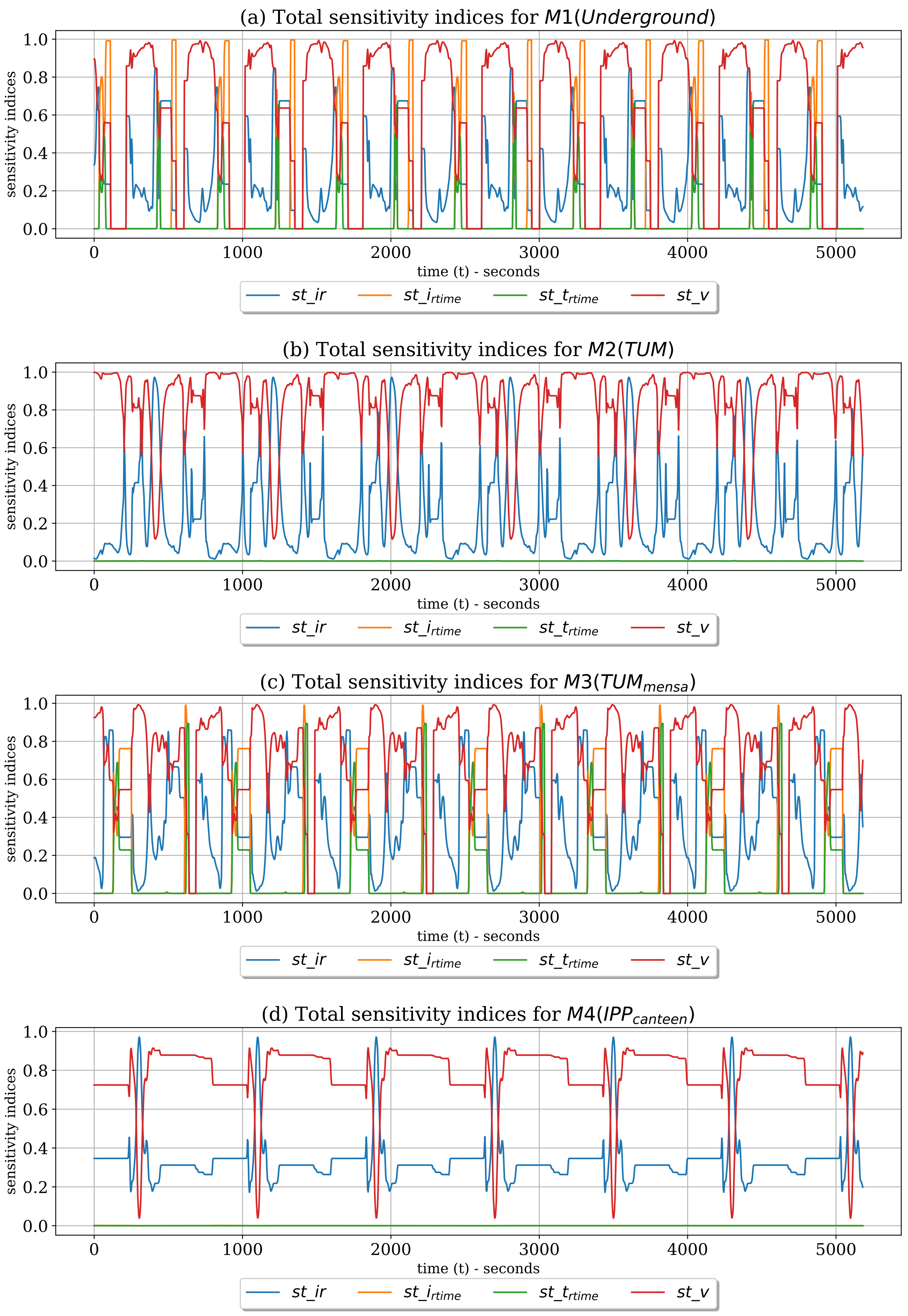


Figure 6.35: Total-order sensitivity indices for the utilisation of the buildings with all of the four uncertain parameters (Table 6.12) for scenario 3: utilisation of a campus. (a) contains the values for $M1(Underground)$, (b) for $M2(TUM)$, (c) for $M3(TUM_{mensa})$, and (d) for $M4(IPP_{canteen})$.

In Figure 6.34(a) and Figure 6.34(c), also, a high interaction between the parameters can be observed. In Figure 6.35, the corresponding total-order sensitivity indices are shown. It is expected that the individual residence times in the canteen only influence the utilisation of that canteen. This is true for st_t_{rtime} in Figure 6.35(d). But in Figure 6.35(c) there is a noticeable influence based on st_i_{rtime} visible. The behaviour can be caused by small jams and counterflows when the pedestrians move to a canteen which results in a temporary slow down of the moving speed on the common path and on the fork to the IPP canteen.

Figure 6.36 gives further possibilities to investigate the general utilisation under the given uncertain conditions. The relationship of each measure zone to all others is visualised. For that,

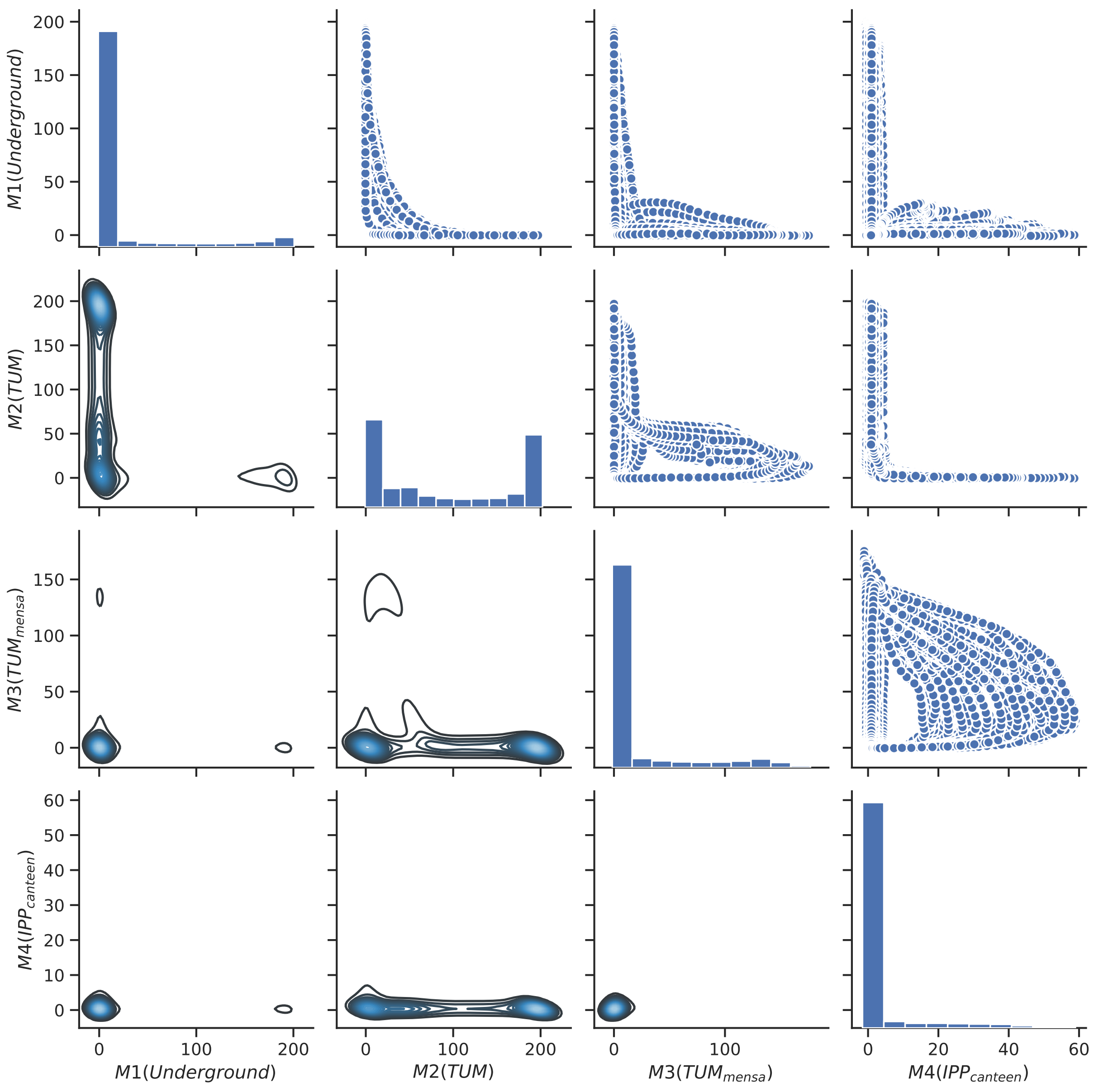


Figure 6.36: Pairwise relationship of the utilisation for the four buildings (Table 6.11) in scenario 3: utilisation of a campus. The diagonal axis contains the histogram of the utilisation for each building. The upper right side contains the scatter plot of the utilisation values and the lower left side the contour plots.

the generated gPCE $u_N(x, t, \boldsymbol{\zeta})$ (Equation (2.7)) is sampled on 100 nodes which have been drawn from the multivariate probability density function of the uncertain parameters from Table 6.12.

That results in $100 * 6{,}000 = 600{,}000$ values for each measure zone, including the utilisation for each time step. There is no information about the simulation time in these plots, only the utilisation values are considered.

On the diagonal axis, the histogram shows the relative frequency of the utilisation values for each measure zone. The scale on the y scale does not show the correct values for the histogram—it is meant for the other diagrams, and the height of the bar only shows the qualitative frequency for each bar (bucket). For the histogram, the value range of the utilisation is split into equally sized buckets, here 10. All 600,000 values are placed into one of the buckets, and the number of values inside each bucket is the height of the bars. For $M1(Underground)$, most of the time there are almost zero pedestrians in the building; for $M2(TUM)$, it shows two accumulations of the values for about zero and 200 pedestrians; for $M3(TUM_{mensa})$ and $M4(IPP_{canteen})$, almost zero pedestrians are in both buildings for most of the simulation time, which is plausible because the pedestrians do also walk for some time and are not inside of one of the four buildings.

The scatter plots (upper right triangle) inform about the distribution of the values. Depending on the other measure zone, the values are plotted. The scatter plot is good to see where the values are, but it does not deliver the information about the frequency (the number of values on the same position). For this, the contour plot (lower left triangle of plots) are useful. The contour lines in the contour give the information where most values lie. They are calculated through a kernel density estimate, and the colour shows the density: black lines show a small density and blue lines a higher density. Until a certain density, the area is left blank.

For example (as can be seen on column one), if almost no one is in measure zone $M1(Underground)$, then it is very likely that the pedestrians are not in $M3(TUM_{mensa})$ and $M4(IPP_{canteen})$, but in $M2(TUM)$. The separated black contour lines, when about 200 pedestrians are in $M1(Underground)$, indicates that there are almost no pedestrians in the other buildings. Column two gives the information, that when about zero or 200 pedestrians are in $M2(TUM)$, there are mostly no pedestrians in $M3(TUM_{mensa})$ and $M4(IPP_{canteen})$. In the contour plot of $M2(TUM)$ vs $M3(TUM_{mensa})$, the single black contour line, which is separated from the large clustered values, indicates that if about 10–50 pedestrians are in $M2(TUM)$, 110–150 pedestrians are likely in $M3(TUM_{mensa})$. The scatter plot of $M4(IPP_{canteen})$ vs $M3(TUM_{mensa})$ (row three, column four) has a strong bellied form of the values which one might think that a lot of values are in this bellied part. Together with the corresponding contour plot (row four, column three), it can be seen on the contour lines, that most of the values are about zero, which means that if no one is in $M3(TUM_{mensa})$, also no one is in $M4(IPP_{canteen})$.

The daily utilisation cycles for each measure zone is visible in Figure 6.37. It shows the first three principle components (obtained by a principal component analysis (PCA) [79] with the `numpy.linalg.svd` Python package) of the mean utilisation for each building. The PCA values

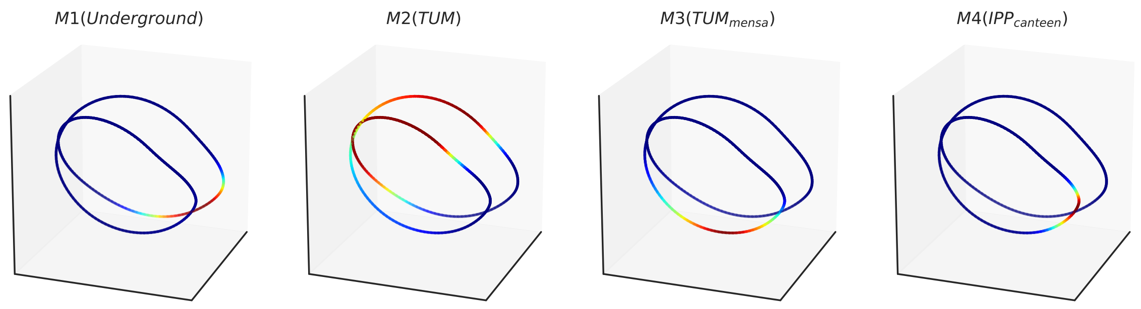


Figure 6.37: Visualisation of the first three principal components for the four buildings (Table 6.11) in scenario 3: utilisation of a campus. A dark blue colour indicates less or no pedestrians at this point and a dark red colour a high utilisation for the measure zone.

have a shape of a loop which is wrapped around two times that do not actually touch each other; they are a single loop projected into a 3D space like this. The blue colour along the circles indicates that there are less or no pedestrians in the measure zone. The deep red colour indicates that there are a lot of pedestrians in the measure zone. The two working periods for the pedestrians are nicely visible in $M2(TUM)$. In $M1(Underground)$, only one continuous red area is visible, because the start and the end of the daily cycle lie somewhere in the middle of the red area.

If this uncertainty analysis is, e.g. used for planning of such a campus, or to estimate the required staff in the canteen, the mean utilisation may help a lot. Additionally, the 90% confidence interval (5th to 95th percentile) is useful to see the upper and lower boundaries. For a more detailed planning, it is useful to know the intensity of the influence of the parameters. This allows to influence such situations actively, e.g. having the required stuff available, routing of the pedestrians, or reducing the waiting times in the food distribution. Another possibility is to try to influence the pedestrian's preference (e.g. with marketing campaigns) on the canteen and predict than the expected utilisation. Moreover, because for this scenario the COSM is generated and available as a surrogate, the UQ simulation can be repeated very fast with different probability distributions (and values for the distributions) to obtain the results in a reasonable time.

6.5.5 Computational efficiency and error results

This section gives an overview of the computational efficiency of the performed UQ simulations with the constructed COSM. After that, the quality of the resulting QoI is determined by several error measurements of the COSM compared to the Vadere simulator.

Figure 6.38(a) contains the measured propagation runtimes T_{Prop} for the performed UQ simulations with the COSM and the Vadere simulator as the model. It can be seen that with Vadere, the propagation runtime T_{Prop} growth heavily with an increasing q. Whereas with the COSM, the T_{Prop} is constantly very small. In Figure 6.38(b), the same values are plotted with a logarithmic scale: Now it is visible that T_{Prop} with the COSM also increases, but it is a factor of about 10^2 between the COSM and the Vadere for all performed q. As a reminder, Table 6.13

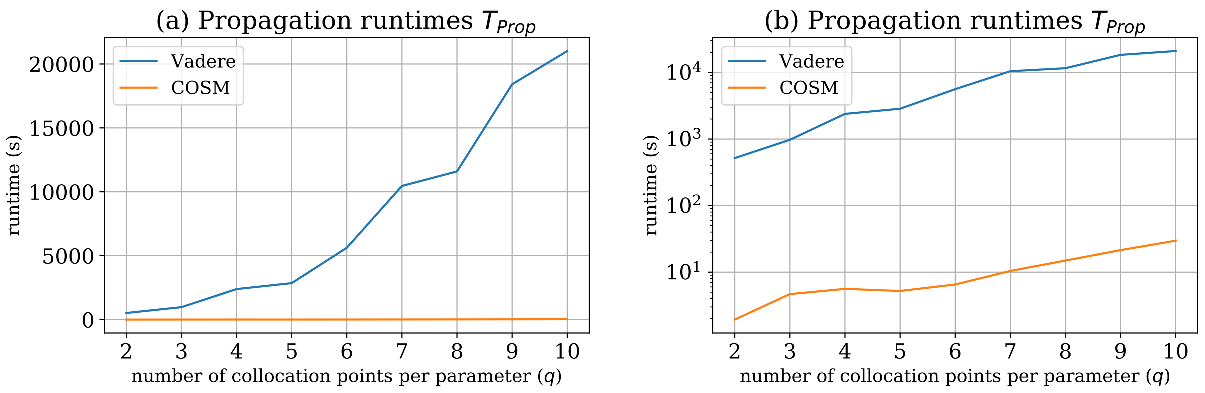


Figure 6.38: Measured propagation runtimes T_{Prop} using the COSM and the Vadere simulator as the model for scenario 3: utilisation of a campus. (a) contains the absolute propagation runtime T_{Prop} with a linear scale and (b) with a logarithmic scale with a varying number $q = 2, 3, \ldots, 10$ of collocation points per parameter.

contains the number of computing units cn for each performed q. To create the COSM, $5^4 = 625$ black-box runs as denoted in Table 6.15 have been performed, which took about 2852.8 seconds (cf. the entry for $q = 5$ in Table 6.16) with dynamic scheduling and $cn = 4$ cluster nodes.

Additionally, the creation of the closed observables took about 584 seconds[7]. In summary, an initial overhead of about $2852.8 + 584 = 3436,8$ seconds was required to construct the COSM.

The achieved speed-ups are visible in Figure 6.39 and Table 6.16. For all performed UQ simulations, the speed-ups are ≥ 200. For $q = 7$, a speed-up factor of about 1,000 could be achieved. This impressively shows the computational efficiency of the COSM in the online phase.

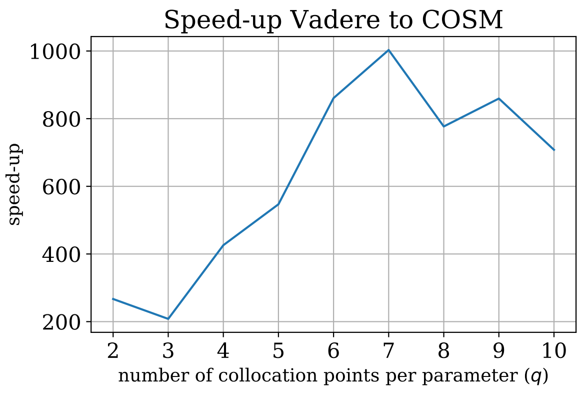


Figure 6.39: Speed-up for T_{Prop} for scenario 3: utilisation of a campus. The speed-up is calculated for a varying number $q = 2, 3, \ldots, 10$ of collocation points per parameter.

q	Vadere	COSM	Speed-up
2	517.4	1.9	266.9
3	974.0	4.6	208.1
4	2,389.3	5.6	425.9
5	2,852.8	5.2	546.9
6	5,618.4	6.5	860.8
7	10,455.3	10.4	1,002.9
8	11,594.2	14.9	776.9
9	18,410.1	21.4	859.4
10	21,013.9	29.6	707.9

Table 6.16: List of values for the propagation runtimes T_{Prop} and the achieved speed-ups for the different $q = 2, 3, \ldots, 10$ of collocation points per parameter

This speed-up is at the expense of the prediction quality, which is determined in the following. For $q = 10$, Figure 6.40 shows the absolute mean error ϵs_i of the utilisation for every time step. Besides at the very beginning, the values are low and reach up to 11.9 for $M1(Underground)$, 23.5 for $M2(TUM)$, 9.23 for $M3(TUM_{mensa})$, and 4.0 for $M4(IPP_{canteen})$ for singular phases. The corresponding relative errors $\epsilon s_{i,rel}$ in Figure 6.41 show for most time steps small values. For some points (besides the very first time steps) maximum relative errors $\epsilon s_{i,rel}$ of about 6% for $M1(Underground)$, 11% for $M2(TUM)$, 6% for $M3(TUM_{mensa})$, and 11% for $M4(IPP_{canteen})$ occur.

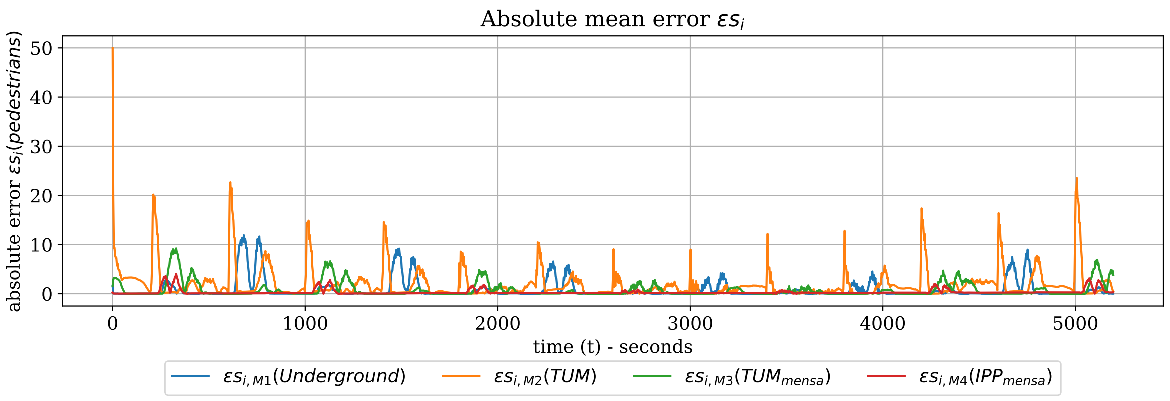


Figure 6.40: Absolute mean error ϵs_i for scenario 3: utilisation of a campus. The values are plotted for all measure zones (Table 6.11) on every time step for $q = 10$ number of collocation points per parameter.

[7]The closed observables are computed by Dr Felix Dietrich, on a Microsoft Surface Pro 4 with a single-threaded software written in Python.

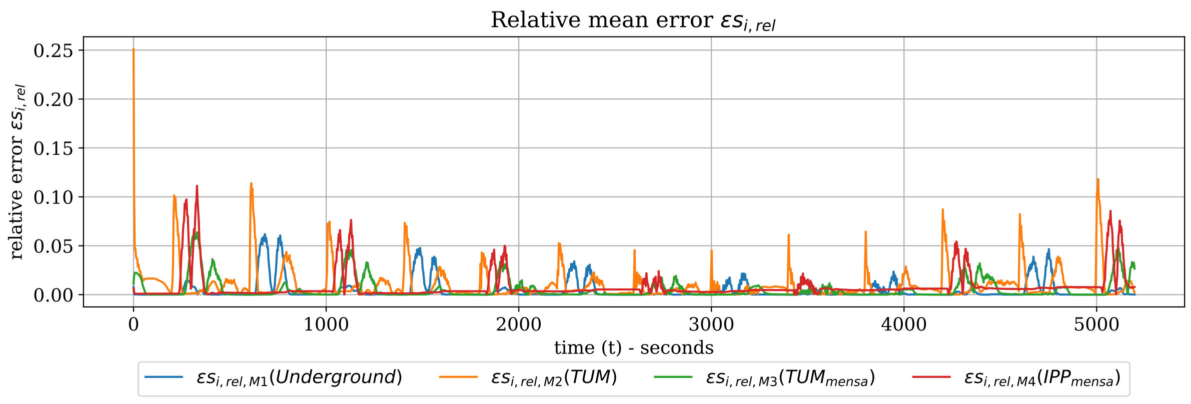


Figure 6.41: Relative mean error $\epsilon s_{i,rel}$ for scenario 3: utilisation of a campus. The values are plotted for all measure zones (Table 6.11) on every time step for $q = 10$ number of collocation points per parameter.

To assess the errors for the whole time series, the mean absolute error $\mu(\epsilon s)$ (Equation (4.20)) and $L^2(\epsilon s)$ error norm (Equation (4.21)), as well as the relative versions $\mu(\epsilon s_{\text{rel}})$ (Equation (4.22)), and $L^2(\epsilon s_{\text{rel}})$ (Equation (4.23)) are calculated and listed in Table 6.17. This shows for all measure zones a relative mean error $\mu(\epsilon s_{\text{rel}})$ of $\leq 0.9\%$ and $\leq 1.9\%$ for $L^2(\epsilon s_{\text{rel}})$, which indicates a good prediction quality for the whole time series.

	$M1(Underground)$	$M2(TUM)$	$M3(TUM_{mensa})$	$M4(IPP_{canteen})$
$\mu(\epsilon s)$	0.785	1.962	0.728	0.244
$L^2(\epsilon s)$	1.934	3.844	1.572	0.616
$\mu(\epsilon s_{\text{rel}})$	0.004	0.009	0.005	0.006
$L^2(\epsilon s_{\text{rel}})$	0.010	0.019	0.010	0.016

Table 6.17: Error statistics $\mu(\epsilon s)$ (Equation (4.20)), $L^2(\epsilon s)$ (Equation (4.21)), $\mu(\epsilon s_{\text{rel}})$ (Equation (4.22)), and, $L^2(\epsilon s_{\text{rel}})$ (Equation (4.23)) for the four values of interest $M1(Underground)$, $M2(TUM)$, $M3(TUM_{mensa})$, and $M4(IPP_{canteen})$.

The errors are also measured for different $q = 2, 3 \ldots, 10$. Figure 6.42(a) contains the absolute mean error $\mu(\epsilon s)$ and Figure 6.42(b) the corresponding $L^2(\epsilon s)$ error norm. The errors are relatively stable for all q in all measure zones. The relative error measurements in Figure 6.43 also confirms the good prediction quality of the COSM.

As discussed in Section 4.7.3, it is not a good idea to exactly trust a specific utilisation value at a specific point in time. Because on a specific time step, high absolute ϵs_i and relative $\epsilon s_{i,rel}$ errors can occur. But as long as small absolute $\mu(\epsilon s)$ and relative $\mu(\epsilon s_{\text{rel}})$ mean errors for the whole time series are measured, the predicted values can be used to obtain trends and to use this for decision-making.

The concept of the COSM in combination with UQ works well and is highly recommended for UQ simulations that have to be performed repeatedly with different probability distributions and parameter values for those distributions, where it matters to obtain the results in a very short time with a certain error tolerance.

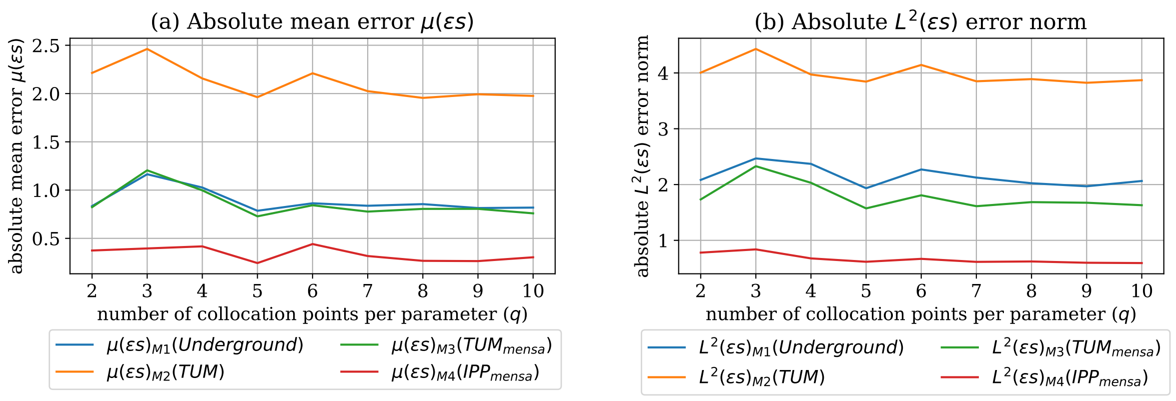


Figure 6.42: Absolute error values for scenario 3: utilisation of a campus. (a) contains the absolute mean error $\mu(\epsilon s)$ values and (b) the corresponding $L^2(\epsilon s)$ error norm for a varying number $q = 2, 3, \ldots, 10$ of collocation points per parameter.

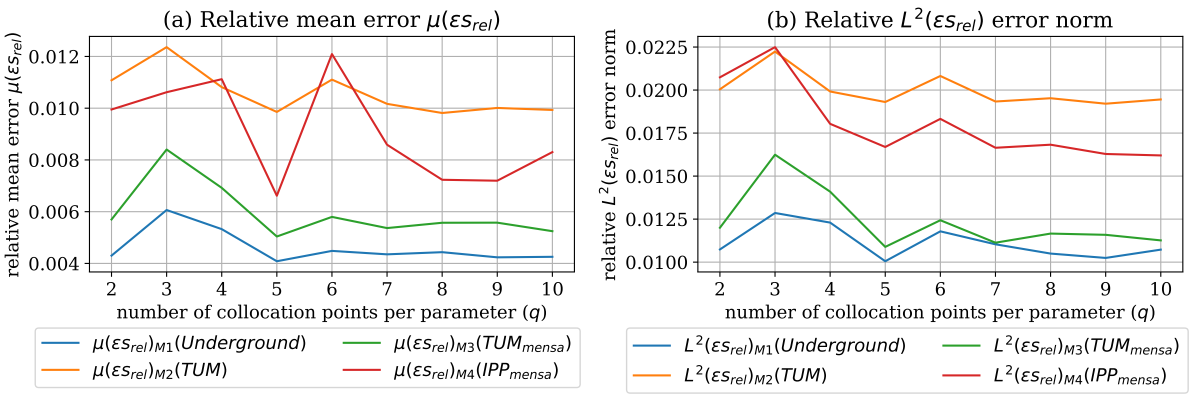


Figure 6.43: Relative error values for scenario 3: utilisation of a campus. (a) contains the relative mean error $\mu(\epsilon s_{\mathrm{rel}})$ values and (b) the corresponding $L^2(\epsilon s_{\mathrm{rel}})$ error norm for a varying number $q = 2, 3, \ldots, 10$ of collocation points per parameter.

6.5.6 Summary

The utilisation of the TUM campus with a daily routine of the pedestrians is simulated under uncertain conditions in this scenario. Vadere is chosen as a microscopic simulator, and the TUM campus is modelled with four buildings (on a coarse level) as a Vadere scenario. Again, the efficient stochastic collocation with the pseudo-spectral approach (Section 2.2.2) is used as the UQ method to quantify the uncertainty.

For this scenario, four uncertain parameters (Table 6.12) are identified. The utilisation of all four modelled buildings is chosen as the OoI. With the UQ simulation, the daily utilisation cycles of the buildings with the variation in the OoI values caused by the uncertain parameters (because of the uncertainty) is now visible. The statistical moments of the QoI allow determining the influence of the uncertainty for all time steps of the simulation. A high standard deviation is observed for each of the four buildings, which means that there is a high influence of the uncertainty in the uncertain input parameters on the scenario.

The global sensitivity analysis (Section 2.3) on top of the UQ simulation allows determining the influence of the different parameters for each time step. In this scenario, the influence of the parameters change continuously with the utilisation cycles of the buildings. An additional finding is that for the different buildings, the importance of the parameters is different.

For a fast quantification of the uncertainty, the closed observables surrogate model (COSM) is used, and its computational efficiency, as well as the introduced error, compared to the use of the Vadere as the model is determined. Once, the closed observables are generated and saved into a file, they can be loaded and used very fast. Speed-ups of two to three orders of magnitude depending on the number q of collocation points per parameter and the used number of computing nodes cn are observed when using the COSM to quantify the uncertainty of the TUM campus utilisation. The relative mean errors of $\leq 0.9\%$ indicate a good prediction quality.

All UQ simulations are based on the UQEF (Chapter 5). Because of the separation of the model and the statistics code, the calculation of the statistical moment for the QoI only has to be implemented once and could be used for the simulations with the COSM and be reused for the comparison runs with Vadere. All production runs are performed on the Linux-Cluster CoolMUC2 using the DWP scheduling strategy implementation of UQEF.

7 Case study: Efficient uncertainty quantification in hydrological modelling

In the field of hydrological modelling [77, 23], water resources of the real-world are investigated with water balance models considering, e.g. surface water, soil water, wetland, ground water, or estuary. The goal is to understand, predict, or to manage the water resources. Especially the prediction of huge water runoffs in flood events or low water situations in rivers are of interest of latest research [95, 96, 195, 36].

The purpose of this case study is to show that the UQEF software framework also supports other research fields and allows to implement new custom models and statistics easily. Beyond that, through the ongoing Hydro-Bits[1] research project, whose goal is to find a modern IT structure for the water management of Bavaria, there is a strong interest to apply uncertainty quantification methods in the hydrological modelling field with a great potential of new insights and improvements.

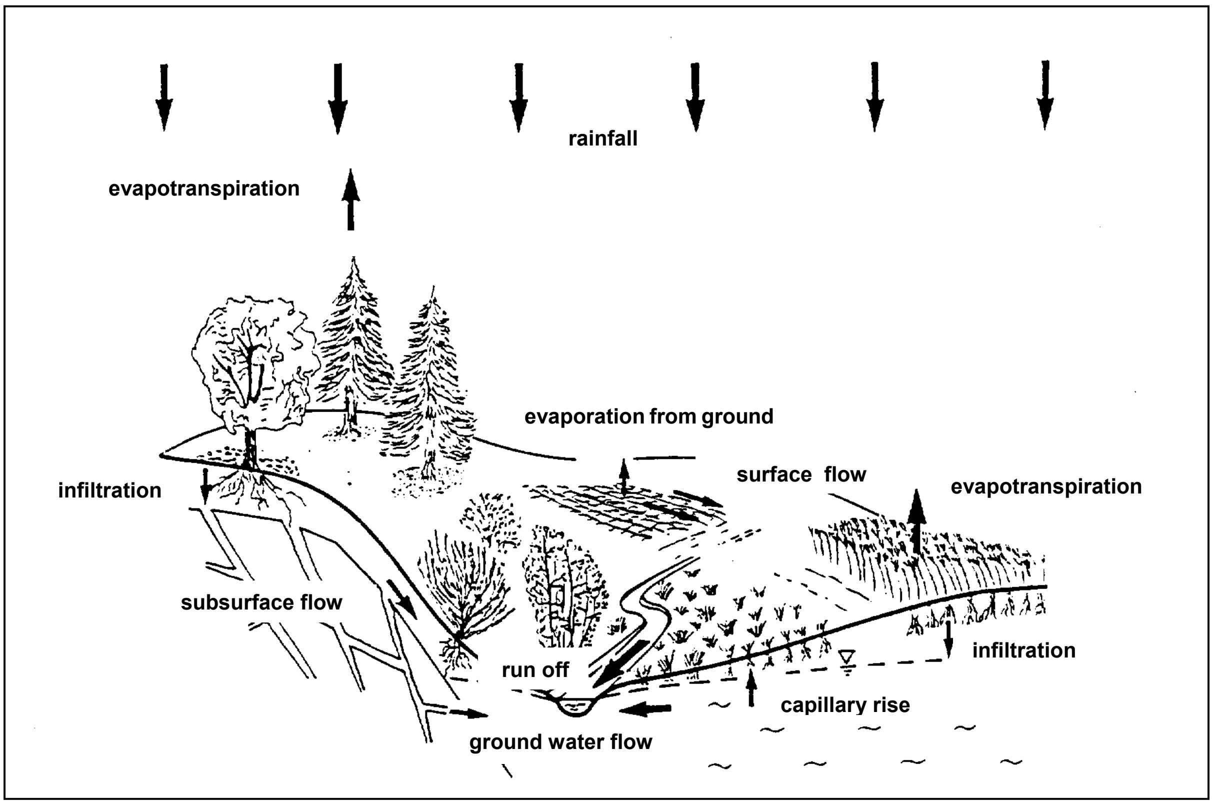


Figure 7.1: Illustration of the possible components of a water balance model (source: [117, 209]).

Figure 7.1 visualises the components [117, 209] of a water balance model. All components like rainfall, evapotranspiration, infiltration, subsurface flow, capillary rise, runoff, and ground

[1] `https://www.lrz.de/forschung/projekte/forschung-e-infra/HydroBITS/`

water flow can be taken into account to quantify spatial and temporal distributions of hydrometeorological data. Such water balance models allow to (a) display the current state of a system, (b) calculate forecasts of the water runoff based on the current system states, and (c) to simulate the water balance with changed system states, e.g. to understand the effect on climate change or the changes of the surface.

Water balance models deal with two types of input data [117]: system data (like elevation, land-use, soil parameters) and hydrometeorological time series (like precipitation, air temperature, air humidity, wind speed, global radiation, water temperature and discharge). The water balance may be determined by several different and complex hydrological processes, like precipitation through rain or snowfall, infiltration into the soil, water storage in the soil, ground water flow, rivers and lakes, as well as the water withdrawal by plants. Figure 7.2 visualises such a soil structure with its associated parameters.

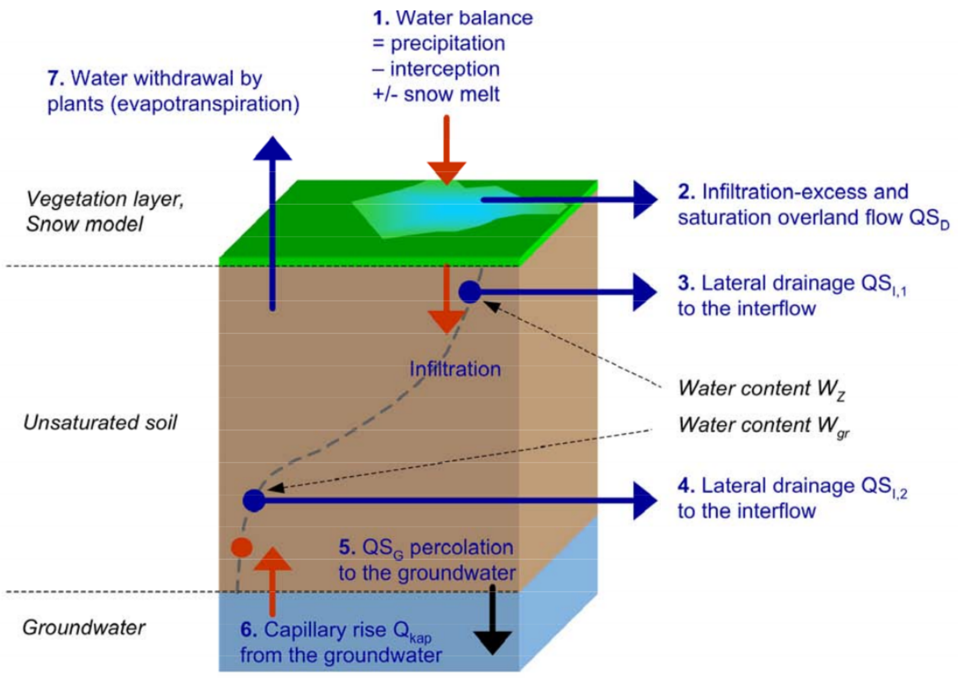

1. Water balance
= precipitation
– interception
+/- snow melt
7. Water withdrawal by plants (evapotranspiration)
Vegetation layer, Snow model
2. Infiltration-excess and saturation overland flow QS_D
3. Lateral drainage $QS_{I,1}$ to the interflow
Infiltration
Water content W_Z
Water content W_{gr}
Unsaturated soil
4. Lateral drainage $QS_{I,2}$ to the interflow
5. QS_G percolation to the groundwater
Groundwater
6. Capillary rise Q_{kap} from the groundwater

Figure 7.2: Illustration of a soil structure with its associated parameters for the use in water balance models (source: [117]).

With such water balance models, a lot of interesting scenarios with many values of interests, and, hence, possible uncertain input parameters exists. The influence of snow melting and subsequent large runoffs is chosen as the scenario, which is further investigated to quantify the uncertainty on some calibration parameters figuring in model's mathematical equations, which are hard to be precisely determined or calibrated and might exhibit a large value of uncertainty.

In this thesis, the water balance model simulator LARSIM is used, which is further described in Section 7.1, followed by Section 7.2, which lists challenges of quantifying the uncertainty in LARSIM. The concrete scenario "large runoffs due to snow melting" with its results is described in Section 7.3.

7.1 LARSIM: a water balance model simulator

The large area runoff simulation model (LARSIM) is a water balance model with many features, which has been widely used since years in academia and by civil institutions. LARSIM is developed by the LARSIM development community, which is a consortium of different civil institutions in Germany. For Bavaria, it is the "Hochwassernachrichtenzentrale (LfU BY)[2]". It is not only used in Germany, but also in France, Luxembourg, Austria, and Switzerland [105].

The most important feature [117] of LARSIM is continuous forecasting of runoff in catchments and river networks, with the special attention to flood and low flow detection, as well as a continuous estimation of different parameters such as water temperature, soil water content and snow cover. Besides the simulations of flood protection planning, it is further used for the prediction of the water balance caused by land-use changes or climate change.

In LARSIM, the sub-area elements of catchments are connected by flood routing elements like rivers. The spatial structure can be based on a spatial grid (rasterisation) or real hydrologic sub-catchments. LARSIM operates on the so-called mesoscale [7] that support cells with an area ranging from 1 km^2 up to several 1,000 km^2.

LARSIM supports two usage modes: operational forecast and simulation (prognosis) mode.

Operational mode The operational mode is used to forecast low-flow, continuous daily or hourly discharge, and flood or water temperature cases. For the operational mode, usually, only precipitation data are used to perform the forecast.

Simulation mode The simulation mode is used for a prognosis of a more detailed water balance model to predict the effect on the water balance caused by climate changes or changes in land usage. Because the simulation mode is more complex, the time series of several input data [117] like global radiation, duration of sunshine, relative air humidity, dew point temperature, air pressure, and water temperatures have to be specified.

Before the actual simulation of the water balance starts, a calibration phase prior to the start of the forecast is performed with measured time series data that are provided by the user. The duration of the calibration phase prior to forecasting can be set by the user and is often in the range of two to three days.[3] During the calibration phase, the model's output is disregarded, however, with the special mechanism of internal state update, it is ensured that the model is warmed-up and the error in the initial state is minimised.

The hydrological model in LARSIM is separated into several components which are illustrated in Figure 7.3. The input for the model is the meteorological data. The first layer is the water storage and transport per land-use in a subbasin (a subbasin is a sub-area of a structural basin which are geological depressions [132]) which includes snow storage, interception and evapotranspiration, as well as soil storage. The next layer is the catchment storage layer that takes lateral water storage and transportation in a subbasin into account. The main parts are surface runoff storage, interflow storage, and groundwater storage. The third layer is based on rivers and lakes with flood routing in a river, in- and outflow, and lake retention with reservoir control. All of this influences the final amount of the simulated runoff from a subbasin. Because of the structural model of the spatial structure, the runoff of one or more grid elements or hydrological sub-catchments can be the inflow of another. This depends on the real composition of the modelled surface with its domes and depressions.

For all that different data that LARSIM deals with, several input and output files and formats exist. Figure 7.4 gives a simplified overview of the most important input and output files.

A brief description of the most important input and output files is given in the following:

`.WHM` The `.WHM` files contain the initial states of the hydrometeorological time series data, including the water storage values for each sub-area.

`tape10` The `tape10` file defines the usage mode of LARSIM and specifies the area and time period, as well as general parameters for the simulation.

[2]`https://www.hnd.bayern.de`

[3]In this work, the calibration phase is performed for 53 hours.

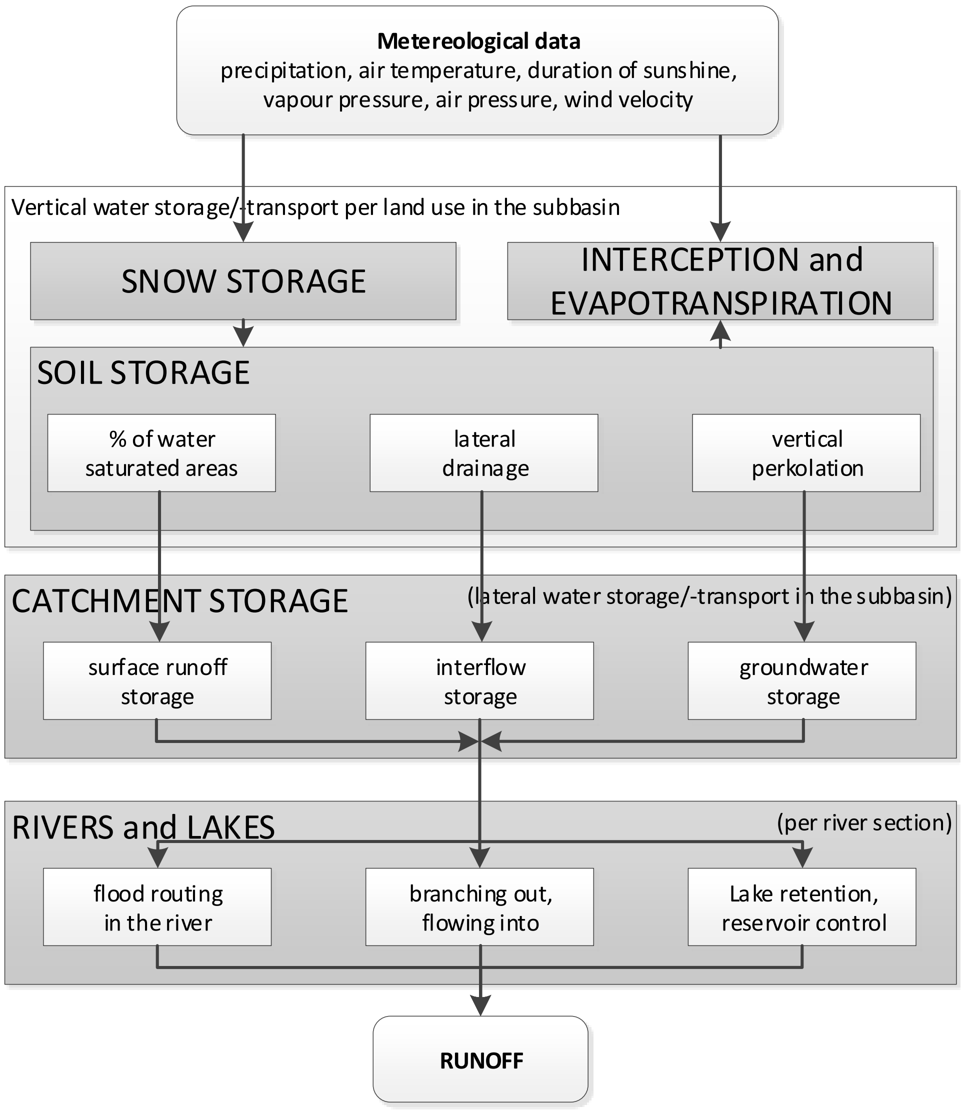

Metereological data
precipitation, air temperature, duration of sunshine, vapour pressure, air pressure, wind velocity
Vertical water storage/-transport per land use in the subbasin
SNOW STORAGE
INTERCEPTION and EVAPOTRANSPIRATION
SOIL STORAGE
% of water saturated areas
lateral drainage
vertical perkolation
CATCHMENT STORAGE
(lateral water storage/-transport in the subbasin)
surface runoff storage
interflow storage
groundwater storage
RIVERS and LAKES
(per river section)
flood routing in the river
branching out, flowing into
Lake retention, reservoir control
RUNOFF

Figure 7.3: Illustration of the layered components in the scheme of the water balance model LARSIM (modified version of: [117]).

`tape12` The `tape12` file contains area-dependent values for variables of the individual sub-areas or elements of the river basin. Additionally, the river basin structure is determined through this information about the gauge stations defined in `tape12`.

`tape35` The `tape35` file contains sub-area specific model parameters, e.g. the calibration values for the retention size of the direct discharge or the interflow.

`lanu.par` The `lanu.par` contains information about land usage, which specifies areas, e.g. as forest or meadow.

`.lila` The `.lila` (*Listenformat für LARSIM*) file format contains meteorological input and output data of station or point related time series. The data for these files are gathered through many gauge stations within an area which continuously measure the real meteorological data.

`.kala` The `.kala` (*Kartenformat für LARSIM*) file format contains the input and output of area or raster data, e.g. weather forecasts or spatially interpolated measurement data, as

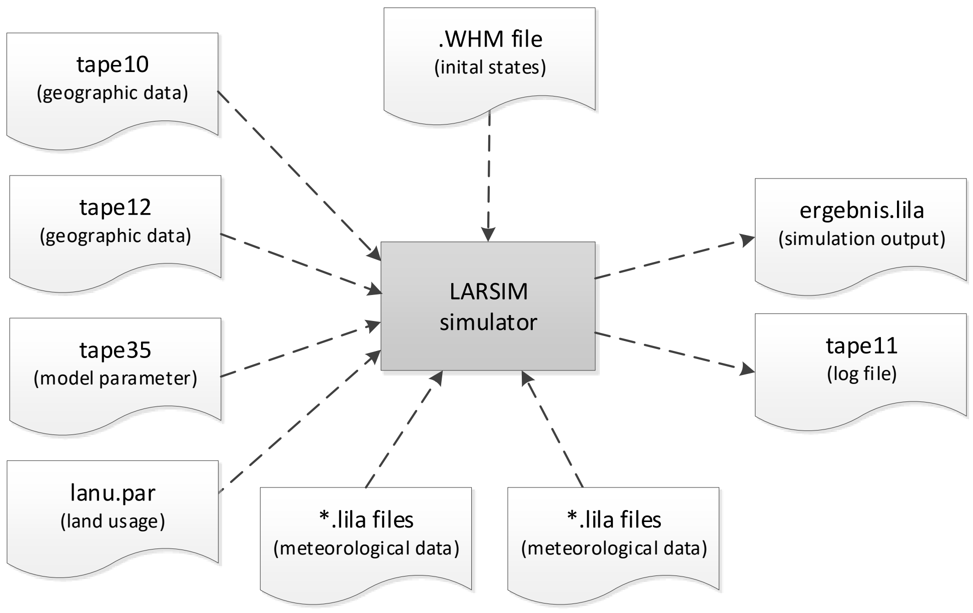


Figure 7.4: Illustration of the most important input and output files of the LARSIM simulator.

well as results of previous simulations for the same area.

`tape11` The `tape11` is the default output file for log messages of LARSIM. It contains general information about the execution of the simulator, the used options and parameters, as well as error messages in the case of errors during the simulation.

`ergebnis.lila` The `ergebnis.lila` contains the resulting time series values including the runoffs of a water balance model simulation.

7.2 Challenges of quantifying uncertainty with LARSIM

The quantification of uncertainties with LARSIM is challenging because of the complexity of the water balance model. The aspects are discussed in different sub-topics: high stochastic dimensionality, unknown probability distribution for parameters, hard to verify the correctness, long runtimes for large or detailed scenarios, and automatic change of internal equations inside LARSIM.

High stochastic dimensionality

LARSIM works with many input parameters and assumptions. Additionally, the calibration phase (for details about the calibration of LARSIM see [60]) before the actual water balance simulation is also based on many assumptions. These parameters and assumptions are based on the season and the investigated catchment regions. Because LARSIM supports large catchment regions, a lot of measurement gauges exist that continuously measure data. Each gauge provides its own set of values for some regionally based parameters, which highly increases the number of parameters. For example, the values for the water content soil storage are provided for each gauge. If this should be considered as independent uncertain parameters, then a probability distribution for each parameter for all gauges have to be used, which strongly leads to the curse of dimensionality.

Additionally, LARSIM works with measured weather data in the simulation mode for simulations of the past. To perform forecasts of the water balance, weather forecast data are used, which are also only predicted, and usually only have a certain accuracy for a few days into the future. This introduces an additional source of uncertainty [135] and makes the prediction of the water balance for LARSIM even harder.

Unknown probability distribution for parameters

For many of the LARSIM parameters, there exist valid minimum and maximum values, which are specified in the various configuration files. But a probability distribution is not given. As described above, many values are somehow calibrated by optimisation algorithms and the calibration phase in the simulation mode prior to the actual starting point of the simulation, which leads to many changes in the values and a probability distribution would, therefore, be a moving target that cannot be defined explicitly. A possible approach to model the uncertain parameters is to apply uniform probability distributions with the specified minimum and maximum values to quantify the uncertainty. Another way is to use automatic measurement-driven distribution generation (see Section 3.4) for measured real data or measured through the calibration phase, to generate the probability distributions dynamically.

Hard to verify the correctness

Unfortunately, it is not possible to obtain data for the water balance through experiments for large areas. However, it is possible to measure—with a certain accuracy—the real hydrometeorological and the runoff data. As already explained, LARSIM calibrates itself with that measured data to produce values that are close to the measured ones. This only works for the past and not for the future, which indicates some uncertainty when performing predictions and highly recommends the use of uncertainty quantification.

Long runtimes for large or detailed scenarios

LARSIM allows simulating the water balance for large areas, many regions, and different time periods. The more regions and the longer the time period that should be taken into account, the longer the runtime of a single LARSIM run or the more LARSIM runs are required.

Automatic change of internal equations inside LARSIM

For some parts, e.g. the surface runoff, LARSIM has several equations (cf. [117, p. 21–22]) from which it selects automatically, based on the values of some input parameters. In UQ, LARSIM is called several times with different parameter values, which can lead to changes in the internally used equation in LARSIM and may introduce non-smooth functions values for some values of interest.

7.3 Scenario: Large runoffs due to snow melting

The understanding and simulation of runoffs is an important part in the field of hydrological modelling [37, 112, 184, 181]. In Bavaria, the spring 2013 was somehow unusual because of high temperatures in the beginning, followed by cold temperatures and a strong snowfall, which led to large runoffs afterwards. After the spring, heavy rainfall led additionally to flooding in June.

The district Regen is chosen as the area under investigation and is visualised in Figure 7.5. Regen is in the south-east of Bavaria and is close to Czechia. The district has an area of about 975 km^2 which is almost fully covered by the Bavarian Forest. The name Regen comes originally from the river with the same name. The river flows in a wide arc from the east to the west through the district. In Zwiesel, a small city on the east within the district, two headwater

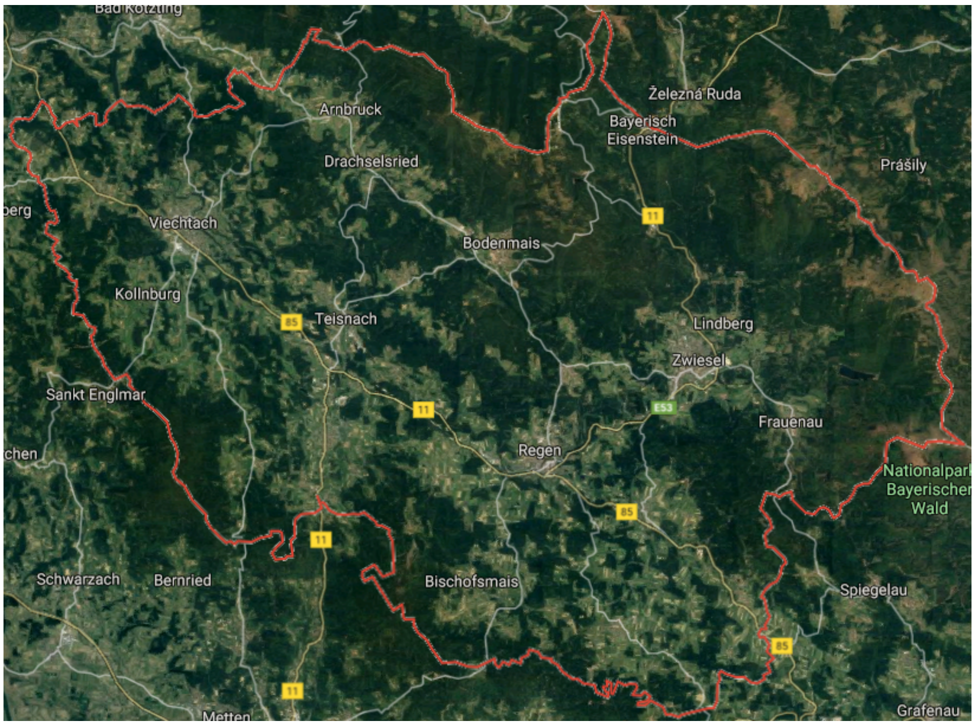


Figure 7.5: Illustration (satellite map) of the district Regen (source: [56]) with the river Regen of the same name.

streams (small and big Regen) joins to the so-called headwater stream Schwarzer Regen into the river Regen.

The goal for this scenario[4] is to investigate the situation with the strong snowfall for the region of Regen and quantify the uncertainty for the large runoffs when the snow melts using LARSIM. Figure 7.6 shows the measured temperatures from 6th of March to 30th of April in 2013 for Zwiesel. The cold period with the low temperatures, which is the 6th coldest March since 1881 according to [51], is quite clearly visible. During this cold period, a closed snow cover could be observed for a wide area in Germany, which did not melt for a long time because of the low temperatures.

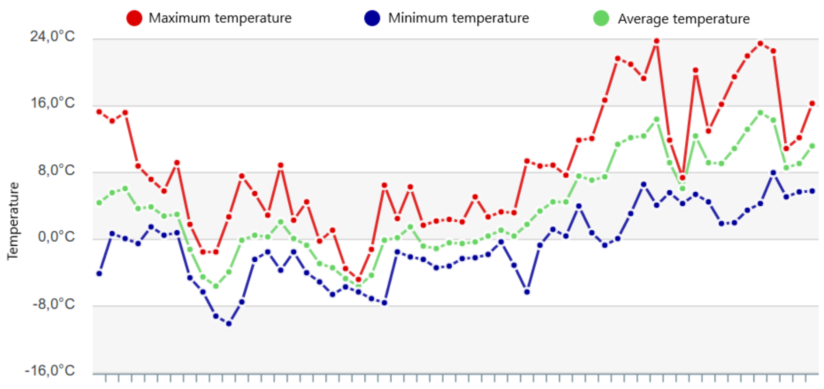


Figure 7.6: Illustration of the measured temperatures from 6th March to 30th April in Zwiesel (district Regen). Every tick on the x axis represents one day. (source: [205])

[4]This scenario is the result of a collaborative work with Ivana Jovanovic. We both supervised a student, Frank Schraufstetter, for his bachelor thesis [162] with the goal of implementing a custom *Model* and *Statistics* class to use the LARSIM simulator within an early version of the UQEF framework.

The regionally based structure for Regen is modelled with rasterisation and a high resolution for the sub-areas of 1 km^2 in size. The simulation of the runoff starts on 1st of March and ends on 30th April 2013. All the meteorological data including the measured time series weather data is provided by the Bavarian Environment Agency[5].

The snow melting and water runoff behaviour are further explained in Section 7.3.1. In Section 7.3.2, the UQ simulation setup is documented, and Section 7.3.3 presents the numerical results for this scenario. Finally, the scenario is summarised in Section 7.3.4.

7.3.1 Snow melting and water runoff behaviour

A flood is caused by precipitation and soil-water contribution and means that there is an additional overflow of water from usually dry surfaces. The reason is an overflow of water from a river, a lake, or an ocean which further flows over their boundaries. Snow may also contribute significantly to the runoff because the snow cover is accumulated and stored precipitation on the surface area. Due to rainfall or snowmelt (caused by warm temperatures), the snowpack melts and is, therefore, an additional source for the runoff which can lead to floods.

LARSIM supports the snowmelt simulation with the complete energy balance based on the extended method of Knauf [117], which allows to model different conditions in the temperatures in more detail compared to the method of Knauf [93], with the goal of more realistic runoff results.

In this thesis, four parameters (*Tmit_Sr*, *Tspann_Sr*, *Abso*, and *SRet*) out of several that significantly influence the snow accumulation and snowmelt behaviour are further taken into account. These parameters are chosen, because they may contain a certain uncertainty which influences the runoff and could possibly lead to floods.

Tmit_Sr Does it snow, rain, or is it a mixture of snow/rain? *Tmit_Sr* defines the average limit for the temperature in [°C] to decide whether precipitation falls as rain or snow.

Tspann_Sr With the *Tspann_Sr* parameter, a range around *Tmit_Sr* is defined that controls the snow/rain mixture. Equation (7.1) shows the dependency of *Tmit_Sr* and *Tspann_Sr*, which uses T_L, the measured air temperature in [°C] 2 m above ground, to decide if the precipitation falls as rain, as snow/rain mixture, or as snow.

$$precipitation = \begin{cases} rain, & \text{if } T_L \leq Tmit\_Sr - Tspann\_Sr/2 \\ snow/rain, & \text{if } Tmit\_Sr - Tspann\_Sr/2 < T_L < Tmit\_Sr + Tspann\_Sr/2 \\ snow, & \text{if } T_L \geq Tmit\_Sr + Tspann\_Sr/2 \end{cases} \tag{7.1}$$

Abso The *Abso* parameter controls the absorption rate of the short wave net radiation for the accumulated snowpack on the surface. The higher the absorption rate, the higher the potential snowmelt rate and, hence, the possible runoff.

SRet A snowpack is able to store liquid water until a critical value. The maximum retention rate or liquid water in [%] *SRet* is used to calculate the snowmelt intensity, which leads to additional runoff.

These four parameters are sub-area dependent and therefore specified in the `tape35` input file, which contains different values for each sub-area. The district Regen is modelled with 29 sub-areas, which means there are $29 \times 4 = 116$ values for the four parameters *Tmit_Sr*, *Tspann_Sr*, *Abso*, and *SRet*. In Table 7.1, the supported minimum and maximum values in LARSIM are listed.

[5] `https://www.lfu.bayern.de`

Parameter	Description	Minimum	Maximum
Tmit_Sr	Average limit temperature of snow/rain mixture [°C]	-3	2
Tspann_Sr	Temperature range around *Tmit_Sr* [°C]	2	8
Abso	Absorption rate of the short wave net radiation	0.02	0.25
SRet	Maximum retention rate of liquid water [%]	5.0	47.0

Table 7.1: List of relevant parameters for scenario: large runoffs due to snow melting. For each parameter, the supported minimum and maximum values within LARSIM are listed.

7.3.2 UQ simulation setup

To study the behaviour of the water balance model for the district Regen under uncertainty, an appropriate UQ simulation has been performed. For that, the four parameters of Table 7.1 are assumed to be uncertain. Because no detailed information about their possible probability distributions with their specific parameters are available, a different approach is pursued.

If for each of the 29 sub-areas within the district Regen an individual probability distribution would be applied, then this would end up with $29 \times 4 = 116$ different probability distributions. With the usage of stochastic collocation with the pseudo-spectral approach (Section 2.2.2) that comprises a full tensor, this would result in $Q = q^{116}$ collocation points. To avoid this curse of dimensionality, only one probability distribution is used for each of the four parameters, which leads to $Q = q^4$. For each individual black-box simulation run of LARSIM in the propagation phase, the pre-calibrated values within `tape35` are taken, and the values from the generated nodes are added to each sub-area. This allows to consider the individual pre-calibrated values for each sub-area–but to vary the values as it is needed to quantify the uncertainty.

The distributions with their values are listed in Table 7.2. The values for the distributions are chosen in such a way that the full range of the allowed minimum and maximum values for each parameter is used.

Parameter	Description	Distribution
Tmit_Sr	Average limit temperature of snow/rain mixture [°C]	$U(-2, 4)$
Tspann_Sr	Temperature range around *Tmit_Sr* [°C]	$U(-0.08, 0.1)$
Abso	Absorption rate of the short wave net radiation	$U(-25, 17)$
SRet	Maximum retention rate of liquid water [%]	$U(-2, 3)$

Table 7.2: List of uncertain parameters for scenario: large runoffs due to snow melting. The parameters are uniformly distributed between the specified minimum and maximum values.

The output of interest is the runoff (denoted by $\mathcal{Q}$) inside the defined time range from 1st March to 30th April 2013. LARSIM is used in the hourly simulation mode, whereas the QoI and the resulting plots contain the runoff in a daily aggregated form. Because often historical measurements are also documented with daily runoffs, which allows an easier comparison. The runoff of the station called `MARI` is taken. `MARI` is located directly at the river Regen in Marienthal, which is nearby Regensburg. The road distance between the station `MARI` and the city of Regen is about 96.4 km, and from the station `MARI` to the city of Regensburg about 33.8 km. It is assumed that the uncertainty on the four snow parameters in Table 7.2 lead to a large varying runoff during the additional snowmelt in the distant station `MARI` and later also in the city of Regensburg because the river Regen flows from the higher located district Regen via Marienthal to the lower located city of Regensburg.

For this scenario, the stochastic collocation with the pseudo-spectral approach is used. To have a decent coverage of the parameter range, $q = 10$ number of collocation points for each uncertain parameter are used, which leads to $Q = 10^4 = 10{,}000$ collocation points z_i. The highest order of the orthogonal polynomials $\Phi_j(\boldsymbol{\zeta})$ of the gPCE (Equation (2.7)) is set to $P = 6$.

The source code for the UQ simulation with LARSIM is comprised in mainly two classes, which are derived from the interfaces provided by UQEF. Figure 7.7 illustrates the additional classes: The *SnowMeltingModel* class is derived from the *Model* interface and contains the source code for configuring and calling LARSIM. Within the *SnowMeltingStatistics* class, which is derived from the *Statistics* interface, all the code for the certification phase to calculate the statistical moments for the quantity of interest—the runoff—is implemented.

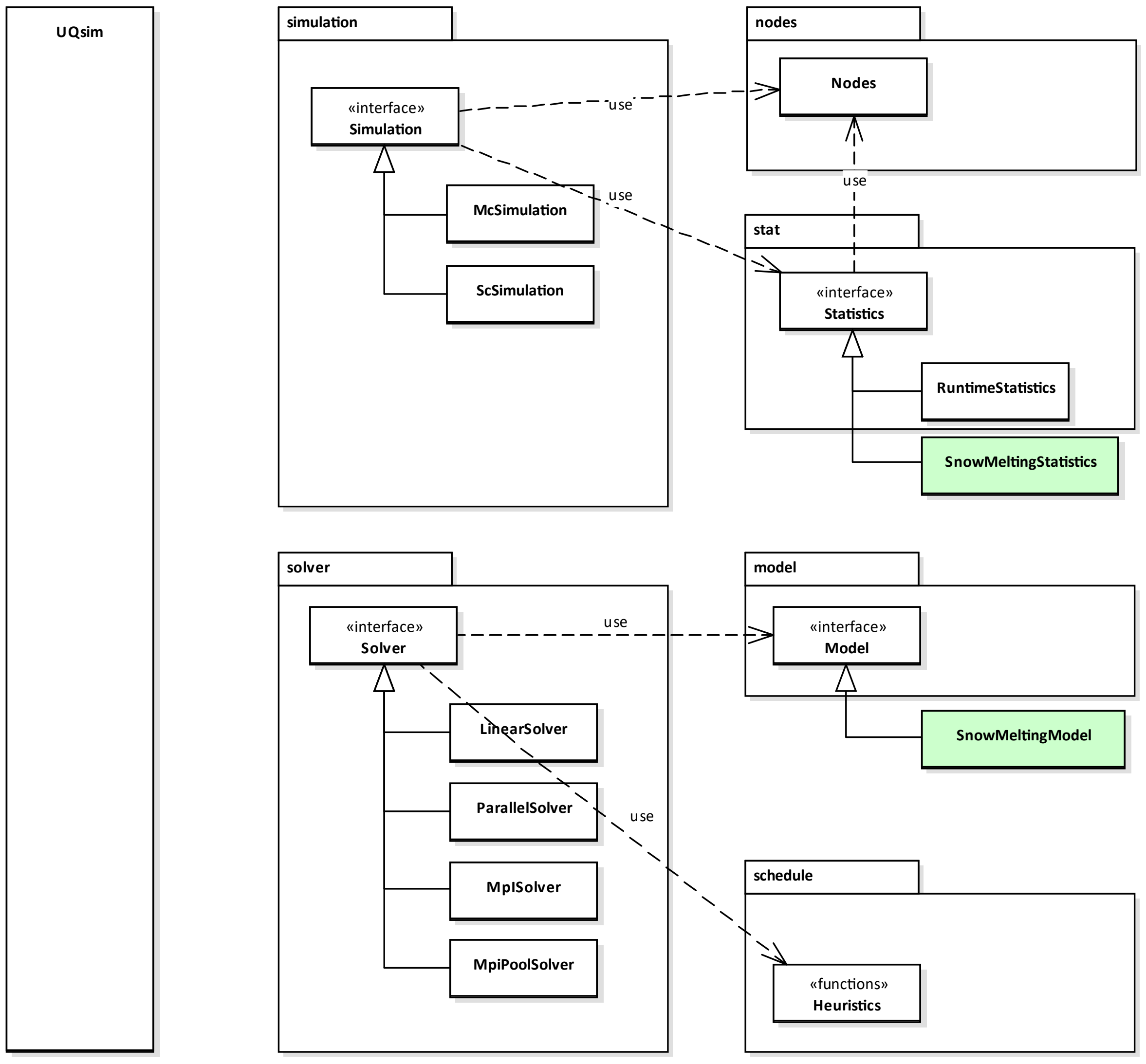


Figure 7.7: Illustration of derived classes for scenario: large runoffs due to snow melting. The light green rectangles indicate the implemented custom classes for this scenario. See Section 5.1 for a detailed description of the overall UQEF architecture.

The UQ simulation for this scenario is executed on the Linux-Cluster CoolMUC2 of the Leibniz Supercomputing Centre [115]. The SLURM job is configured to use eight computing nodes. On each computing node, 20 out of the 28 CPU cores with 55 GB of RAM are used. With the DWP scheduling strategy (Section 4.4.4) and the *MpiPoolSolver* one LARSIM run is started in parallel on each CPU core. Not all CPU cores are used, to not exceed the memory limits. It could be observed that using all 28 CPU cores led to process crashes caused by out-of-memory errors. With the usage of 20 out of the 28 CPU cores per computing node, the simulations are much more stable.

Each LARSIM process needs its own directory environment with all data. LARSIM works with predefined paths and uses the working directory for its operation. Hence, for each LARSIM black-box run one corresponding directory with all required input and output data has to be prepared. It is not possible to set the same working directory for two or more parallel running LARSIM processes because LARSIM also changes some input files, and competing processes would override, e.g. the `ergebnis.lila` of each other. UQEF supports this by generating a unique identifier for each black-box model run, which can be used to generate appropriate directory names and to identify each black-box model run.

Remark: If for the propagation the *ParallelSolver* or the *MpiSolver* of UQEF is used, which uses threading to utilise all CPU cores, one additional detail has to be considered when calling the LARSIM simulator: It is very common to change the working directory with `os.chdir(new_working_dir)` within Python. However, in common operating systems such as Windows or Linux, there is only one working directory per process—and not per thread! Because a process is a managing container for all its threads, it is a mistake to change the working directory within each thread with `os.chdir()` before starting the LARSIM simulator with, e.g. `subprocess.run()`. This would change the working directory for all threads and led to undefined behaviour because the parallel running LARSIM simulators may then use the same working directory and read/overwrite each other's files. Instead, the working directory change has to be done in the newly created subprocess just before the LARSIM simulator is started.

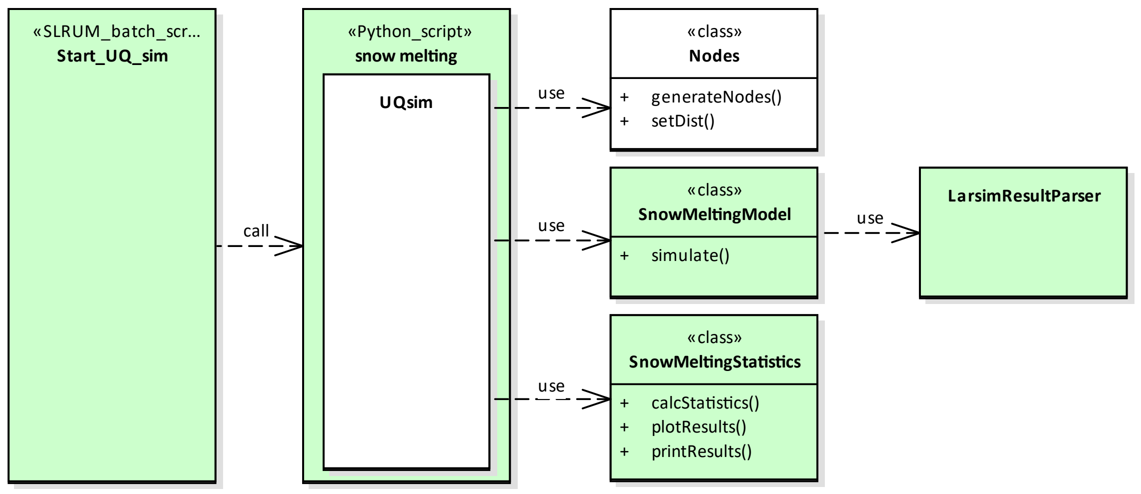


Figure 7.8: Illustration of the simulation program architecture for scenario: large runoffs due to snow melting. The light green rectangles indicate the custom implementation parts for this scenario. The white rectangles indicates the UQEF classes that are instantiated and used directly, but are not changed.

In Figure 7.8, the resulting program architecture is illustrated. Similar to the scenarios in Chapter 6, a SLURM batch script submits a batch job that starts the *snow melting* Python script. Inside the *snow melting* Python script, an instance called `uqsim` of the *UQsim* class is created and the scenario-specific classes *SnowMeltingModel* and *SnowMeltingStatistics* are registered at the `uqsim` object. The *SnowMeltingModel* class uses additionally the *LarsimResultParser* helper class because the various files of LARSIM do use a custom file format, which requires some complex manual parsing of the input and output files because no persistence layers are available for Python. The probability distributions with their specific values for the four uncertain snow parameters are read from a JSON configuration file using the config file parametrisation feature of UQEF.

7.3.3 Numerical results

This section contains the numerical results for the quantification of the uncertainties for the runoff $\mathcal{Q}$ with all the four snow parameters in Table 7.2 as uncertain. All statistical moments for the runoff $\mathcal{Q}$, which is the quantity of interest, are generated for the station MARI (Marienthal).

Figure 7.9 shows the mean runoff $\mu(\mathcal{Q})$ at the station MARI for each day in the investigated period from 1st March to 30th April 2013. Between this time period, a mean runoff $\mu(\mathcal{Q})$ of up to about 62 $[m^3/s]$ is observed, and between 18th April to 22nd April 2013 the runoff reaches up to 80 $[m^3/s]$. Due to the snowmelt, the runoff may nearly double its value within 2 days. For Marienthal, the "Gewässerkundlicher Dienst Bayern"[6] reports an average runoff over all days since 1st November 1900 of about 37.3 $[m^3/s]$, an average runoff of the highest values of about 304 $[m^3/s]$, and a highest measured runoff of about 720 $[m^3/s]$. The observed heavy snowmelt from 1st March to 30th April 2013 does, under this investigated uncertainty, not reach the average runoff of the highest values of about 304 $[m^3/s]$. But if snowmelt combined with heavy precipitation occurs, then the snowmelt may significantly contribute to the overall runoff, and its uncertainty plays an additional role. The 80% interval by the $p_{10}(\mathcal{Q})$ and $p_{90}(\mathcal{Q})$ percentiles indicates that there is indeed some uncertainty in the four snow parameters (Table 7.1). The red line in the plot is the real measured runoff $\mathcal{Q}$. On many days, the real measured runoff $\mathcal{Q}$ is not inside the 80% interval that the percentiles span around the mean runoff $\mu(\mathcal{Q})$. This indicates that LARSIM can, with the specified parameter values, not exactly simulate the real runoff behaviour under such extreme weather conditions. This shows that there are other, additional sources of error, such as uncertainty in the provided data or in the assumed model structure, which obviously introduce some bias in model prediction. Focusing on and quantifying these uncertainties can be follow-up work.

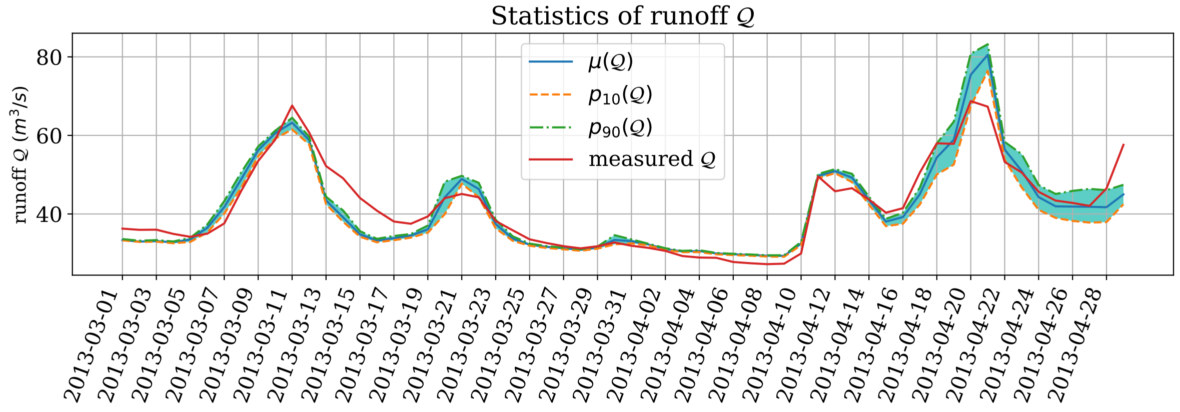


Figure 7.9: QoI results of the mean runoff $\mu(\mathcal{Q})$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting. The light blue area around the mean runoff $\mu(\mathcal{Q})$ is the 80% interval of the $p_{10}(\mathcal{Q})$ and $p_{90}(\mathcal{Q})$ percentiles. The red line shows the real measured runoff $\mathcal{Q}$ data at station MARI.

The standard deviation $\sigma(\mathcal{Q})$ of the runoff $\mathcal{Q}$ in Figure 7.10 shows that especially in the runoff peaks, the standard deviation is high. This shows the importance of using techniques such as

[6]`https://www.gkd.bayern.de/de/fluesse/abfluss/passau/marienthal-15207507`

uncertainty quantification because only one LARSIM run would not give an idea about the entire range that has to be expected.

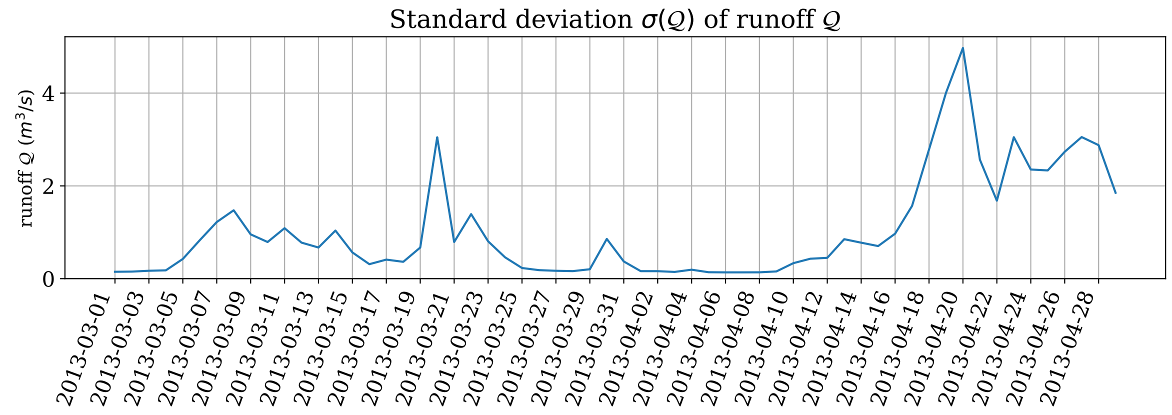


Figure 7.10: Standard deviation $\sigma(\mathcal{Q})$ of the runoff $\mathcal{Q}$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting.

To evaluate the quality of the UQ simulation, the absolute error $\epsilon(\mathcal{Q})$ (Equation (7.2)) between the mean runoff $\mu(\mathcal{Q})$ and the measured runoff $\mathcal{Q}$ is calculated for every day in the investigated period. The corresponding relative error $\epsilon_{rel}(\mathcal{Q})$ is defined in Equation (7.3).

$$\epsilon(\mathcal{Q}) := |\mu(\mathcal{Q}) - \text{measured } \mathcal{Q}| \qquad (7.2) \qquad\qquad \epsilon_{rel}(\mathcal{Q}) := \frac{\epsilon(\mathcal{Q})}{\text{measured } \mathcal{Q}} \qquad (7.3)$$

Figure 7.11 shows the calculated absolute error $\epsilon(\mathcal{Q})$. On most days, $\epsilon(\mathcal{Q})$ is below 5 $[m^3/s]$. On the days with the large changes in the runoff $\mathcal{Q}$, the error reaches up to about 10 $[m^3/s]$ on 15th March 2013, and up to about $[m^3/s]$ on 22nd April 2013. The values in Figure 7.12 shows relative errors $\epsilon_{rel}(\mathcal{Q})$ of ≤ 0.1 for most days. On the days with the large changes in the runoff $\mathcal{Q}$, the relative errors $\epsilon_{rel}(\mathcal{Q})$ reaches up to about 0.21.

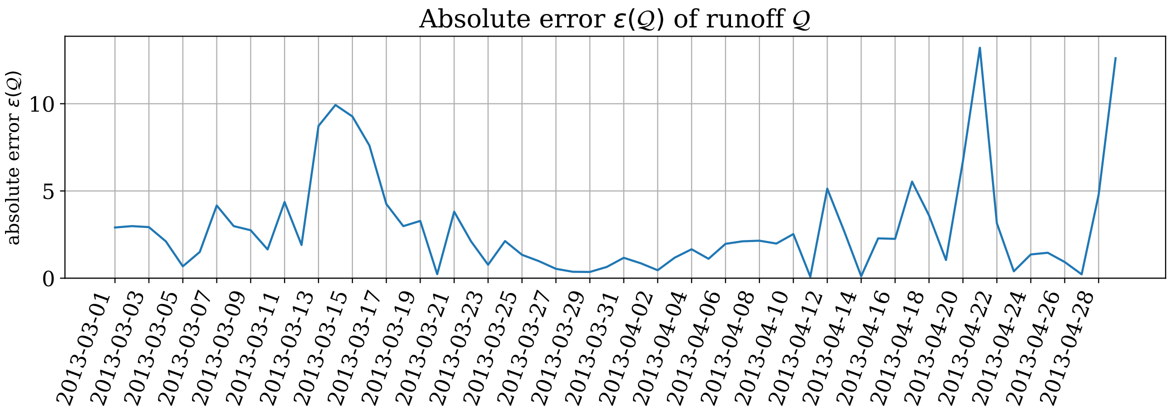


Figure 7.11: Absolute error $\epsilon(\mathcal{Q})$ of the runoff $\mathcal{Q}$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting.

Which of the four snow parameters of Table 7.1 have the most influence on the uncertainty? Two answer this, the sensitivity indices have been calculated. Figure 7.13 shows the first-order sensitivity indices. The most contribution comes from *SRet* and *Tmit_Sr*, which do change its influence over the days. On the days with precipitation[7], *Tmit_Sr* becomes more important.

[7]The precipitation data are not plotted here, but can be directly downloaded from `https://www.gkd.bayern.de/de/meteo/niederschlag/passau/teisnach-5002` provided by the “Gewässerkundlicher Dienst Bayern” for the station Teisnach which is located in the district Regen.

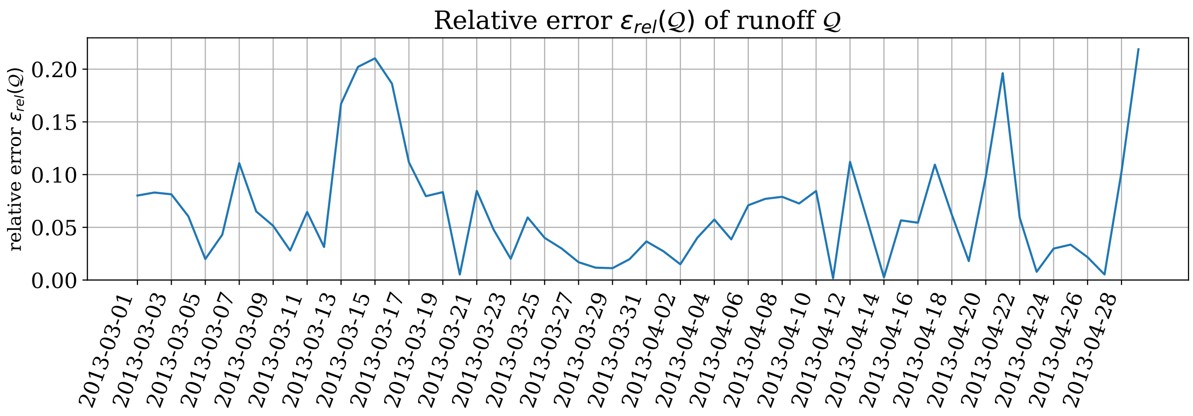


Figure 7.12: Relative error $\epsilon_{rel}(\mathcal{Q})$ of the runoff $\mathcal{Q}$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting.

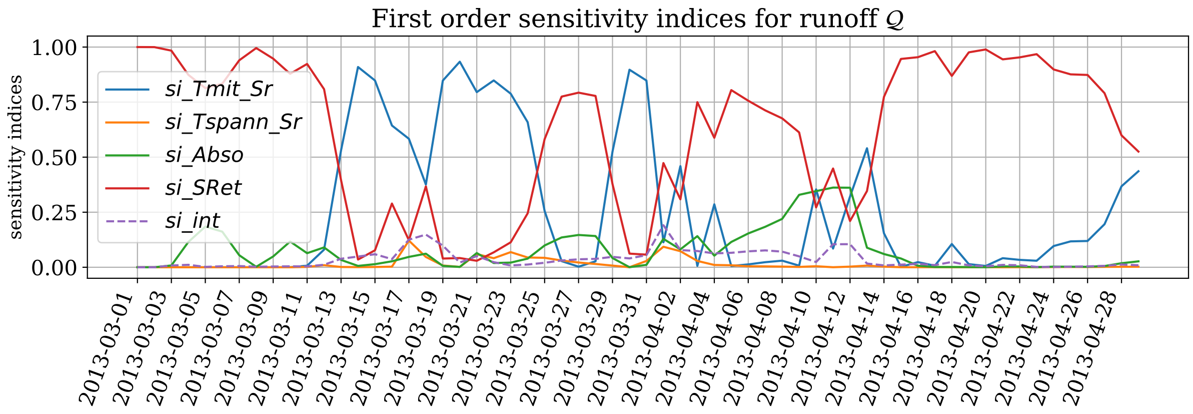


Figure 7.13: First-order sensitivity indices of the four snow parameters (Table 7.1) on the runoff $\mathcal{Q}$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting.

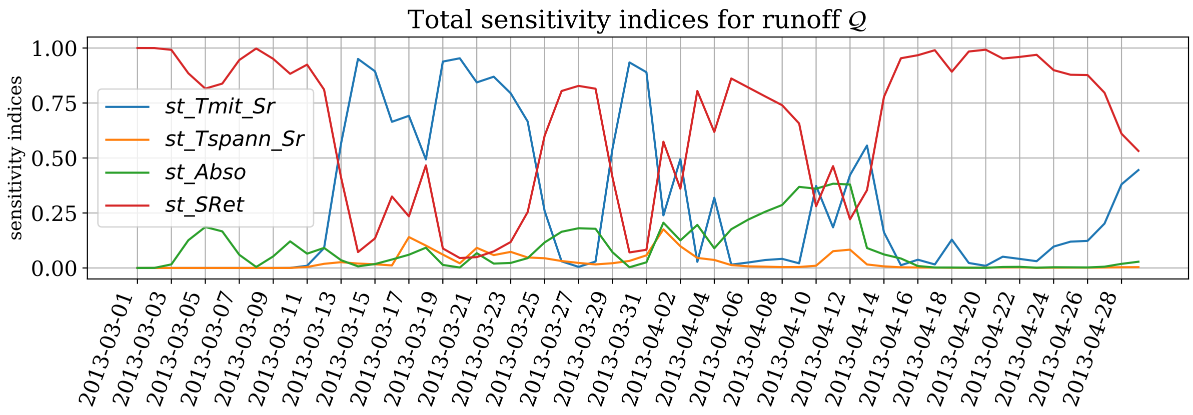


Figure 7.14: Total-order sensitivity indices of the four snow parameters (Table 7.1) on the runoff $\mathcal{Q}$ in station MARI (Marienthal) for scenario: large runoffs due to snow melting.

Because then it has a direct influence in the form of the precipitation of snow, rain, or a snow/rain mixture. If the stored liquid water exceeds its critical value, the snow releases water and *SRet* has the most influence. The other two parameters do also contribute to the uncertainty but not that much. In Figure 7.13, si_{int} shows that there is only little interaction between the parameters. Hence, the total-order sensitivity indices in Figure 7.14 have similar values than the first-order sensitivity indices and do not contain additional information.

7.3.4 Summary

This scenario shows that uncertainty in the snow parameters does have a significant influence on the overall runoff $\mathcal{Q}$. Hence, uncertainty quantification should be the default when performing predictions of the runoff $\mathcal{Q}$, especially because with LARSIM it is not possible to predict the extreme cases exactly, which increases the importance of knowing the intervals instead of a single LARSIM run. Finally, with the sensitivity indices, the understanding of the behaviour of the water balance model can be greatly improved.

Future work may also take parts of the measured weather data or weather forecast data (especially the amount of precipitation and the temperatures) as uncertain parameters to investigate whether the prediction intervals for such extreme cases can be improved with LARSIM.

With UQEF and the dynamic work package scheduling (Section 4.4.4) using the *MpiPoolSolver*, the propagation time T_{Prop} was about 6h using four computing nodes on the CoolMUC2. The development of the sources for this UQ simulation codes and the actual execution of the simulation on the CoolMUC2 greatly benefited from the UQEF.

8 Conclusion

This chapter concludes this thesis: Section 8.1 contains a summary of the contributions with the obtained results, and Section 8.2 gives an outlook of future work.

8.1 Summary

The contributions in this work have been driven by the idea of quantifying the uncertainty of large-scale simulation scenarios efficiently: combining fast prototyping, state-of-the-art UQ methods, and a reusable software solution for a fast and efficient UQ simulation. For that, four different efficiency aspects have been defined: (a) fast development time to solution, (b) short simulation time, (c) high utilisation of computational resources, and (d) fast and easy creation and interpretation of the UQ simulation results.

The four main contributions are (1) employ modern, state-of-the-art UQ methods to quantify the uncertainty of large-scale simulation scenarios (considering (b) and (d)), (2) investigate and improve the propagation phase of a UQ simulation with the ability to predict the simulation time and reducing the idling (considering (b) and (c)), (3) support (real-time) decision-making problems under uncertainty with the COSM (considering (b)), and finally, (4) the developed UQEF software framework (considering (a), (b), (c), and (d)).

The first main contribution contains the use of stochastic collocation with the pseudo-spectral approach in the fields of pedestrian dynamics and hydrological modelling. For the pedestrian dynamics field using the Vadere simulator, three scenarios of the *evacuation of a train station*, *evacuation of a building with separated families*, and the *utilisation of a campus* have been investigated in Chapter 6. It could be shown, that in all scenarios, the use of UQ helped to understand the scenarios better and to obtain additional insights, which is very useful for the developers of the models itself as well as for the users, e.g. for the planning of buildings considering the evacuation, or the utilisation of a large campus. In the field of hydrological modelling (Chapter 7) with the use of the LARSIM simulator, the scenario of *large runoffs due to snow melting* has been considered. It turned out that the uncertainty in the investigated parameters have a significant influence in the runoff and should be the default when predicting extreme weather cases. For the investigated scenarios, it could be shown with different visualisation techniques which parameters contributes most and when to the output of interest, which is very valuable. Additionally, for each field, the specific challenges are identified and considered in the scenarios. The identified important development aspects of UQ simulations with concrete hints to efficiently model uncertain parameters and interpret the UQ simulation results are described in Chapter 3.

The second main contribution (Chapter 4) concerning the propagation phase of a UQ simulation showed that using three standard scheduling strategies (static work packages, static work packages with thread pool on node level, and dynamic work packages) idling can occur easily if the uncertain input parameters influence the model's runtime. With the novel idea of using the runtime of a model as a quantify of interest, it turned out that it is possible to create a runtime predictor that can predict the runtime of a single black-box model run under the given uncertainty and additionally for the whole propagation phase of a UQ simulation. Using the runtime prediction information in UQ simulations to reorder the individual black-box model runs with the goal of minimising the idling, the three standard scheduling strategies could be optimised. Speed-ups of 1.7–2.3 from the worst standard scheduling strategy, the static work packages, compared to the best-optimised scheduling, the optimised dynamic work packages, could be achieved for the *evacuation of a building with separated families* scenario in the pedestrian dynamics field.

The third main contribution tackles the need for real-time decision-making. For that, the concept of the closed observables surrogate model (COSM) has been developed in Chapter 4. The COSM is used for the scenario *utilisation of a campus* in the pedestrian dynamics field in Chapter 6. Using the COSM, speed-ups of up to three orders of magnitude could be achieved when quantifying the uncertainty compared to the use of the original model (the Vadere simulator), by only introducing absolute mean errors of 1.962 and relative mean errors of 0.009 for the whole simulation time. One additional advantage in the use of the COSM compared to other surrogate model techniques is that if a different probability distribution for an input parameter should be used, the surrogate model can be reused, without the need to sample the original model again.

The fourth main contribution is the implemented UQEF software framework in Chapter 5 that considers the defined efficiency aspects. The UQEF is used to implement the UQ simulation source codes in all scenarios of Chapters 6 and 7. Because UQEF is based on Chaospy, it is still a lightweight and flexible solution that allows rapid prototyping with all the existing Chaospy functionality. Because in UQEF, the runtime prediction mechanism and the three scheduling strategies with their optimised versions are implemented, it closes the missing part of Chaospy and supports an efficient propagation phase that automatically scales from a development PC to a compute cluster such as an HPC system.

With the proposed solutions in this thesis, it is now possible for developers using the implemented UQEF software framework, to easily quantify the uncertainty of large-scale simulation scenarios even more efficiently.

8.2 Outlook

Currently, the UQEF software framework is actively used in the hydrological modelling field in conjunction with the LARSIM water balance simulator to quantify the uncertainty of various scenarios. For that, an additional UQ method, the Saltelli method [156], is implemented and added to the UQEF by Ivana Jovanovic.

The presented runtime prediction and optimised scheduling mechanisms are applicable to Bayesian inverse problems as well. Like in forward UQ, the main task in Bayesian inversion is to repeatedly evaluate the forward model for a (potentially large) number of input samples, i.e., in an outer-loop fashion. When the forward numerical model is solved using iterative algorithms or when adaptive algorithms are used, the runtime of different simulations might differ significantly. Thus, standard scheduling strategies might be ineffective, which is undesirable, especially when the cost of one forward simulation is large. For example, in [41], dimension-adaptive sparse grids are combined with multilevel decompositions to find non-intrusive surrogates of the posterior density in Bayesian inverse problems. The algorithm in [41] is sequential, in the sense that current level computations depend on computations from the previous level, and cannot be performed ahead of time. Moreover, since multilevel decompositions are employed, at each level, a different grid resolution is applied to discretise the forward model. Therefore, the runtime between different levels might differ significantly, which is detrimental when standard scheduling strategies are used. In addition, when runtimes between different simulations at the same level vary significantly (for example, when the forward model is characterised by an algorithm that has to reach an imposed tolerance for convergence), standard scheduling leads to ineffective use of resources. In the latter situation it might be more appropriate to use the runtime of each simulation as a cost penalty in adaptive algorithms to ensure that if two or more grid points have similar benefits, the one with the smallest computational cost is added to the grid.

Another idea is to integrate the spatially adaptive sparse grid combination approach (sparseSpACE) [129], developed by Michael Obersteiner, into UQEF. Instead of using the Gauss quadrature in the stochastic collocation with the pseudo-spectral approach to define the collocation points z_i and the corresponding weights w_i to actually calculate the coefficients c_j, the spatially adaptive sparse grid combination approach can be used to find proper collocation points z_i and weights w_i to perform the integration and to compute the coefficients c_j. This would allow tackling higher-dimensional problems with good accuracy. Because of the adaptivity of this approach, the grid points are iteratively refined, which gives a further possibility to take advantage of the runtime prediction mechanism and the optimised scheduling strategies which could minimise the idling for each iteration step.

The idea of using an additional, synthetic value of interest of a model as the QoI is not limited to the runtime. Also, other performance measures are possible, like memory usage or CPU usage, which offers additional use cases in predicting the required resources and improving the scheduling.

The runtime prediction mechanism in combination with a scheduling strategy may also be used in an optimisation problem to find a proper setup for a computing environment: For example, the maximum propagation runtime could be defined as the optimisation criterion, and the optimisation problem could then try to find a proper scheduling strategy with the required resources (number of computing nodes) to fulfil the given runtime requirement.

A Appendix

A.1 Visualisation of common distributions and their quadrature

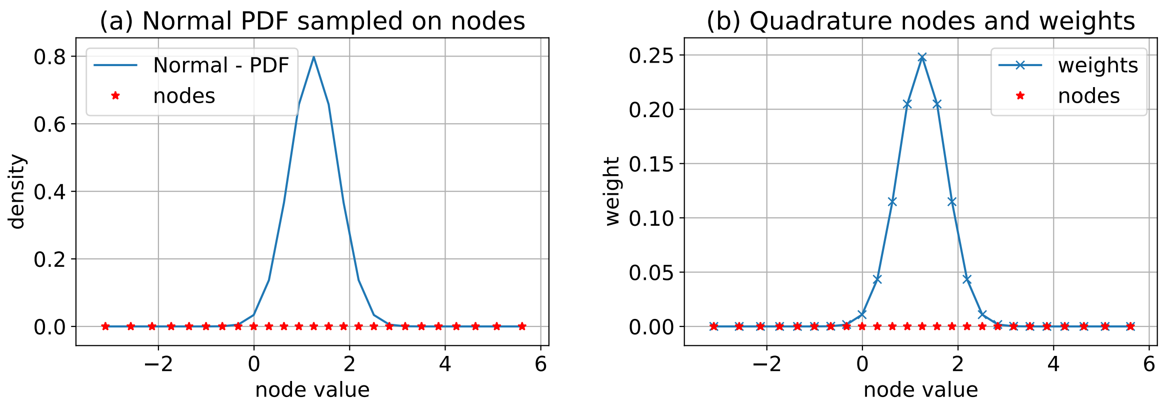


Figure A.1: Visualisation of the probability density function of a $\zeta \sim \mathcal{N}(0, 1)$ distributed random variable ζ in (a) and the corresponding generated collocation points z_i with its weights w_i in (b). For this visualisation, $q = 25$ collocation points for ζ are used.

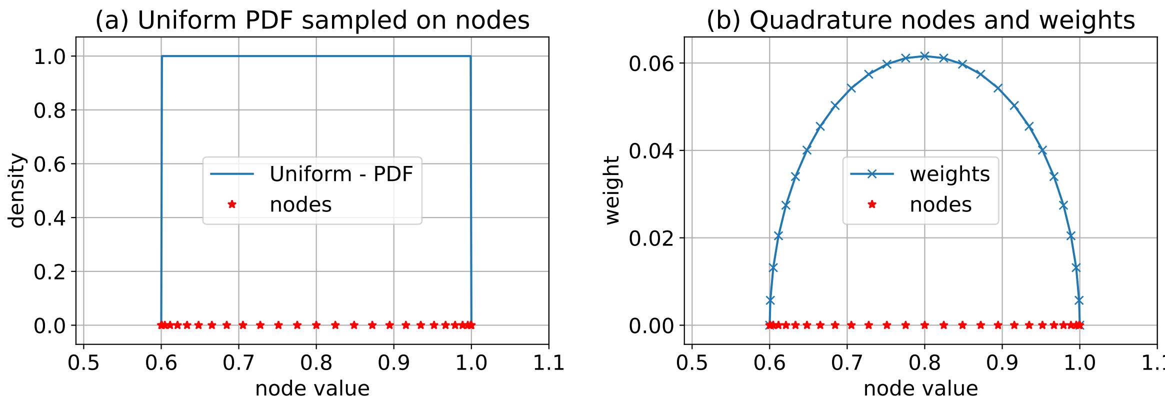


Figure A.2: Visualisation of the probability density function of a $\zeta \sim \mathcal{U}(0.6, 1.0)$ distributed random variable ζ in (a) and the corresponding generated collocation points z_i with its weights w_i in (b). For this visualisation, $q = 25$ collocation points for ζ are used.

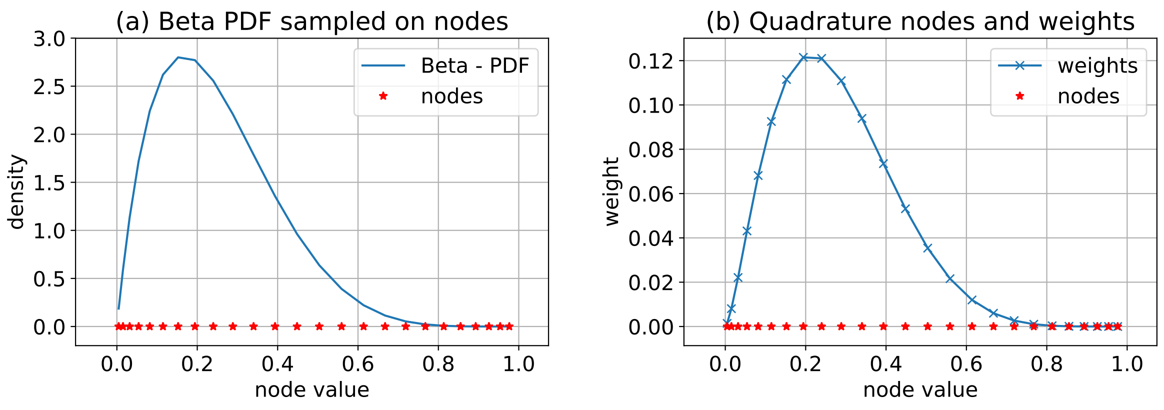


Figure A.3: Visualisation of the probability density function of a $\zeta \sim \mathcal{B}(alpha = 2, beta = 2, lower = 0, upper = 1.0)$ distributed random variable ζ in (a) and the corresponding generated collocation points z_i with its weights w_i in (b). For this visualisation, $q = 25$ collocation points for ζ are used.

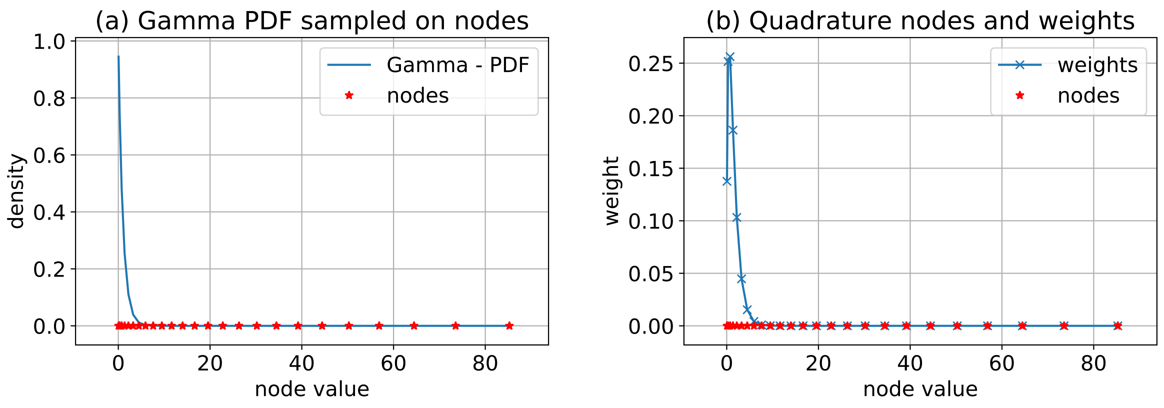


Figure A.4: Visualisation of the probability density function of a $\zeta \sim \mathcal{G}(sahpe = 1, scale = 1, shift = 0)$ distributed random variable ζ in (a) and the corresponding generated collocation points z_i with its weights w_i in (b). For this visualisation, $q = 25$ collocation points for ζ are used.

A.2 Chaospy introduction examples

A.2.1 Monte Carlo

```python
import chaospy as cp # import chaospy library
import numpy as np   # import numpy library

# model definition: simple example model with 2 parameters
def model(p1, p2):
    return p1+p2  # returns the value of interest (VoI)

# 1. assimilation: determine distributions
p1_dist = cp.Uniform(lower=-5, upper=1)  # creates a uniform distribution
p2_dist = cp.Normal(mu=0, sigma=1)       # creates a normal (Gaussian) distribution

dists = cp.J(p1_dist, p2_dist)           # joins the dists to a multivariate dist.

nodes = dists.sample(1000)               # generate samples nodes according to the
                                         # multivariate distributions

# 2. propagation
solves = [model(p1, p2) for p1, p2 in nodes.T]

# 3. certification: determine statistics of QoI
E       = np.mean(solves)                # determine E (mean)
Var     = np.var(solves)                 # determine Var (variance)
StdDev  = np.std(solves)                 # determine StdDev (standard deviation)
P5, P95 = np.percentile(solves, [5, 95]) # determine 5th and 95th percentile

# print results
print("E: {}".format(E))
print("Var: {}".format(Var))
print("StdDev: {}".format(StdDev))
print("P5: {}".format(P5))
print("P95: {}".format(P95))
```

Listing A.1: Full introduction example of the Monte Carlo method with Chaospy, regarding to Section 2.4.2.

A.2.2 Point collocation

```python
import chaospy as cp # import chaospy library
import numpy as np   # import numpy library

# model definition: simple example model with 2 parameters
def model(p1, p2):
    return p1+p2  # returns the value of interest (VoI)

# 1. assimilation: determine distributions
p1_dist = cp.Uniform(lower=-5, upper=1)  # creates a uniform distribution
p2_dist = cp.Normal(mu=0, sigma=1)       # creates a normal (Gaussian) distribution

dists = cp.J(p1_dist, p2_dist)           # joins the dists to a multivariate dists.

nodes = dists.sample(1000)               # generate samples nodes according to the
                                         # multivariate distributions

# 2. propagation
solves = [model(p1, p2) for p1, p2 in nodes.T]

# 3. certification: determine statistics of QoI
OP = cp.orth_ttr(2, dists) # creates orthogonal polynomials according the multivariate
                           # distribution using three terms recursion

gPCE = cp.fit_regression(OP, nodes, solves) # generate gPCE

E       = cp.E(gPCE, dists)                 # determine E (mean)
Var     = cp.Var(gPCE, dists)               # determine Var (variance)
StdDev  = cp.Std(gPCE, dists)               # determine StdDev (standard deviation)
P5, P95 = cp.Perc(gPCE, [5, 95], dists)     # determine 5th and 95th percentile

# print results
print("E: {}".format(E))
print("Var: {}".format(Var))
print("StdDev: {}".format(StdDev))
print("P5: {}".format(P5))
print("P95: {}".format(P95))
```

Listing A.2: Full introduction example of the point collocation method with Chaospy, regarding to Section 2.4.2.

A.2.3 Stochastic collocation with the pseudo-spectral approach

```python
import chaospy as cp # import chaospy library
import numpy as np   # import numpy library

# model definition: simple example model with 2 parameters
def model(p1, p2):
    return p1+p2  # returns the value of interest (VoI)

# 1. assimilation: determine distributions
p1_dist = cp.Uniform(lower=-5, upper=1)  # creates a uniform distribution
p2_dist = cp.Normal(mu=0, sigma=1)       # creates a normal (Gaussian) distribution

dists = cp.J(p1_dist, p2_dist)           # joins the dists to a multivariate dists.

# creates 2+1 nodes per parameter and corresponding weights according to
# the multivariate distribution and the Gaussian integration scheme
nodes, weights = cp.generate_quadrature(2, dists, rule="G")

# 2. propagation
solves = [model(p1, p2) for p1, p2 in nodes.T]

# 3. certification: determine statistics of QoI
OP = cp.orth_ttr(2, dists) # creates orthogonal polynomials according the multivariate
                           # distribution using three terms recursion

gPCE = cp.fit_quadrature(OP, nodes, weights, solves) # generate gPCE

E       = cp.E(gPCE, dists)             # determine E (mean)
Var     = cp.Var(gPCE, dists)           # determine Var (variance)
StdDev  = cp.Std(gPCE, dists)           # determine StdDev (standard deviation)
P5, P95 = cp.Perc(gPCE, [5, 95], dists) # determine 5th and 95th percentile

# print results
print("E: {}".format(E))
print("Var: {}".format(Var))
print("StdDev: {}".format(StdDev))
print("P5: {}".format(P5))
print("P95: {}".format(P95))
```

Listing A.3: Full introduction example of the stochastic collocation with the pseudo-spectral approach with Chaospy, regarding to Section 2.4.2.

A.3 Propagation runtimes of scheduling strategies

A.3.1 Propagation runtimes of academic example 1

		Scheduling strategies (time in seconds)					
cn	q	SWP	SWPT	DWP	SWP$_{OPT}$	SWPT$_{OPT}$	DWP$_{OPT}$
2	4	48.5	30.1	25.3	27.2	24.3	24.2
	5	78.8	48.9	33.4	32.0	42.7	36.5
	6	103.3	79.4	55.7	55.5	65.8	57.0
	7	187.3	129.4	91.4	86.1	94.0	89.6
	8	268.3	191.9	133.2	128.2	136.5	131.7
	9	376.9	270.2	189.0	181.7	187.1	187.4
	10	485.9	359.8	257.9	248.6	258.3	256.7
	11	648.6	479.5	341.4	330.9	332.4	340.5
	12	839.2	622.3	440.6	430.0	444.0	439.9
3	4	24.2	24.3	25.7	24.2	24.3	24.2
	5	50.5	42.7	26.7	25.2	29.5	25.2
	6	77.5	65.9	41.3	38.7	50.4	39.9
	7	131.3	106.9	63.4	57.9	69.2	59.3
	8	185.5	154.5	90.8	85.7	97.8	87.8
	9	240.1	209.9	123.3	122.1	125.7	124.0
	10	321.7	278.2	174.1	166.4	177.0	171.4
	11	430.6	367.0	228.8	221.5	231.3	227.5
	12	566.6	484.8	291.0	287.2	302.0	290.7
4	4	24.3	24.4	25.3	24.5	24.3	24.3
	5	50.5	42.7	28.4	25.3	25.3	25.2
	6	51.9	50.4	31.1	28.1	50.3	30.3
	7	105.1	91.1	48.0	43.7	51.6	47.5
	8	132.7	122.1	67.5	64.7	78.0	68.3
	9	187.0	170.3	95.4	91.6	104.7	93.6
	10	241.5	223.2	131.6	125.6	131.7	128.5
	11	323.1	294.6	167.7	166.2	180.3	170.6
	12	432.0	393.3	220.1	215.6	232.8	218.8
5	4	24.3	24.3	27.7	24.5	24.3	24.3
	5	25.3	25.3	28.3	25.3	25.3	25.2
	6	51.9	50.4	27.0	25.9	28.8	25.8
	7	78.9	76.6	37.8	36.6	51.5	41.3
	8	106.3	101.2	54.6	52.0	67.8	54.3
	9	160.4	150.6	78.2	74.6	93.0	74.7
	10	214.7	202.8	107.4	100.8	113.4	102.2
	11	269.3	251.8	138.7	133.3	150.2	135.8
	12	351.0	328.0	176.8	172.8	187.0	175.4

Table A.2: Full list of propagation runtimes T_{Prop} (in seconds) for the academic example function $\hat{f}_{ex1}$ (Equation (3.1)) regarding to Section 4.6.4 with the six scheduling strategies and their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.3.2 Propagation runtimes of academic example 2

		Scheduling strategies (time in seconds)					
cn	q	SWP	SWPT	DWP	SWP$_{OPT}$	SWPT$_{OPT}$	DWP$_{OPT}$
2	4	33.2	20.3	20.7	18.5	17.9	17.9
	5	59.4	38.7	35.5	27.6	30.3	31.2
	6	68.7	49.6	52.3	44.4	46.6	46.2
	7	121.8	79.9	78.9	72.4	75.7	73.4
	8	168.8	107.6	110.6	106.8	105.2	104.8
	9	252.5	160.4	155.8	155.7	151.3	150.5
	10	290.5	202.1	207.9	212.6	198.9	203.3
	11	409.6	270.2	271.7	279.0	265.9	266.6
	12	536.2	344.1	352.5	373.2	345.4	346.4
3	4	17.9	17.9	19.2	18.5	17.9	17.9
	5	42.0	28.7	26.9	22.9	25.6	22.8
	6	54.7	40.1	37.8	29.7	36.6	31.7
	7	92.7	58.7	55.8	49.5	51.5	49.6
	8	116.8	83.4	77.2	70.9	77.4	71.6
	9	156.6	118.1	106.8	102.3	102.7	101.1
	10	206.6	154.1	141.6	143.5	140.4	135.1
	11	284.7	202.2	183.6	190.7	177.6	177.5
	12	364.3	263.6	237.1	251.6	229.8	232.3
4	4	17.9	18.1	20.3	18.6	17.9	17.9
	5	42.0	27.5	23.2	22.9	21.9	21.9
	6	38.3	30.2	30.6	23.1	29.9	27.8
	7	80.1	49.0	44.7	37.1	43.2	37.3
	8	91.4	61.7	60.6	54.2	56.5	55.1
	9	129.1	94.0	82.4	81.5	82.7	75.4
	10	152.9	112.1	108.7	105.7	102.1	102.2
	11	206.3	150.0	142.4	141.9	138.3	133.7
	12	283.7	189.1	180.3	186.0	179.1	173.2
5	4	17.9	18.1	19.8	18.6	17.9	18.1
	5	21.9	21.9	23.5	22.9	21.9	21.9
	6	38.3	28.7	27.7	21.4	27.3	22.3
	7	62.7	43.4	37.6	30.4	37.4	32.3
	8	76.3	55.4	50.8	45.1	50.6	44.1
	9	116.5	77.2	68.7	64.1	68.0	61.9
	10	140.8	101.4	88.4	84.6	88.3	81.9
	11	172.7	127.3	114.1	116.4	113.3	107.0
	12	215.6	171.5	146.1	150.4	143.6	138.9

Table A.4: Full list of propagation runtimes T_{Prop} (in seconds) for the academic example function $\hat{f}_{ex2}$ (Equation (3.2)) regarding to Section 4.6.4 with the six scheduling strategies and their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.3.3 Propagation runtimes of pedestrian dynamics scenario 2

		Scheduling strategies (time in seconds)					
cn	*q*	SWP	SWPT	DWP	SWP$_{OPT}$	SWPT$_{OPT}$	DWP$_{OPT}$
2	4	1857.0	940.3	930.2	975.9	644.8	939.3
	5	2898.2	1716.6	1171.0	1137.9	1234.6	1280.8
	6	3884.4	3024.3	1861.2	1892.3	2121.4	1894.1
	7	6697.6	4764.2	2987.7	2919.8	3137.8	2980.3
	8	9393.9	7186.0	4450.6	4463.0	4793.6	4444.1
	9	13020.3	9781.2	6096.4	7112.2	6557.5	6154.9
	10	17054.7	13992.0	8602.8	8810.7	9230.7	8578.4
	11	22880.5	17999.7	11295.9	12277.3	12082.7	11268.2
	12	29060.0	23686.3	14650.3	17002.0	15720.8	14619.2
3	4	971.7	701.0	890.9	987.9	513.2	922.7
	5	1974.6	1351.9	1023.6	1050.6	974.5	1034.6
	6	3004.1	2314.3	1306.3	1267.0	1419.0	1419.2
	7	4626.1	3635.5	1984.4	2052.8	2196.8	1995.2
	8	6545.8	5229.4	2957.2	3030.1	3290.3	3018.2
	9	8485.7	7618.2	4046.4	4757.5	4379.0	4036.3
	10	11384.9	10120.7	5723.0	6196.8	6252.1	5723.8
	11	15244.0	13532.6	7508.7	8338.4	8193.9	7504.0
	12	19844.6	17672.9	9729.5	13332.5	10559.9	9702.4
4	4	965.8	597.6	967.0	1009.0	472.5	936.6
	5	1995.3	1132.3	997.9	1064.3	614.8	1011.0
	6	1987.9	1713.6	996.2	1055.1	1123.7	1136.7
	7	3796.8	2860.6	1583.3	1525.0	1610.3	1662.5
	8	4796.7	4116.4	2260.9	2267.5	2438.2	2226.6
	9	6612.2	5809.0	3039.2	3543.7	3364.5	3041.3
	10	8864.8	7764.1	4300.4	4653.3	5127.2	4328.7
	11	11532.7	10468.7	5636.5	6690.6	6256.4	5664.5
	12	15597.1	13932.5	7286.9	11444.6	7998.9	7269.6
5	4	932.8	503.3	960.5	961.4	452.6	962.7
	5	976.1	891.1	1025.1	1041.0	585.1	1001.4
	6	1989.4	1486.6	1014.8	1093.5	909.8	1009.4
	7	2984.2	2268.7	1424.9	1285.7	1400.6	1507.8
	8	3904.6	3273.7	1829.0	1860.9	2048.8	1857.6
	9	5753.6	4809.6	2452.0	2991.7	2676.4	2484.8
	10	7692.8	6631.6	3441.0	3673.3	3775.2	3480.5
	11	9609.6	8496.6	4496.2	5532.2	4948.3	4534.9
	12	12535.7	11121.6	5839.7	10246.6	6391.9	5814.9

Table A.6: Full list of propagation runtimes T_{Prop} (in seconds) for the pedestrian evacuation scenario 2 in Section 6.4 with the six scheduling strategies and their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.4 Speed-ups of scheduling strategies

A.4.1 Speed-ups of academic example 1

		Speed-ups				
cn	q	SWP to SWP$_{OPT}$	SWP to SWPT$_{OPT}$	SWP to DWP$_{OPT}$	SWPT to SWPT$_{OPT}$	DWP to DWP$_{OPT}$
2	4	1.7	1.9	2.0	1.2	1.0
	5	2.4	1.8	2.1	1.1	0.9
	6	1.8	1.5	1.8	1.2	0.9
	7	2.1	1.9	2.0	1.3	1.0
	8	2.0	1.9	2.0	1.4	1.0
	9	2.0	2.0	2.0	1.4	1.0
	10	1.9	1.8	1.8	1.3	1.0
	11	1.9	1.9	1.9	1.4	1.0
	12	1.9	1.8	1.9	1.4	1.0
3	4	1.0	1.0	1.0	1.0	1.0
	5	2.0	1.7	2.0	1.4	1.0
	6	2.0	1.5	1.9	1.3	1.0
	7	2.2	1.9	2.2	1.5	1.0
	8	2.1	1.9	2.1	1.5	1.0
	9	1.9	1.9	1.9	1.6	0.9
	10	1.9	1.8	1.8	1.5	1.0
	11	1.9	1.8	1.8	1.5	1.0
	12	1.9	1.8	1.9	1.6	1.0
4	4	0.9	1.0	1.0	1.0	1.0
	5	2.0	1.9	2.0	1.6	1.1
	6	1.8	1.0	1.7	1.0	1.0
	7	2.4	2.0	2.2	1.7	1.0
	8	2.0	1.7	1.9	1.5	0.9
	9	2.0	1.7	2.0	1.6	1.0
	10	1.9	1.8	1.8	1.6	1.0
	11	1.9	1.7	1.8	1.6	0.9
	12	2.0	1.8	1.9	1.6	1.0
5	4	0.9	1.0	1.0	1.0	1.1
	5	1.0	1.0	1.0	1.0	1.1
	6	2.0	1.8	2.0	1.7	1.0
	7	2.1	1.5	1.9	1.4	0.9
	8	2.0	1.5	1.9	1.4	1.0
	9	2.1	1.7	2.1	1.6	1.0
	10	2.1	1.8	2.1	1.7	1.0
	11	2.0	1.7	1.9	1.6	1.0
	12	2.0	1.8	2.0	1.7	1.0

Table A.8: Full list of speed-ups for the academic example function $\hat{f}_{ex1}$ (Equation (3.1)) regarding to Section 4.6.4 with their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.4.2 Speed-ups runtimes of academic example 2

		Speed-ups				
cn	*q*	SWP to SWP$_{OPT}$	SWP to SWPT$_{OPT}$	SWP to DWP$_{OPT}$	SWPT to SWPT$_{OPT}$	DWP to DWP$_{OPT}$
2	4	1.7	1.8	1.8	1.1	1.1
	5	2.1	1.9	1.9	1.2	1.1
	6	1.5	1.4	1.4	1.0	1.1
	7	1.6	1.6	1.6	1.0	1.0
	8	1.5	1.6	1.6	1.0	1.0
	9	1.6	1.6	1.6	1.0	1.0
	10	1.3	1.4	1.4	1.0	1.0
	11	1.4	1.5	1.5	1.0	1.0
	12	1.4	1.5	1.5	1.0	1.0
3	4	0.9	1.0	1.0	1.0	1.0
	5	1.8	1.6	1.8	1.1	1.1
	6	1.8	1.5	1.7	1.1	1.1
	7	1.8	1.8	1.8	1.1	1.1
	8	1.6	1.5	1.6	1.0	1.0
	9	1.5	1.5	1.5	1.1	1.0
	10	1.4	1.4	1.5	1.1	1.0
	11	1.4	1.6	1.6	1.1	1.0
	12	1.4	1.5	1.5	1.1	1.0
4	4	0.9	1.0	1.0	1.0	1.1
	5	1.8	1.9	1.9	1.2	1.0
	6	1.6	1.2	1.3	1.0	1.1
	7	2.1	1.8	2.1	1.1	1.2
	8	1.6	1.6	1.6	1.0	1.1
	9	1.5	1.5	1.7	1.1	1.0
	10	1.4	1.5	1.4	1.1	1.0
	11	1.4	1.4	1.5	1.0	1.0
	12	1.5	1.5	1.6	1.0	1.0
5	4	0.9	1.0	0.9	1.0	1.1
	5	0.9	1.0	1.0	1.0	1.0
	6	1.7	1.4	1.7	1.0	1.2
	7	2.0	1.6	1.9	1.1	1.1
	8	1.6	1.5	1.7	1.1	1.1
	9	1.8	1.7	1.8	1.1	1.1
	10	1.6	1.5	1.7	1.1	1.0
	11	1.4	1.5	1.6	1.1	1.0
	12	1.4	1.5	1.5	1.1	1.0

Table A.10: Full list of speed-ups for the academic example function $\hat{f}_{ex2}$ (Equation (3.2)) regarding to Section 4.6.4 with their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.4.3 Speed-ups runtimes of pedestrian dynamics scenario 2

		Speed-ups				
cn	*q*	SWP to SWP$_{OPT}$	SWP to SWPT$_{OPT}$	SWP to DWP$_{OPT}$	SWPT to SWPT$_{OPT}$	DWP to DWP$_{OPT}$
2	4	1.9	2.8	1.9	1.4	0.9
	5	2.5	2.3	2.2	1.3	0.9
	6	2.0	1.8	2.0	1.4	0.9
	7	2.2	2.1	2.2	1.5	1.0
	8	2.1	1.9	2.1	1.5	1.0
	9	1.8	1.9	2.1	1.4	0.9
	10	1.9	1.8	1.9	1.5	1.0
	11	1.8	1.8	2.0	1.4	1.0
	12	1.7	1.8	1.9	1.5	1.0
3	4	0.9	1.8	1.0	1.3	0.9
	5	1.8	2.0	1.9	1.3	0.9
	6	2.3	2.1	2.1	1.6	0.9
	7	2.2	2.1	2.3	1.6	0.9
	8	2.1	1.9	2.1	1.5	0.9
	9	1.7	1.9	2.1	1.7	1.0
	10	1.8	1.8	1.9	1.6	1.0
	11	1.8	1.8	2.0	1.6	1.0
	12	1.4	1.8	2.0	1.6	1.0
4	4	0.9	2.0	1.0	1.2	1.0
	5	1.8	3.2	1.9	1.8	0.9
	6	1.8	1.7	1.7	1.5	0.8
	7	2.4	2.3	2.2	1.7	0.9
	8	2.1	1.9	2.1	1.6	1.0
	9	1.8	1.9	2.1	1.7	1.0
	10	1.9	1.7	2.0	1.5	0.9
	11	1.7	1.8	2.0	1.6	1.0
	12	1.3	1.9	2.1	1.7	1.0
5	4	0.9	2.0	0.9	1.1	1.0
	5	0.9	1.6	0.9	1.5	1.0
	6	1.8	2.1	1.9	1.6	1.0
	7	2.3	2.1	1.9	1.6	0.9
	8	2.1	1.9	2.1	1.6	0.9
	9	1.9	2.1	2.3	1.8	0.9
	10	2.0	2.0	2.2	1.7	0.9
	11	1.7	1.9	2.1	1.7	0.9
	12	1.2	1.9	2.1	1.7	1.0

Table A.12: Full list of speed-ups for the pedestrian evacuation scenario 2 in Section 6.4 with their variations: $cn = 2, 3, 4, 5$ denotes the number of used cluster nodes (each cluster node has 28 CPU cores), and $q = 2, 3, \ldots, 12$ is the number of collocation points for each uncertain parameter.

A.5 Case Study: UQ with Vadere scenario 1 and Monte Carlo

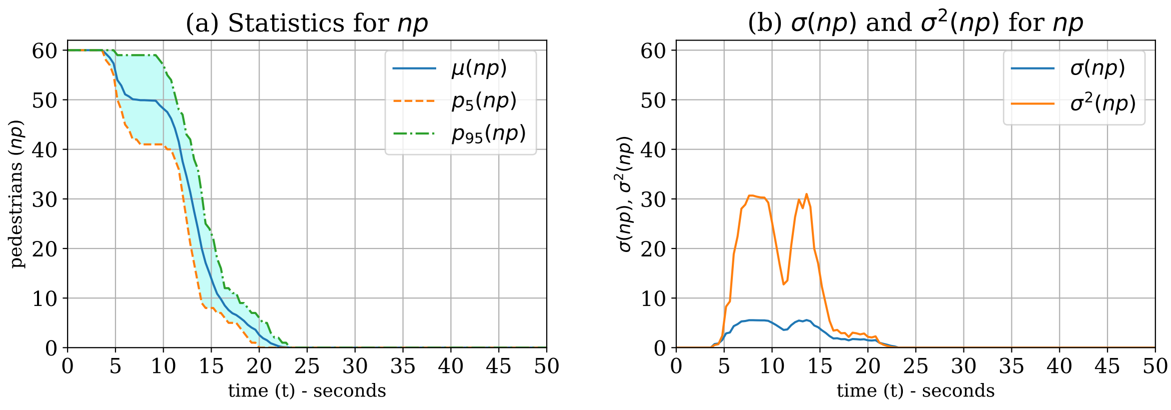


Figure A.5: QoI results (for a Monte Carlo UQ simulation with 100,000 performed samples) for the number of pedestrians np remaining in the car of the train with *pedestrians sharing a social identity* ($perc_{sharingSI}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

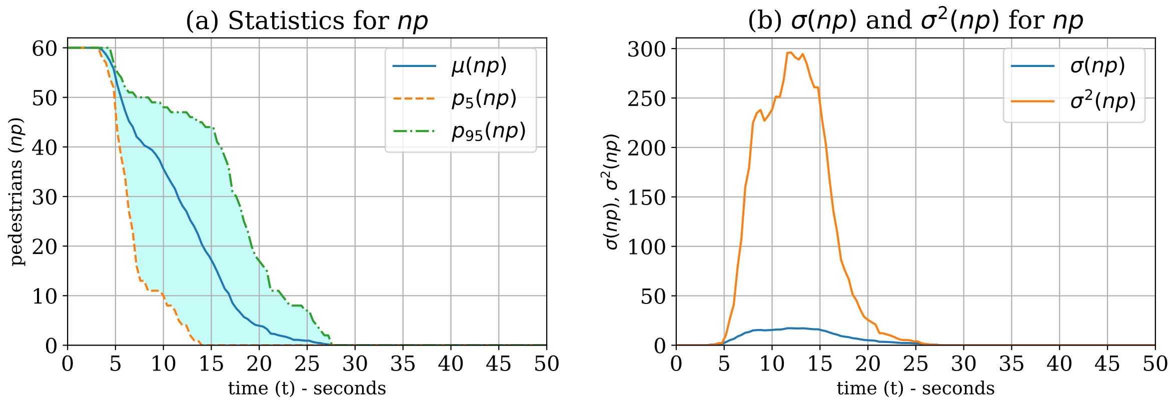
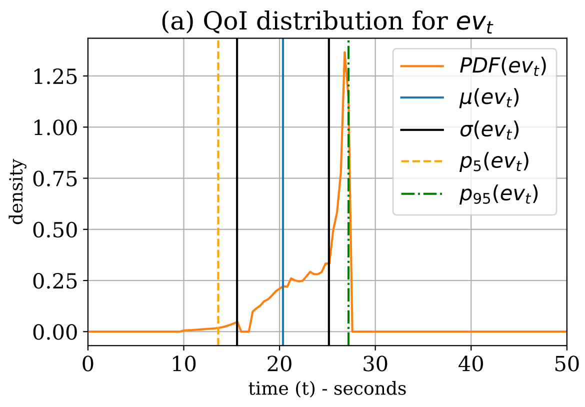


Figure A.6: QoI results (for a Monte Carlo UQ simulation with 100,000 performed samples) for the number of pedestrians np remaining in the car of the train with *percentage of injured pedestrians* ($perc_{injPeds}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.



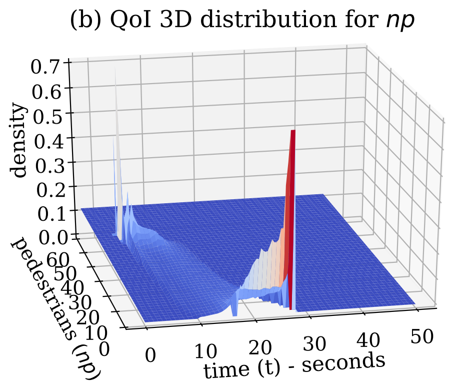
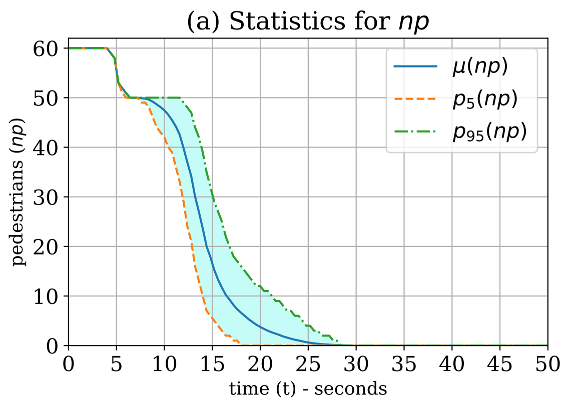


Figure A.7: QoI distribution reconstruction (for a Monte Carlo UQ simulation with 100,000 performed samples) for *percentage of injured pedestrians* ($perc_{injPeds}$) as the uncertain parameter for scenario 1: evacuation of a train station. (a) contains the plots of the reconstructed QoI distribution of the maximum evacuation time ev_t. (b) visualises the reconstructed QoI 3D distribution of the number of pedestrians np in the car for every time step in the simulation.



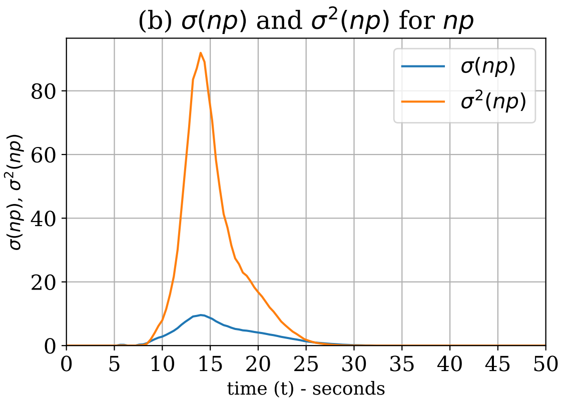


Figure A.8: QoI results (for a Monte Carlo UQ simulation with 100,000 performed samples) for the number of pedestrians np remaining in the car of the train with *speed of a helper with an injured pedestrian* (v_{inj}) as the uncertain parameter for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

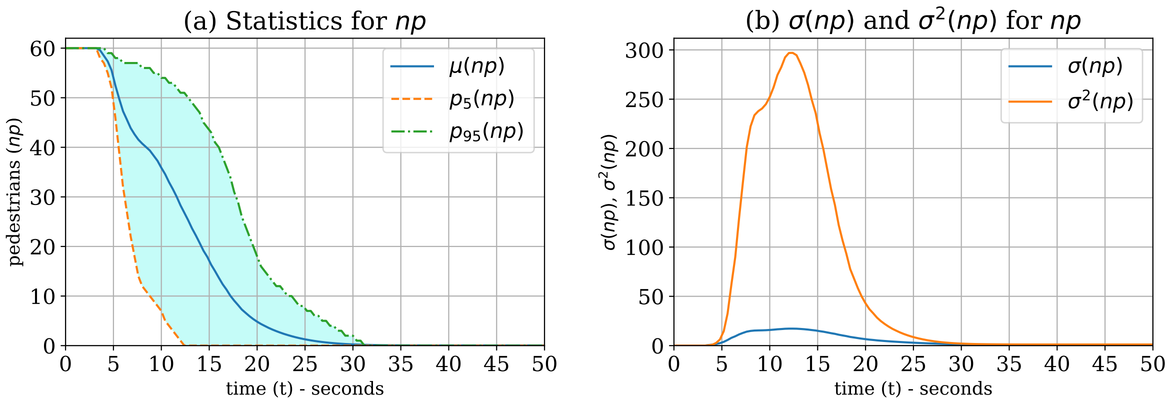


Figure A.9: QoI results (for a Monte Carlo UQ simulation with 100,000 performed samples) for the number of pedestrians np remaining in the car of the train with all of the three uncertain parameters (Table 6.5) for scenario 1: evacuation of a train station. (a) shows the mean $\mu(np)$ and the percentiles ($p_5(np)$ and $p_{95}(np)$) for each time step; (b) shows the corresponding standard deviation $\sigma(np)$ and the variance $\sigma^2(np)$ for every time step.

Bibliography

[1] Adams, B.M., Bohnhoff, W.J., Dalbey, K.R., Ebeida, M.S., Eddy, J.P., Eldred, M.S., Geraci, G., Hooper, R.W., Hough, P.D., Hu, K.T., Jakeman, J.D., Khalil, M., Maupin, K.A., Monschke, J.A., Ridgway, E.M., Rushdi, A.A., Stephens, J.A., Swiler, L.P., Vigil, D.M., Wildey, T.M., and Winokur, J.G.: Dakota, A Multilevel Parallel Object-Oriented Framework for Design Optimization, Parameter Estimation, Uncertainty Quantification, and Sensitivity Analysis: Version 6.11 User's Manual. Tech. rep., Sandia Technical Report SAND2014-4633 (July 2014; updated November 2019)

[2] Asher, M.J., Croke, B., Jakeman, A., Peeters, L.: A Review of Surrogate Models and Their Application to Groundwater Modeling. Water Resources Research **Volume 51**(8), 5957–5973 (2015). DOI 10.1002/2015WR016967

[3] Aveni, A.F.: The Not-So-Lonely Crowd: Friendship Groups in Collective Behavior. Sociometry **40**(1), 96–99 (1977). DOI 10.2307/3033551

[4] Baker, W., Barnett, J., Marrion, C., Milke, J., Nelson, H.: World Trade Center Building Performance Study: Data Collection, Preliminary Observations, and Recommendations, chap. Chapter 2: WTC1 and WTC2, pp. 2–1 – 2–40. Federal Emergency Management Agency (2002)

[5] Barnes, M., Abel, I.G., Dorland, W., Görler, T., Hammett, G.W., Jenko, F.: Direct Multiscale Coupling of a Transport Code to Gyrokinetic Turbulence Codes. Physics of Plasmas (2010). URL `https://aip.scitation.org/doi/abs/10.1063/1.3323082#Metrics-content`

[6] Baudin, M., Dutfoy, A., Iooss, B., Popelin, A.L.: Open TURNS: An industrial software for uncertainty quantification in simulation. arXiv preprint arXiv:1501.05242 (2015)

[7] Becker, A.: Neue Anforderungen und Lösungen bei der großflächigen hydrologischen Modellierung. Wasserwirtschaft, Wasertechnik **7**, 150–152 (1986)

[8] Benner, P., Gugercin, S., Willcox, K.: A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems. SIAM Rev. **Volume 57, Issue 4**, 483–531 (2015). DOI 10.1137/130932715

[9] Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., Weglarz, J.: Scheduling Computer and Manufacturing Processes. Springer Berlin Heidelberg (2001)

[10] Bock, H., Carraro, T., Jäger, W., Körkel, S., Rannacher, R., Schlöder, J.: Model Based Parameter Estimation: Theory and Applications. Contributions in Mathematical and Computational Sciences. Springer Berlin Heidelberg (2013)

[11] Brown, R.: Animated Visual Vibrations as an Uncertainty Visualisation Technique. In: Proceedings of the 2nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia, GRAPHITE '04, p. 84–89. Association for Computing Machinery, New York, NY, USA (2004). DOI 10.1145/988834.988849. URL `https://doi.org/10.1145/988834.988849`

[12] Caflisch, R.E.: Monte Carlo and quasi-Monte Carlo methods. Acta Numerica **7**, 1–49 (1998). DOI 10.1017/S0962492900002804

[13] Chen, C., Härdle, W., Unwin, A.: Handbook of Data Visualization. Springer Handbooks of Computational Statistics. Springer Berlin Heidelberg (2007)

[14] Coffman, E.G., Garey, M.R., Johnson, D.S.: An Application of Bin-Packing to Multiprocessor Scheduling. SIAM Journal on Computing **7**(1), 1–17 (1978). DOI 10.1137/0207001. URL `https://epubs.siam.org/doi/abs/10.1137/0207001`

[15] Coifman, R.R., Lafon, S.: Diffusion Maps. Appl. Comput. Harmon. Anal. **Volume 21**(1), 5–30 (2006). DOI 10.1016/j.acha.2006.04.006

[16] Cornwell, B., Harmon, W., Mason, M., Merz, B., Lampe, M.: Panic or Situational Constraints? The Case of the M/V Estonia. International journal of mass emergencies and disasters **19**(1), 5–25 (2001)

[17] Crowd simulation team at Munich University of Applied Sciences: openVADERE Simulation Framework. www.vadere.org (2016)

[18] Cui, J., Lin, D.: Utilisation of underground pedestrian systems for urban sustainability. Tunnelling and Underground Space Technology **55**, 194 – 204 (2016). DOI https://doi.org/10.1016/j.tust.2015.11.004. URL `http://www.sciencedirect.com/science/article/pii/S0886779815302662`. Urban Underground Space: A Growing Imperative Perspectives and Current Research in Planning and Design for Underground Space Use

[19] Dalcín, L., Paz, R., Storti, M.: MPI for Python. Journal of Parallel and Distributed Computing **65**(9), 1108 – 1115 (2005). DOI https://doi.org/10.1016/j.jpdc.2005.03.010. URL `http://www.sciencedirect.com/science/article/pii/S0743731505000560`

[20] Debusschere, B., Sargsyan, K., Safta, C., Chowdhary, K.: The Uncertainty Quantification Toolkit (UQTk). In: R. Ghanem, D. Higdon, H. Owhadi (eds.) Handbook of Uncertainty Quantification, pp. 1807–1827. Springer (2017)

[21] D'Elia, M., Phipps, E., Rushdi, A., Ebeida, M.: Surrogate-based Ensemble Grouping Strategies for Embedded Sampling-based Uncertainty Quantification. arXiv e-prints (2017)

[22] Denman, S., Fookes, C., Ryan, D., Sridharan, S.: Large scale monitoring of crowds and building utilisation: A new database and distributed approach. In: 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), pp. 1–6 (2015)

[23] Devia, G.K., Ganasri, B., Dwarakish, G.: A Review on Hydrological Models. Aquatic Procedia **4**, 1001 – 1007 (2015). DOI https://doi.org/10.1016/j.aqpro.2015.02.126. URL `http://www.sciencedirect.com/science/article/pii/S2214241X15001273`. INTERNATIONAL CONFERENCE ON WATER RESOURCES, COASTAL AND OCEAN ENGINEERING (ICWRCOE'15)

[24] Dietrich, F.: Data-Driven Surrogate Models for Dynamical Systems. Dissertation, Technische Universität München, München (2017)

[25] Dietrich, F., Albrecht, F., Köster, G.: Surrogate Models for Bottleneck Scenarios. In: Proc. of 8th Int. Conf. on Pedestrian and Evacuation Dynamics (PED2016). Hefei, China (2016)

[26] Dietrich, F., Köster, G.: Gradient Navigation Model for Pedestrian Dynamics. Phys. Rev. E **Volume 89**(6), 062801 (2014). DOI 10.1103/PhysRevE.89.062801

[27] Dietrich, F., Köster, G., Bungartz, H.J.: Numerical Model Construction with Closed Observables. SIAM J. Appl. Dyn. Syst. **Volume 15**(4), pp. 2078–2108 (2016). DOI 10.1137/15M1043613

[28] Dietrich, F., Künzner, F., Neckel, T., Köster, G., Bungartz, H.J.: FAST AND FLEXIBLE UNCERTAINTY QUANTIFICATION THROUGH A DATA-DRIVEN SURROGATE MODEL. International Journal for Uncertainty Quantification **8**(2), 175–192 (2018)

[29] Drozdowski, M.: Scheduling for Parallel Processing. Computer Communications and Networks. Springer London (2010)

[30] Drury, J., Cocking, C., Reicher, S.: Everyone for themselves? A comparative study of crowd solidarity among emergency survivors. British Journal of Social Psychology **28**, 487–506 (2009). DOI 10.1348/014466608X357893

[31] Drury, J., Cocking, C., Reicher, S.: The nature of collective resilience: Survivor reactions to the 2005 London bombings. International Journal of Mass Emergencies and Disasters **27**(1), 66–95 (2009)

[32] Drury, J., Cocking, C., Reicher, S., Burton, A., Schofield, D., Hardwick, A., Graham, D., Langston, P.: Cooperation versus competition in a mass emergency evacuation: A new laboratory simulation and a new theoretical model. Behavior Research Methods **41**(3), 957–970 (2009). DOI 10.3758/BRM.41.3.957

[33] Du, Y., Mak, C.M.: Improving pedestrian level low wind velocity environment in high-density cities: A general framework and case study. Sustainable Cities and Society **42**, 314 – 324 (2018). DOI https://doi.org/10.1016/j.scs.2018.08.001. URL `http://www.sciencedirect.com/science/article/pii/S2210670718308217`

[34] Eldred, M., Burkardt, J.: Comparison of Non-Intrusive Polynomial Chaos and Stochastic Collocation Methods for Uncertainty Quantification. 47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition (2009). DOI 10.2514/6.2009-976. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2009-976`

[35] Elman, H.C., Miller, C.W.: Stochastic collocation with kernel density estimation. Computer methods in applied mechanics and engineering **245**, 36–46 (2012)

[36] Facchi, A., Negri, C., Rienzner, M., Chiaradia, E., Romani, M.: Groundwater Recharge Through Winter Flooding of Rice Areas. In: A. Coppola, G.C. Di Renzo, G. Altieri, P. D'Antonio (eds.) Innovative Biosystems Engineering for Sustainable Agriculture, Forestry and Food Production, pp. 79–87. Springer International Publishing, Cham (2020)

[37] Fang, S., Xu, L., Pei, H., Liu, Y., Liu, Z., Zhu, Y., Yan, J., Zhang, H.: An Integrated Approach to Snowmelt Flood Forecasting in Water Resource Management. IEEE Transactions on Industrial Informatics **10**(1), 548–558 (2014)

[38] Farcaş, I.G., Görler, T., Bungartz, H.J., Jenko, F., Neckel, T.: Sensitivity-driven adaptive sparse stochastic approximations in plasma microinstability analysis. arXiv e-prints arXiv:1812.00080 (2018)

[39] Farcaş, I.G., Sârbu, P.C., Bungartz, H.J., Neckel, T., Uekermann, B.: Multilevel Adaptive Stochastic Collocation with Dimensionality Reduction. In: J. Garcke, D. Pflüger, C.G.

Webster, G. Zhang (eds.) Sparse Grids and Applications - Miami 2016, pp. 43–68. Springer International Publishing, Cham (2018)

[40] Farcas, I.G., Uekermann, B., Neckel, T., Bungartz, H.J.: Nonintrusive Uncertainty Analysis of Fluid-Structure Interaction with Spatially Adaptive Sparse Grids and Polynomial Chaos Expansion. SIAM J. Scientific Computing **40** (2018)

[41] Farcaş, I.G., Latz, J., Ullmann, E., Neckel, T., Bungartz, H.J.: Multilevel Adaptive Sparse Leja Approximations for Bayesian Inverse Problems. SIAM Journal on Scientific Computing **42**(1), A424–A451 (2020). DOI 10.1137/19M1260293. URL `https://doi.org/10.1137/19M1260293`

[42] Feinberg, J., Eck, V.G., Langtangen, H.P.: MULTIVARIATE POLYNOMIAL CHAOS EXPANSIONS WITH DEPENDENT VARIABLES. SIAM Journal on Scientific Computing **40**(1), A199 – A223 (2018). URL `http://search.ebscohost.com.eaccess.ub.tum.de/login.aspx?direct=true&db=asn&AN=128496969&site=ehost-live`

[43] Feinberg, J., Langtangen, H.P.: Chaospy: An open source tool for designing methods of uncertainty quantification. Journal of Computational Science **Volume 11**, Pages 46–57 (2015). DOI 10.1016/j.jocs.2015.08.008

[44] Fernandes, M., Walls, L., Munson, S., Hullman, J., Kay, M.: Uncertainty Displays Using Quantile Dotplots or CDFs Improve Transit Decision-Making. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI '18, p. 1–12. Association for Computing Machinery, New York, NY, USA (2018). DOI 10.1145/3173574.3173718. URL `https://doi.org/10.1145/3173574.3173718`

[45] Franzelin, F., Diehl, P., Pflüger, D.: Non-intrusive Uncertainty Quantification with Sparse Grids for Multivariate Peridynamic Simulations. In: M. Griebel, M.A. Schweitzer (eds.) Meshfree Methods for Partial Differential Equations VII, pp. 115–143. Springer International Publishing (2015)

[46] Franzelin, F., Pflüger, D.: From Data to Uncertainty: An Efficient Integrated Data-Driven Sparse Grid Approach to Propagate Uncertainty. In: J. Garcke, D. Pflüger (eds.) Sparse Grids and Applications - Stuttgart 2014, pp. 29–49. Springer International Publishing (2016)

[47] Gael Varoquaux: joblib. URL `https://github.com/joblib/joblib`

[48] Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software., 1 edn. Prentice Hall (1994)

[49] Ganapathysubramanian, B., Zabaras, N.: Sparse grid collocation schemes for stochastic natural convection problems. Journal of Computational Physics **225**(1), 652 – 685 (2007). DOI https://doi.org/10.1016/j.jcp.2006.12.014. URL `http://www.sciencedirect.com/science/article/pii/S0021999106006152`

[50] Gaudier, F.: URANIE: The CEA/DEN Uncertainty and Sensitivity platform. Procedia - Social and Behavioral Sciences **2**(6), 7660 – 7661 (2010). DOI https://doi.org/10.1016/j.sbspro.2010.05.166. URL `http://www.sciencedirect.com/science/article/pii/S1877042810013078`. Sixth International Conference on Sensitivity Analysis of Model Output

[51] The weather in Germany in March 2013. URL `https://www.dwd.de/EN/press/press_release/EN/2013/20130409_DeutschlandwetterimMaerz2013_e.pdf?__blob=publicationFile&v=7`

[52] Gerstner, T., Griebel, M.: Dimension–Adaptive Tensor–Product Quadrature. Computing **71**(1), 65–87 (2003). DOI 10.1007/s00607-003-0015-5. URL `https://doi.org/10.1007/s00607-003-0015-5`

[53] Ghanem, R., Higdon, D., Owhadi, H.: Handbook of uncertainty quantification, vol. 6. Springer (2017)

[54] Ghanem, R., Spanos, P.: Stochastic Finite Elements: A Spectral Approach. Civil, Mechanical and Other Engineering Series. Dover Publications (2003)

[55] Giles, M.B.: Multilevel Monte Carlo methods. Acta Numerica **24**, 259–328 (2015). DOI 10.1017/S096249291500001X

[56] Google maps: District Regen. URL `https://www.google.com/maps/place/Regen/@48.9986283,12.9506176,65357m/data=!3m1!1e3!4m5!3m4!1s0x47753c90c2e85ccb:0x92ce5ca7f8f099f0!8m2!3d48.9771334!4d13.1283578`

[57] Gramacki, A.: Nonparametric kernel density estimation and its computational aspects. Springer (2018)

[58] Griebel, M., Bungartz, H.J.: Sparse Grids. Acta Numerica **Volume 13**, 147–269 (2004)

[59] Gupta, J.N.D., Ruiz-Torres, A.J.: A LISTFIT heuristic for minimizing makespan on identical parallel machines. Production Planning & Control **12**(1), 28–36 (2001). DOI 10.1080/09537280150203951. URL `https://doi.org/10.1080/09537280150203951`

[60] Haag, I., Johst, M., Sieber, A., Bremicker, M.: Guideline for the Calibration of LARSIM Water Balance Models for operational Application in Flood Forecasting. Calibration guide (2016)

[61] Hadjidoukas, P.E., Angelikopoulos, P., Papadimitriou, C., Koumoutsakos, P.: Π4U: A high performance computing framework for Bayesian uncertainty quantification of complex models. J. Comput. Phys. **284**, 1–21 (2015). DOI https://doi.org/10.1016/j.jcp.2014.12.006. URL `http://www.cse-lab.ethz.ch/wp-content/papercite-data/pdf/hadjidoukas2015b.pdf`

[62] Hammersley, J.M.: MONTE CARLO METHODS FOR SOLVING MULTIVARIABLE PROBLEMS. Annals of the New York Academy of Sciences **86**(3), 844–874 (1960). DOI 10.1111/j.1749-6632.1960.tb42846.x. URL `https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1960.tb42846.x`

[63] Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour, R.: Evaluation of Job-Scheduling Strategies for Grid Computing. In: Grid Computing — GRID 2000, pp. 191–202. Springer Berlin Heidelberg, Berlin, Heidelberg (2000)

[64] Helbing, D., Molnár, P.: Social Force Model for pedestrian dynamics. Physical Review E **51**(5), 4282–4286 (1995). DOI 10.1103/PhysRevE.51.4282

[65] Hengl, T.: Visualisation of uncertainty using the HSI colour model: computations with colours pp. 1–12 (2003)

[66] Hesthaven, J., Gottlieb, S., Gottlieb, D.: Spectral Methods for Time-Dependent Problems. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press (2007)

[67] Heuveline, V., Schick, M., Webster, C., Zaspel, P.: Uncertainty Quantification and High Performance Computing (Dagstuhl Seminar 16372). Dagstuhl Reports **6**(9), 59–73 (2017). DOI 10.4230/DagRep.6.9.59. URL `http://drops.dagstuhl.de/opus/volltexte/2017/6915`

[68] Hofman, J.M., Goldstein, D.G., Hullman, J.: How Visualizing Inferential Uncertainty Can Mislead Readers About Treatment Effects in Scientific Results. In: Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, CHI '20, p. 1–12. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3313831.3376454. URL `https://doi.org/10.1145/3313831.3376454`

[69] Hosder, S., Walters, R., Balch, M.: Efficient Sampling for Non-Intrusive Polynomial Chaos Applications with Multiple Uncertain Input Variables (2007). DOI 10.2514/6.2007-1939. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2007-1939`

[70] Hullman, J.: Why Authors Don't Visualize Uncertainty. IEEE Transactions on Visualization and Computer Graphics **26**(1), 130–139 (2020)

[71] Hürsch, W.L., Lopes, C.V.: Separation of Concerns. Tech. rep., Northeastern University Boston (1995)

[72] Iaccarino, G.: Quantification of Uncertainty in Flow Simulations using Probabilistic Methods. In: VKI Lecture Series, Sept. 8 – 12 (2008)

[73] Institut de Radioprotection et de Surete Nucleaire: The PROMETHEE project. URL `https://www.irsn.fr/EN/Research/Scientific-tools/Computer-codes/Pages/PROMETHEE-project-5069.aspx`

[74] ISO/TC 159/SC 4: Ergonomics of human-system interaction - Part 11: Usability: Definitions and concepts. Standard ISO 9241-11:2018, International Organization for Standardization, London, UK (2018)

[75] James, J.: The Distribution of Free-Forming Small Group Size. American Sociological Review **18**(5), 569–570 (1953)

[76] Jaros, M., Di Angelo, M., Ferschin, P.: Modeling and simulation of pedestrian behaviour: As planning support for building design. In: 2016 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH), pp. 1–8 (2016)

[77] Jayawardena, A.: Environmental and Hydrological Systems Modelling. Taylor & Francis (2014)

[78] Johnson, C.: Lessons from the evacuation of the World Trade Center, Sept 11th 2001 for the future development of computer simulations. Cognition, Technology and Work **7**(4), 214–240 (2005). URL `http://eprints.gla.ac.uk/3456/`

[79] Jolliffe, I.T.: Principal component analysis. Springer, New York, NY (2004). DOI 10.1007/b98835

[80] Jonathan, Feinberg: Chaospy. URL `https://github.com/jonathf/chaospy`

[81] Jr., R.V.F., Constantine, P., Boslough, M.: Statistical Surrogate Models for Prediction of High-Consequence Climate Change. Int. J. Uncertainty Quantif. **Volume 3**(4), 341–355 (2013)

[82] JSON. URL https://www.json.org/

[83] Kale, A., Kay, M., Hullman, J.: Decision-Making Under Uncertainty in Research Synthesis: Designing for the Garden of Forking Paths. In: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI ’19, p. 1–14. Association for Computing Machinery, New York, NY, USA (2019). DOI 10.1145/3290605.3300432. URL https://doi.org/10.1145/3290605.3300432

[84] Kardos, J., Benwell, G., Moore, A.: The Visualisation of Uncertainty for Spatially Referenced Census Data Using Hierarchical Tessellations. Transactions in GIS **9**(1), 19–34 (2005). DOI 10.1111/j.1467-9671.2005.00203.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9671.2005.00203.x

[85] Kawai, S., Shimoyama, K.: Kriging-model-based uncertainty quantification in computational fluid dynamics. DOI 10.2514/6.2014-2737. URL https://arc.aiaa.org/doi/abs/10.2514/6.2014-2737

[86] Kennedy, M.C., O’Hagan, A.: Bayesian calibration of computer models. Journal of the Royal Statistical Society: Series B (Statistical Methodology) **63**(3), 425–464 (2001). DOI 10.1111/1467-9868.00294. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00294

[87] Kevrekidis, I.G., Samaey, G.: Equation-Free Multiscale Computation: Algorithms and Applications. Annual Review of Physical Chemistry **60**(1), 321–344 (2009). DOI 10.1146/annurev.physchem.59.032607.093610. URL https://doi.org/10.1146/annurev.physchem.59.032607.093610. PMID: 19335220

[88] Kinkeldey, C., MacEachren, A.M., Schiewe, J.: How to Assess Visual Communication of Uncertainty? A Systematic Review of Geospatial Uncertainty Visualisation User Studies. The Cartographic Journal **51**(4), 372–386 (2014). DOI 10.1179/1743277414Y.0000000099. URL https://doi.org/10.1179/1743277414Y.0000000099

[89] Kirk, A.: Data Visualisation: A Handbook for Data Driven Design. SAGE Publications (2019)

[90] Kleijnen, J.P.C., Ridder, A.A.N., Rubinstein, R.Y.: Variance Reduction Techniques in Monte Carlo Methods, pp. 1598–1610. Springer US, Boston, MA (2013). DOI 10.1007/978-1-4419-1153-7_638. URL https://doi.org/10.1007/978-1-4419-1153-7_638

[91] Kleinmeier, B., Zönnchen, B., Gödel, M., Köster, G.: Vadere: An Open-Source Simulation Framework to Promote Interdisciplinary Understanding. Collective Dynamics **4**, 1–34 (2019). DOI 10.17815/CD.2019.21. URL https://collective-dynamics.eu/index.php/cod/article/view/A21

[92] Kloeden, P.E., Platen, E.: Numerical Solution of Stochastic Differential Equations. Stochastic Modelling and Applied Probability. Springer Berlin Heidelberg, Berlin, Heidelberg (1992). DOI 10.1007/978-3-662-12616-5. URL http://link.springer.com/10.1007/978-3-662-12616-5

[93] Knauf, D.: Die Berechnung des Abflusses aus einer Schneedecke. Analyse und Berechnung oberirdischer Abflüsse **46**, 97–133 (1980)

[94] Kneusel, R.: Random Numbers and Computers. Springer International Publishing (2018)

[95] Knijff, J.M.V.D., Younis, J., Roo, A.P.J.D.: LISFLOOD: a GIS-based distributed model for river basin scale water balance and flood simulation. International Journal of Geographical Information Science **24**(2), 189–212 (2010). DOI 10.1080/13658810802549154

[96] Knöll, P., Zirlewagen, J., Scheytt, T.: Using radar-based quantitative precipitation data with coupled soil- and groundwater balance models for stream flow simulation in a karst area. Journal of Hydrology **586**, 124884 (2020). DOI https://doi.org/10.1016/j.jhydrol.2020.124884. URL `http://www.sciencedirect.com/science/article/pii/S0022169420303449`

[97] Koch, K.: Parameter Estimation and Hypothesis Testing in Linear Models. Springer Berlin Heidelberg (2013)

[98] Köster, G., Seitz, M., Treml, F., Hartmann, D., Klein, W.: On modelling the influence of group formations in a crowd. Contemporary Social Science **6**(3), 397–414 (2011). DOI 10.1080/21582041.2011.619867

[99] Köster, G., Treml, F., Gödel, M.: Avoiding numerical pitfalls in social force models. Physical Review E **87**(6), 063305 (2013). DOI 10.1103/PhysRevE.87.063305

[100] Kroese, D.P., Taimre, T., Botev, Z.I.: Handbook of Monte Carlo Methods. Wiley (2013)

[101] Kubicek, M., Minisci, E., Cisternino, M.: High Dimensional Sensitivity Analysis using Surrogate Modeling and High Dimensional Model Representation. Int. J. Uncertainty Quantif. **Volume 5**(5), 393–414 (2015)

[102] Kunde, M.: A multifit algorithm for uniform multiprocessor scheduling. In: Theoretical Computer Science, pp. 175–185. Springer Berlin Heidelberg, Berlin, Heidelberg (1982)

[103] Künzner, F., Neckel, T., Bungartz, H.J.: Prediction and reduction of runtime in non-intrusive forward UQ simulations. Springer Nature Applied Sciences **Volume 1**, Article number 1038 (2019). DOI https://doi.org/10.1007/s42452-019-1066-3

[104] Laha, D., Behera, D.K.: A Comprehensive Review and Evaluation of LPT, MULTIFIT, COMBINE and LISTFIT for Scheduling Identical Parallel Machines. Int. J. Inf. Commun. Techol. **11**(2), 151–165 (2017). DOI 10.1504/IJICT.2017.086246. URL `https://doi.org/10.1504/IJICT.2017.086246`

[105] The Water Balance Model LARSIM. URL `http://larsim.info`

[106] Lautenschlager, F.: Effiziente Speicherung von Zeitreihen mit Betriebsdaten aus Software-Systemen zur Analyse von Laufzeitanomalien. Dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen (2019)

[107] Leach, J.: Why People Freeze in an Emergency: Temporal and Cognitive Constraints on Survival Responses. Aviation, Space, and Environmental Medicine **75**(6), 539–542 (2004). URL `http://www.ingentaconnect.com/content/asma/asem/2004/00000075/00000006/art00011`

[108] Leach, J.: Cognitive Paralysis in an Emergency: The Role of the Supervisory Attentional System. Aviation, Space, and Environmental Medicine **76**(2), 13–136 (2005). URL `http://www.ingentaconnect.com/content/asma/asem/2005/00000076/00000002/art00010`

[109] Lee, C.Y., Massey], J.D.: Multiprocessor scheduling: combining LPT and MULTIFIT. Discrete Applied Mathematics **20**(3), 233 – 242 (1988). DOI https://doi.org/10.1016/0166-218X(88)90079-0. URL `http://www.sciencedirect.com/science/article/pii/0166218X88900790`

[110] Lerche, I., Mudford, B.S.: How many Monte Carlo simulations does one need to do? Energy exploration & exploitation **23**(6), 405–427 (2005)

[111] Levine, M., Prosser, A., Evans, D., Reicher, S.: Identity and emergency intervention: how social group membership and inclusiveness of group boundaries shape helping behavior. Personality and social psychology bulletin **31**(4), 443–453 (2005)

[112] Li, L., Simonovic, S.P.: System dynamics model for predicting floods from snowmelt in North American prairie watersheds. Hydrological Processes **16**(13), 2645–2666 (2002). DOI 10.1002/hyp.1064. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/hyp.1064`

[113] Lisandro Dalcín: mpi4py. URL `https://pypi.org/project/mpi4py`

[114] Leibniz Supercomputing Centre. URL `https://www.lrz.de`

[115] Linux-Cluster of Leibniz Supercomputing Centre. URL `https://www.lrz.de/services/compute/linux-cluster`

[116] Lu, Y., Sarkar, C., Ye, Y., Xiao, Y.: Using the Online Walking Journal to explore the relationship between campus environment and walking behaviour. Journal of Transport & Health **5**, 123 – 132 (2017). DOI https://doi.org/10.1016/j.jth.2016.12.006. URL `http://www.sciencedirect.com/science/article/pii/S2214140516302183`. Walking and Walkability: A review of the evidence on health

[117] Ludwig, K., Bremicker, M.: The Water Balance Model LARSIM - Design, Content and Applications. FREIBURGER SCHRIFTEN ZUR HYDROLOGIE **22** (2006). URL `http://www.larsim.de/fileadmin/files/Dokumentation/FSH-Bd22-Bremicker-Ludwig.pdf`

[118] Mai, C.V., Spiridonakos, M.D., Chatzi, E.N., Sudret, B.: Surrogate Modeling for Stochastic Dynamical Systems by Combining Nonlinear Autoregressive with Exogenous Input Models and Polynomial Chaos Expansions. Int. J. for Uncertainty Quantif. **Volume 6**(4), 313–339 (2016)

[119] Manenti, L., Manzoni, S., Vizzari, G., Ohtsuka, K., Shimura, K.: An agent-based proxemic model for pedestrian and group dynamics: motivations and first experiments. In: Proceedings of the 12th international conference on Multi-Agent-Based Simulation, MABS'11, pp. 74–89. Springer, Taipei, Taiwan (2012)

[120] Marelli, S., Sudret, B.: UQLab: a framework for uncertainty quantification in MATLAB

[121] Marzouk, Y., Willcox, K.: Uncertainty Quantification. In: N. Higham, M. Dennis, P. Glendinning, P. Martin, F. Santosa, J. Tanner (eds.) The Princeton Companion to Applied Mathematics. Princeton University Press (2015)

[122] Mawson, A.R.: Mass Panic and Social Attachment: The Dynamics of Human Behavior. Ashgate Publishing Limited (2007)

[123] McClarren, R.: Uncertainty Quantification and Predictive Computational Science: A Foundation for Physical Scientists and Engineers. Springer International Publishing (2018)

[124] McKay, M.D., Beckman, R.J., Conover, W.J.: A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. Technometrics **21**(2), 239–245 (1979). URL `http://www.jstor.org/stable/1268522`

[125] McKerns, M.M., Strand, L., Sullivan, T., Fang, A., Aivazis, M.A.: Building a framework for predictive science. arXiv preprint arXiv:1202.1056 (2012)

[126] Meeds, E., Welling, M.: Optimization Monte Carlo: Efficient and Embarrassingly Parallel Likelihood-Free Inference. In: NIPS (2015)

[127] Message Passing Interface Forum: MPI: A Message-Passing Interface Standard, Version 3.1. Specification (2015). URL `https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf`

[128] Metropolis, N., Ulam, S.: The Monte Carlo Method. Journal of the American Statistical Association **44**(247), 335–341 (1949). URL `http://www.jstor.org/stable/2280232`

[129] Michael Obersteiner: sparseSpACE - The Sparse Grid Spatially Adaptive Combination Environment. URL `https://github.com/obersteiner/sparseSpACE`

[130] Moussaïd, M., Perozo, N., Garnier, S., Helbing, D., Theraulaz, G.: The Walking Behaviour of Pedestrian Social Groups and Its Impact on Crowd Dynamics. PLoS ONE **5**(4), e10047 (2010). DOI 10.1371/journal.pone.0010047

[131] Neiswanger, W., Wang, C., Xing, E.P.: Asymptotically Exact, Embarrassingly Parallel MCMC. In: UAI (2014)

[132] Neuendorf, K., Mehl, J., Jackson, J.: Glossary of Geology. Glossary of Geology. Springer Berlin Heidelberg (2011)

[133] Oberle, W.: Monte Carlo simulations: Number of iterations and accuracy. Tech. rep., ARMY RESEARCH LAB ABERDEEN PROVING GROUND MD WEAPONS AND MATERIALS RESEARCH DIRECTORATE (2015)

[134] Owen, A.B.: Monte Carlo theory, methods and examples (2013). URL `https://statweb.stanford.edu/~owen/mc/`

[135] Palmer, T.N.: Predicting uncertainty in forecasts of weather and climate. Reports on Progress in Physics **63**(2), 71–116 (2000). DOI 10.1088/0034-4885/63/2/201. URL `https://doi.org/10.1088/0034-4885/63/2/201`

[136] Parno, M., Davis, A., Conrad, P., Marzouk, Y.: MIT uncertainty quantification (MUQ) library (2014)

[137] Patelli, E.: COSSAN: A Multidisciplinary Software Suite for Uncertainty Quantification and Risk Management, pp. 1–69. Springer International Publishing, Cham (2016). DOI 10.1007/978-3-319-11259-6_59-1. URL `https://doi.org/10.1007/978-3-319-11259-6_59-1`

[138] Paulson, J.A., Martin-Casas, M., Mesbah, A.: Fast uncertainty quantification for dynamic flux balance analysis using non-smooth polynomial chaos expansions. PLOS Computational Biology **15**(8), 1–35 (2019). DOI 10.1371/journal.pcbi.1007308. URL `https://doi.org/10.1371/journal.pcbi.1007308`

[139] Paulson, J.A., Martin-Casas, M., Mesbah, A.: Fast uncertainty quantification for dynamic flux balance analysis using non-smooth polynomial chaos expansions. PLOS Computational Biology **15**(8), 1–35 (2019). DOI 10.1371/journal.pcbi.1007308. URL `https://doi.org/10.1371/journal.pcbi.1007308`

[140] Peherstorfer, B., Beran, P.S., Willcox, K.E.: Multifidelity Monte Carlo estimation for large-scale uncertainty propagation. DOI 10.2514/6.2018-1660. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2018-1660`

[141] Peherstorfer, B., Gugercin, S., Willcox, K.: Data-Driven Reduced Model Construction with Time-Domain Loewner Models. SIAM J. Sci. Comput. **Volume 39, Issue 5** (2017)

[142] Peherstorfer, B., Kramer, B., Willcox, K.: Combining Multiple Surrogate Models to Accelerate Failure Probability Estimation with Expensive High-Fidelity Models. J. Comput. Phys. **Volume 341**, 61–75 (2017)

[143] Peherstorfer, B., Willcox, K.: Dynamic data-driven reduced-order models. Computer Methods in Applied Mechanics and Engineering **291**, 21 – 41 (2015). DOI https://doi.org/10.1016/j.cma.2015.03.018. URL `http://www.sciencedirect.com/science/article/pii/S0045782515001280`

[144] Peherstorfer, B., Willcox, K.: Dynamic data-driven model reduction: adapting reduced models from incomplete data. Advanced Modeling and Simulation in Engineering Sciences **3**(1), 11 (2016). DOI 10.1186/s40323-016-0064-x. URL `https://doi.org/10.1186/s40323-016-0064-x`

[145] Peherstorfer, B., Willcox, K., Gunzburger, M.: Survey of Multifidelity Methods in Uncertainty Propagation, Inference, and Optimization. SIAM Review **60(3)**, 550 – 591 (2018). DOI https://doi.org/10.1137/16M1082469. URL `https://epubs.siam.org/doi/abs/10.1137/16M1082469`

[146] Perry, J.: Gait Analysis: Normal and Pathological Function. SLACK Incorporated (1992)

[147] Pflüger, D.M.: Spatially adaptive sparse grids for high-dimensional problems. Ph.D. thesis, Technische Universität München (2010)

[148] Phipps, E., D'Elia, M., Edwards, H.C., Hoemmen, M., Hu, J., Rajamanickam, S.: Embedded ensemble propagation for improving performance, portability, and scalability of uncertainty quantification on emerging computational architectures. SIAM Journal on Scientific Computing **39**(2), 162–193 (2017). DOI 10.1137/15M1044679. URL `http://epubs.siam.org/doi/abs/10.1137/15M1044679?journalCode=sjoce3`

[149] Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer (2016). DOI 10.1007/978-3-319-26580-3

[150] Proulx, G.: Evacuation from a single family house. In: Proceedings of the 4th international symposium on human behaviour in fire. Robinson College, Cambridge, UK, pp. 255–266 (2009)

[151] Prudencio, E.E., Schulz, K.W.: The parallel C++ statistical library 'QUESO': Quantification of Uncertainty for Estimation, Simulation and Optimization. In: Euro-Par 2011: Parallel Processing Workshops, pp. 398–407. Springer (2012). URL `http://dx.doi.org/10.1007/978-3-642-29737-3_44`

[152] Razavi, S., Tolson, B.A., Burn, D.H.: Review of Surrogate Modeling in Water Resources. Water Resources Research **Volume 48**(7) (2012). DOI 10.1029/2011WR011527

[153] Richardson, R.A., Wright, D.W., Edeling, W., Jancauskas, V., Lakhlili, J., Coveney, P.V.: EasyVVUQ: A Library for Verification, Validation and Uncertainty Quantification in High Performance Computing. Journal of Open Research Software **8**(1) (2020)

[154] RiMEA: Guideline for Microscopic Evacuation Analysis. RiMEA e.V. **Version 3.0.0** (2016). URL `https://rimeaweb.files.wordpress.com/2016/06/rimea_richtlinie_3-0-0_-_d-e.pdf`

[155] Rushdi, A.A., Swiler, L.P., Phipps, E.T., D'Elia, M., Ebeida, M.S.: VPS: Voronoi Piecewise Surrogate Models for High-Dimensional Data Fitting. Int. J. for Uncertainty Quantif. **Volume 7**(1), 1–21 (2017)

[156] Saltelli, A., Annoni, P., Azzini, I., Campolongo, F., Ratto, M., Tarantola, S.: Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index. Computer Physics Communications **181**(2), 259 – 270 (2010). DOI https://doi.org/10.1016/j.cpc.2009.09.018. URL `http://www.sciencedirect.com/science/article/pii/S0010465509003087`

[157] Saltelli, A., Ratto, M., Andres, T.H., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., Tarantola, S.: Global Sensitivity Analysis. The Primer. John Wiley & Sons, Ltd. (2008). DOI 10.1002/9780470725184

[158] Saltelli, A., Tarantola, S., Chan, K.: A Quantitative Model-Independent Method for Global Sensitivity Analysis of Model Output. Technometrics **Volume 41**(1), 39–56 (1999). DOI 10.1080/00401706.1999.10485594

[159] Schadschneider, A., Chraibi, M., Seyfried, A., Tordeux, A., Zhang, J.: Pedestrian Dynamics: From Empirical Results to Modeling, pp. 63–102. Springer International Publishing, Cham (2018). DOI 10.1007/978-3-030-05129-7_4. URL `https://doi.org/10.1007/978-3-030-05129-7_4`

[160] Schilders, W.H., van der Vorst, H.A., Rommes, J.: Model Order Reduction: Theory, Research Aspects and Applications. Springer-Verlag Berlin Heidelberg (2008). DOI 10.1007/978-3-540-78841-6

[161] Schilders, W.H.A., van der Vorst, H.A., Rommes, J.: Model Order Reduction: Theory, Research Aspects and Applications. Springer (2008). DOI https://doi.org/10.1007/978-3-540-78841-6

[162] Schraufstetter, F.: Development of a Prototype to Quantify the Uncertainty of the Water Balance Model LARSIM. Bachelor thesis, Techinische Universität München (2019)

[163] Schreiber, M., Riesinger, C., Neckel, T., Bungartz, H.J., Breuer, A.: Invasive Compute Balancing for Applications with Shared and Hybrid Parallelization. International Journal of Parallel Programming **43**(6), 1004–1027 (2015). DOI 10.1007/s10766-014-0336-3. URL `https://doi.org/10.1007/s10766-014-0336-3`

[164] Scott, D.W.: On optimal and data-based histograms. Biometrika **66**(3), 605–610 (1979)

[165] Scott, D.W.: Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons (2015)

[166] Seitz, M.J., Dietrich, F., Köster, G.: The effect of stepping on pedestrian trajectories. Physica A: Statistical Mechanics and its Applications **421**, 594–604 (2015). DOI 10.1016/j.physa.2014.11.064

[167] Seitz, M.J., Köster, G.: Natural discretization of pedestrian movement in continuous space. Physical Review E **86**(4), 046108 (2012). DOI 10.1103/PhysRevE.86.046108

[168] Seitz, M.J., Köster, G.: How update schemes influence crowd simulations. Journal of Statistical Mechanics: Theory and Experiment **2014**(7), P07002 (2014). DOI 10.1088/1742-5468/2014/07/P07002

[169] Silverman, B.: Density Estimation for Statistics and Data Analysis. Chapman and Hall/CRC Monographs on Statistics and Applied Probability. Chapman and Hall/CRC (1998)

[170] Sime, J.D.: Affiliative behaviour during escape to building exits. Journal of Environmental Psychology **3**(1), 21–41 (1983). DOI 10.1016/S0272-4944(83)80019-X

[171] von Sivers, I., Köster, G.: Dynamic Stride Length Adaptation According to Utility And Personal Space. Transportation Research Part B: Methodological **74**, 104 – 117 (2015). DOI 10.1016/j.trb.2015.01.009

[172] von Sivers, I., Künzner, F., Köster, G.: Pedestrian Evacuation Simulation with Separated Families. In: Proceedings of the 8th International Conference on Pedestrian and Evacuation Dynamics (PED2016) (2016). DOI http://dx.doi.org/10.17815/CD.2016.11. URL `https://collective-dynamics.eu/index.php/cod/article/view/A11`

[173] von Sivers, I., Seitz, M.J., Köster, G.: How do people search: a modelling perspective. In: Parallel Processing and Applied Mathematics, 11th International Conference, PPAM 2015, Krakow, Poland, September 6-9, 2015. Revised Selected Papers, Part II, *Lecture Notes in Computer Science*, vol. 9574, pp. 487–496. Springer (2016). DOI 10.1007/978-3-319-32152-3_45

[174] von Sivers, I., Templeton, A., Köster, G., Drury, J., Philippides, A.: Humans do not always act selfishly: Social identity and helping in emergency evacuation simulation. In: The Conference in Pedestrian and Evacuation Dynamics 2014, Transportation Research Procedia, pp. 585–593. Delft, The Netherlands (2014). DOI 10.1016/j.trpro.2014.09.099

[175] von Sivers, I., Templeton, A., Künzner, F., Köster, G., Drury, J., Philippides, A., Neckel, T., Bungartz, H.J.: Modelling social identification and helping in evacuation simulation. Safety Science **89**, 288–300 (2016). DOI http://dx.doi.org/10.1016/j.ssci.2016.07.001

[176] SLURM: SLURM. URL `https://github.com/SchedMD/slurm`

[177] SmartUQ LLC: SmartUQ predictive analytics and uncertainty quantification (UQ) software tool. URL `https://www.smartuq.com/`

[178] Smith, R.C.: Uncertainty Quantification: Theory, Implementation, and Applications. Computational Science and Engineering. Society for Industrial and Applied Mathematics (2014)

[179] Sobczyk, K.: Stochastic Differential Equations: With Applications to Physics and Engineering, mathematic edn. Springer Science & Business Media (2013)

[180] Sobol, I.M.: Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Mathematics and Computers in Simulation **55**, Pages 271–280 (2001)

[181] Stahl, K., Weiler, M., Freudiger, D., Kohn, I., Seibert, J., Vis, M., Gerlinger, K., Böhm, M.: Abflussanteile aus Schnee- und Gletscherschmelze im Rhein und seinen Zuflüssen vor dem Hintergrund des Klimawandels. Tech. rep., Albert-Ludwigs-Universität Freiburg and Universität Zürich and HYDRON GmbH Karlsruhe (2017)

[182] Stoyanov, M., Lebrun-Grandie, D., Burkardt, J., Munster, D.: Tasmanian (2013). DOI 10.11578/dc.20171025.on.1087. URL `https://github.com/ORNL/Tasmanian`

[183] Sudret, B.: Global sensitivity analysis using polynomial chaos expansions. Reliability Engineering & System Safety **93**(7), 964 – 979 (2008). DOI https://doi.org/10.1016/j.ress.2007.04.002. URL http://www.sciencedirect.com/science/article/pii/S0951832007001329. Bayesian Networks in Dependability

[184] Sui, J., Koehler, G.: Rain-on-snow induced flood events in Southern Germany. Journal of Hydrology **252**(1), 205 – 220 (2001). DOI https://doi.org/10.1016/S0022-1694(01)00460-7. URL http://www.sciencedirect.com/science/article/pii/S0022169401004607

[185] Šukys, J.: Multi-Level Monte Carlo finite volume methods for nonlinear systems of stochastic conservation laws in multi-dimensions. In: 14th International Conference on Hyperbolic Problems: Theory, Numerics, Applications (HYP2012). University of Padova (2012)

[186] Sullivan, T.: Introduction to Uncertainty Quantification, 1 edn. Springer International Publishing (2015). DOI 10.1007/978-3-319-23395-6

[187] Tabak, V.: User simulation of space utilisation : system for office building usage simulation. Ph.D. thesis, Department of the Built Environment (2009). DOI 10.6100/IR640457. Proefschrift.

[188] Tabak, V., de Vries, B., Dijkstra, J.: Simulation and Validation of Human Movement in Building Spaces. Environment and Planning B: Planning and Design **37**(4), 592–609 (2010). DOI 10.1068/b35127. URL https://doi.org/10.1068/b35127

[189] Tajfel, H., Turner, J.C.: Psychology of Intergroup Relations, chap. An integrative theory of intergroup conflict, pp. 33–47. Brooks/Cole (1979)

[190] Takens, F.: Detecting strange attractors in turbulence. In: D. Rand, L.S. Young (eds.) Dynamical Systems and Turbulence, Warwick 1980, pp. 366–381. Springer Berlin Heidelberg, Berlin, Heidelberg (1981)

[191] Templeton, A., Drury, J., Philippides, A.: From Mindless Masses to Small Groups: Conceptualizing Collective Behavior in Crowd Modeling. Review of General Psychology **19**(3), 215–229 (2015). DOI 10.1037/gpr0000032

[192] Tennøe, S., Halnes, G., Einevoll, G.T.: Uncertainpy: A Python Toolbox for Uncertainty Quantification and Sensitivity Analysis in Computational Neuroscience. Frontiers in Neuroinformatics **12**, 49 (2018). DOI 10.3389/fninf.2018.00049. URL https://www.frontiersin.org/article/10.3389/fninf.2018.00049

[193] Tong, C.: The psuade software package version 1.0. LLNL code release UCRLCODE-235523 (2007)

[194] Trefethen, L.N.: Cubature, Approximation, and Isotropy in the Hypercube. SIAM Rev. **Volume 59, Issue 3**, 469–491 (2017). DOI 10.1137/16M1066312

[195] Trifonova, T., Trifonov, D., Bukharov, D., Abrakhin, S., Arakelian, M., Arakelian, S.: Global and Regional Aspects for Genesis of Catastrophic Floods: The Problems of Forecasting and Estimation for Mass and Water Balance (Surface Water and Groundwater Contribution). IntechOpen (2020). DOI 10.5772/intechopen.91623

[196] Tsai, J., Fridman, N., Bowring, E., Brown, M., Epstein, S., Kaminka, G., Marsella, S., Ogden, A., Rika, I., Sheel, A., Taylor, M.E., Wang, X., Zilka, A., Tambe, M.: ESCAPES: Evacuation Simulation with Children, Authorities, Parents, Emotions, and

Social Comparison. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2, AAMAS '11, pp. 457 – 464. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2011). URL http://dl.acm.org/citation.cfm?id=2031678.2031682

[197] Tucker, P.M., Pfefferbaum, B., North, C.S., Adrian Kent, J., Burgin, C.E., Parker, D.E., Hossain, A., Jeon-Slaughter, H., Trautman, R.P.: Physiologic reactivity despite emotional resilience several years after direct exposure to terrorism. The American journal of psychiatry **164**(2), 230–235 (2007)

[198] Turner, J.C., Hogg, M.A., Oakes, P.J., Reicher, S.D., Wetherell, M.S.: Rediscovering the social group: A self-categorization theory. Basil Blackwell (1987)

[199] Urban Space – AECOM: Modelling Pedestrian Movement and Interactions with Traffic, 1 edn. ICE Publishing (2019)

[200] Wan, X., Karniadakis, G.E.: An adaptive multi-element generalized polynomial chaos method for stochastic differential equations. Journal of Computational Physics **209**(2), 617 – 642 (2005). DOI https://doi.org/10.1016/j.jcp.2005.03.023. URL http://www.sciencedirect.com/science/article/pii/S0021999105001919

[201] Wan, X., Karniadakis, G.E.: Multi-Element Generalized Polynomial Chaos for Arbitrary Probability Measures. SIAM Journal on Scientific Computing **28**(3), 901–928 (2006). DOI 10.1137/050627630. URL https://doi.org/10.1137/050627630

[202] Wang, C., Duan, Q., Tong, C.H., Di, Z., Gong, W.: A GUI platform for uncertainty quantification of complex dynamical models. Environmental Modelling & Software **76**, 1 – 12 (2016). DOI https://doi.org/10.1016/j.envsoft.2015.11.004. URL http://www.sciencedirect.com/science/article/pii/S1364815215300955

[203] Wangersky, P.J.: Lotka-Volterra population models. Annual Review of Ecology and Systematics **9**(1), 189–218 (1978)

[204] Weidmann, U.: Transporttechnik der Fussgänger. IVT Schriftenreihe **90** (1992)

[205] Wetterkontor.de: Wetterrückblick Zwiesel (Bayerischer Wald, 612 m). URL https://www.wetterkontor.de/de/wetter/deutschland/rueckblick.asp?id=217&datum=30.04.2013&t=8

[206] Wiener, N.: The homogeneous chaos. Amer. J. Math. **60**(4), 897–936 (1938). URL http://www.jstor.org/stable/2371268

[207] Wilke, C.: Fundamentals of Data Visualization: A Primer on Making Informative and Compelling Figures. O'Reilly Media (2019)

[208] Winokur, J., Kim, D., Bisetti, F., Le Maître, O.P., Knio, O.M.: Sparse Pseudo Spectral Projection Methods with Directional Adaptation for Uncertainty Quantification. Journal of Scientific Computing **68**(2), 596–623 (2016). DOI 10.1007/s10915-015-0153-x. URL https://doi.org/10.1007/s10915-015-0153-x

[209] Wohlrab, B.: Landschaftswasserhaushalt: Wasserkreislauf und Gewässer im ländlichen Raum ; Veränderungen durch Bodennutzung, Wasserbau und Kulturtechnik. Parey (1992)

[210] Xiu, D.: Efficient Collocational Approach for Parametric Uncertainty Analysis. Communications in computational physics **2**(2), 293–309 (2007). URL http://www.researchgate.net/publication/228642363_Efficient_collocational_approach_for_parametric_uncertainty_analysis/file/79e4150b0eb6ebe412.pdf

[211] Xiu, D.: Fast numerical methods for stochastic computations: a review. Communications in computational physics **5**(2), 242–272 (2009)

[212] Xiu, D.: Numerical Methods for Stochastic Computations: A Spectral Method Approach. Princeton University Press (2010)

[213] Xiu, D., Karniadakis, G.E.: THE WIENER–ASKEY POLYNOMIAL CHAOS FOR STOCHASTIC DIFFERENTIAL EQUATIONS. SIAM Journal on Scientific Computing **24**(2), 619–644 (2002). URL `http://epubs.siam.org/doi/abs/10.1137/S1064827501387826`

[214] Xiu, D., Karniadakis, G.E.: Modeling uncertainty in flow simulations via generalized polynomial chaos. Journal of Computational Physics **187**(1), 137–167 (2003). URL `http://linkinghub.elsevier.com/retrieve/pii/S0021999103000925`

List of Figures

List of Tables

Listings