Efficient Sampling and Solver Enhancement for Uncertainty Quantification

Efficient Sampling and Solver Enhancement for Uncertainty Quantification

Yous van Halder

Efficient Sampling and Solver Enhancement for Uncertainty Quantification

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven,
op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens,
voor een commissie aangewezen door het College voor Promoties,
in het openbaar te verdedigen op dinsdag 18 januari 2022 om 16.00 uur

door

Yous van Halder

geboren te Loon op Zand

Dit proefschrift is goedgekeurd door de promotor en co-promotor en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. J.J. Lukkien
promotor:	prof.dr.ir. B. Koren
co-promotor:	dr.ir. B. Sanderse (CWI)
leden:	prof.dr.ir. R.A.W.M. Henkes (TU Delft)
	prof.dr. S. Mishra (ETH Zürich)
	prof.dr.ir. C.W. Oosterlee (U Utrecht)
	prof.dr. F. Toschi
adviseur:	ir. L. Brosset (Gaztransport & Technigaz)

Het onderzoek dat in dit proefschrift wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Efficient Sampling and Solver Enhancement for Uncertainty Quantification

The focus of this thesis is on the development of efficient and robust uncertainty quantification algorithms that can be used for a wide range of applications. The goal is to determine how likely certain outcomes are if some aspects of the application are not known exactly, i.e., in case of limited prior knowledge or uncertainties in the parameter values. We focus in particular on quantifying uncertainties in the output of a mathematical model induced by uncertainties in the input parameters, often referred to as forward propagation of parametric uncertainties.

The first part of this thesis discusses how to significantly reduce the required number of samples needed in the forward propagation of uncertainties. In particular, if the quantity of interest is highly non-linear or discontinuous depending on the input parameters, Gibbs phenomena may occur, which deteriorate the accuracy globally. As we often do not know beforehand if a quantity of interest is smooth or discontinuous, we propose a new algorithm that works in both cases.

In the second part of this thesis, instead of placing samples optimally in order to reduce the overall computational cost of forward propagation, we opt to increase the accuracy of the quantity of interest that is outputted by the solver. To clarify, when using numerical discretisation to compute outcomes of the mathematical model, the resolution of the computational grid determines the accuracy of the quantity of interest, but it also determines the computational time it takes to compute this quantity of interest. In general, a quantity of interest that is computed using a coarse computational grid is fast to compute but also inaccurate. As a remedy, we opt to use machine learning to increase the accuracy of a quantity of interest that is computed on a coarse grid, as it is able to deal with highly non-linear behaviour and high-dimensional input/output relations.

The main topics of this thesis are uncertainty quantification and machine learning. We assume that the reader possesses basic prior knowledge on these two subjects. Furthermore, the test cases discussed in both Part I and II of this thesis assume prior knowledge on basic numerical discretisation techniques, in particular finite volume methods.

Short Biography

Yous van Halder is a data scientist in the field of logistics and finance. Yous got his master's degree in applied mathematics at Eindhoven University of Technology with a special focus on fluid mechanics. During his master's studies, Yous did an internship at Stanford University at the LentinkLab and developed an algorithm to induce turbulence in a wind tunnel using a so-called active turbulence grid.

After his master's studies, Yous started PhD research in the field of uncertainty quantification. During this research he developed algorithms to effectively quantify uncertainties in the output of mathematical models of which the parametric inputs are uncertain.

After four years of PhD research, Yous moved away from academics. As the co-founder of SKIAlabs he aims to bring smart-city logistics to a new level. Apart from his role as co-founder of a company, he has a part-time job at Univé, an insurance company in the Netherlands, where he develops AI models to automate financial processes.

Preface

This PhD-thesis is the result of four years of research at the Centrum Wiskunde & Informatica (CWI) in Amsterdam, as a member of the NWO Perspective Program Sloshing of Liquified Natural Gas (SLING). The focus of this thesis is on the development of efficient and robust uncertainty quantification algorithms that can be used for a wide range of applications. In this thesis we consider in particular fluid flow applications that are inspired by the SLING project.

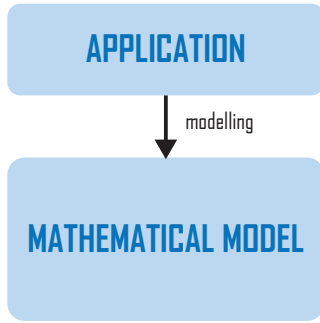
Before I started this PhD project, I was not fully convinced on how relevant uncertainty quantification is in for instance industrial applications. During my PhD-project I noticed that uncertainties are everywhere, and correctly quantifying the effects of these uncertainties is of major importance in many industrial applications.

Introduction

I. 1 Mathematical modelling

Mathematical modelling as a building block for science and engineering, for society in general

Mathematics is omnipresent, often without being noticed. Beit, the transition to renewable fuels, or the fast emerging interest in artificial intelligence, mathematics is one of the tools that allows us to invent and optimise many processes we come across in everyday life. The things we take for granted as humans nowadays, such as flying to our holiday destination inside an Airbus A380, or predicting the weather, would not be possible without the underlying mathematical models. These two applications (and many more) require us to study the flow of liquids and gases, for which the mathematical model is a set of Partial Differential Equations (PDEs) that need to be solved numerically. Computing solutions for PDEs helps us gaining insight in how to operate and/or optimise an industrial process involving fluid flow. In the remainder of this thesis we refer to a mathematical model as a single PDE or a set of PDEs.



Mathematical models are not perfect

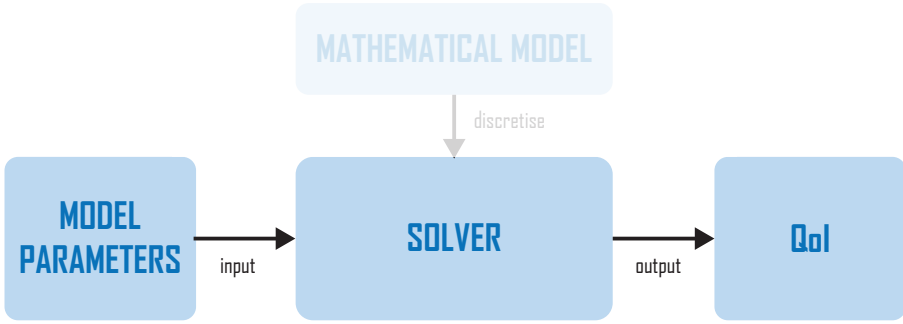
For one given application, multiple mathematical models may be available, that incorporate different amounts of prior knowledge and/or assumptions. Incorporating prior knowledge and/or assumptions into a mathematical model ameliorates the proximity of the mathematical representation to reality. The amount of prior knowledge we can incorporate is often determined by the amount of data/observations that are available to model the underlying physics. Furthermore, even if we are certain that a mathematical model is able to accurately represent reality, the computational power required for obtaining a solution of the model is often a limiting factor. Summarised, it is often impossible to obtain a solution that coincides with reality, is it due to a lack of knowledge to construct a proper model, or due to a lack of computational power to obtain solutions.

Even if the model is perfect, it is not

A mathematical model often possesses several parameters that determine how a physical process behaves. For instance, in fluid dynamics commonly encountered parameters are viscosity, density and surface tension, and they significantly influence how a fluid behaves. Getting the correct parameter values can be difficult and may lead to an error between a solution of a mathematical model and reality.

Solvers for mathematical models to approximate reality

A mathematical model may be solved numerically, using for instance a Finite Element Method (FEM), in order to map a set of model/input parameter values to a quantity in which one is interested. For instance, when studying the flow of air over an airplane, the ***Quantity of Interest (QoI)*** may be the drag coefficient of the wing.



The quality of the solutions is often determined by the fidelity of the solver. In the field of mathematical modelling, fidelity refers to the degree to which a model or simulation reproduces reality, and often determines the computational effort that is required to obtain a numerical solution. To clarify, high-fidelity solutions represent reality closely at a high computational cost, while a low-fidelity solution is less accurate, but requires less computational effort. The fidelity of the model also determines the accuracy of the computed QoIs, but also determines the time it takes to compute a QoI.

I. 2 Uncertainty quantification

Uncertainty Quantification (UQ) is the science of characterisation and reduction of uncertainties and the goal is to determine how likely certain outcomes are if some aspects of the application are not known exactly, i.e., in case of limited prior knowledge or uncertainties in the parameter values. In this thesis we focus on how to quantify uncertainties in the output of a mathematical model induced by uncertainties in the input parameters, often referred to as forward propagation of parametric uncertainties.

Steps in forward UQ

Quantifying the effects of uncertainties in computational engineering typically consists of three steps:

1. Each of the input uncertainties is characterised in terms of a ***Probability Density Function (PDF)***, which follows from observations or physical evidence.

2. The uncertainties are propagated through the model.

3. The output is post-processed; the QoI is expressed in terms of its statistical properties.

In the present work we focus on the (second) propagation step, and the input distributions are assumed to be given.

Non-intrusive sampling methods for forward UQ

For problems which have a complex underlying model, one often uses so-called *non-intrusive* methods for forward propagation of uncertainties through the model, also known as *sampling methods*. The model is solved deterministically a number of times for different parameter values, and a surrogate of the QoI is constructed as a function of the uncertain parameters using these deterministic samples. The number of samples at which the model needs to be evaluated and the time it takes to perform a single sample determine the overall computational cost of the forward propagation of

parameter uncertainties. Hence, in order to increase efficiency of a non-intrusive UQ method, one can either reduce the required number of samples for obtaining statistics on the QoI, or decrease the required computational time for computing a single sample. In this thesis we explore both strategies in more detail and show that both approaches are suited depending on the underlying application.

Why this thesis?

A well known sampling method for propagating uncertainties through a model is the *Monte Carlo* method. Despite its easy implementation and wide applicability, the Monte Carlo method suffers from slow convergence with increasing number of model evaluations, when approximating the QoI. As a consequence of this slow convergence rate, many samples are required for obtaining high quality stochastic solutions. Even though Monte Carlo methods are often superior for high-dimensional problems, proper alternatives have been proposed to speed up convergence in many cases. As an alternative to Monte Carlo methods, other methods were introduced, improving the slow convergence of Monte Carlo significantly. Three popular alternatives to Monte Carlo are:

- **Kriging (Gaussian process regression):** is a method of regression for which the regressed values are modeled by a Gaussian process governed by prior covariances. The advantage of using Kriging is that it automatically quantifies the regression errors (Kriging variance). A disadvantage of Kriging is that it is computationally expensive compared to other local regression methods.

- **Generalised polynomial chaos (gPC):** performs a polynomial expansion of the QoI, where the type of polynomials are determined by the underlying PDF. An advantage of gPC is that it is robust and results in a stable regression of the QoI. A drawback of gPC is its rather poor flexibility, i.e., the sample locations are fixed by the zeros of the chosen polynomial basis, which does not allow for any alteration in the sample locations without constructing a whole new polynomial basis. As a remedy, novel basis construction methods were introduced that increase the flexibility in sample location placement.

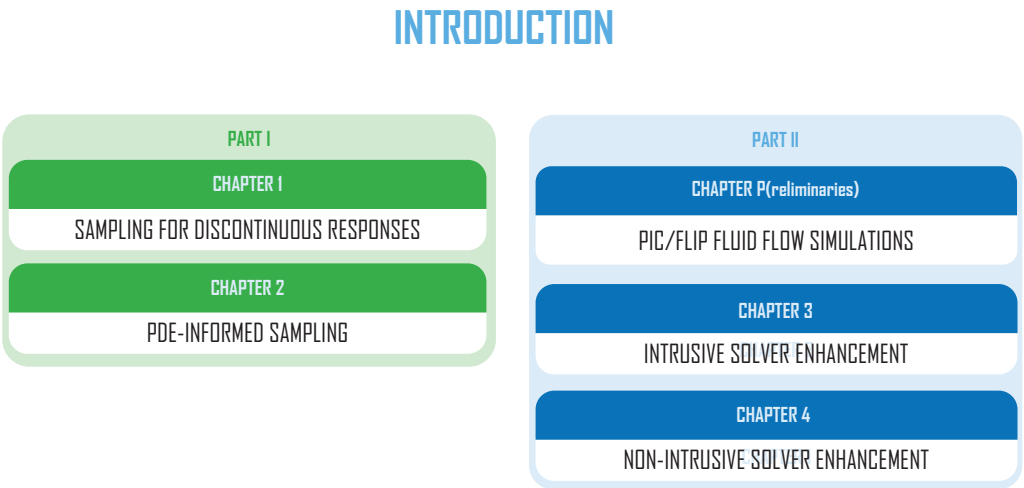
- **Stochastic collocation (SC):** is an interpolation based on Lagrange polynomials. Its primary advantage lies in its ease of implementation. A drawback of the original SC method is that it does not scale well for high-dimensional problems and does not exploit observed behaviour in the QoI.

These are the most common alternatives to Monte Carlo. The introduction of these alternatives resulted in a decrease of required samples to achieve a certain accuracy in comparison to Monte Carlo methods. In this thesis we focus on SC-based approaches, as they are often easy to implement and easy to use.

Reducing the required number of samples by using for instance SC if often not enough. In many fluid dynamics applications the model is extremely expensive to sample from, which makes even SC infeasible, as performing a single sample is time consuming. Furthermore, in many real applications we have to deal with many uncertainties, leading to the so-called ***curse of dimensionality***, which states the number of samples needed to estimate an arbitrary function with a given level of accuracy grows exponentially with respect to the number of input variables of the function. Another complication may occur when the space spanned by the uncertainties is not a hypercube, which does not allow for structurally placing samples and exploiting symmetries. Lastly, it is possible that the QoI exhibits a discontinuous response with respect to an uncertain parameter. These discontinuities significantly complicate surrogate construction and often lead to a significant increase in the required number of samples. ***Efficiently constructing a surrogate for a QoI in case the underlying model is expensive to sample from, a QoI which lives in a (non-hypercube) high-dimensional space and/or exhibits discontinuities, is the aim of this thesis.***

I. 3 Outline of this thesis

This thesis explores the two ways to make forward UQ more efficient and the outline of the thesis is shown below.



Part I: Reducing the required number of samples

The first part of this thesis discusses how to significantly reduce the required number of samples needed in the forward propagation of uncertainties.

In particular, if the QoI is highly non-linear or discontinuous depending on the input parameters, Gibbs phenomena may occur, which deteriorate the accuracy globally. Many alternatives to SC have been proposed that are able to deal with discontinuities in the QoI, but they have the downside that they often lead to inefficient sample placement when used for a smooth QoI. As we often do not know beforehand if a QoI is smooth or discontinuous, we propose a new algorithm that works in both cases.

Another drawback of conventional SC is that it does not take the underlying model into account, i.e., samples are placed solely based on the characterisation of the input parameters in terms of a PDF. Adaptive sampling strategies have been proposed that use information about previously sampled QoI values in order to determine the optimal set of input parameter values for which to sample the model next. We propose a new adaptive sampling strategy that incorporates information about both the underlying model PDE and PDF of the input parameters, to effectively sample the QoI.

Part II: Increasing the accuracy of inexpensive, but inaccurate solutions

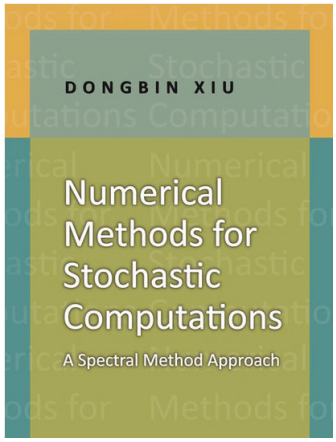
Instead of placing samples optimally in order to reduce the overall computational cost of forward UQ, we also opt to increase the accuracy of the QoI that is outputted by the solver. To clarify, when using numerical discretisation to compute outcomes of the mathematical model, the resolution of the computational grid determines the accuracy of the QoI, but it also determines the computational time it takes to compute this QoI. In general, a QoI that is computed using a coarse computational grid is often fast to compute but also inaccurate. As a remedy, we opt to use *machine learning* to increase the accuracy of a QoI that is computed on a coarse grid, as it is able to deal with highly non-linear behaviour and high-dimensional input/output relations.

We first study how machine learning can be used as a non-intrusive post-processing tool to increase the accuracy of the QoI. This non-intrusive approach is flexible as it does not require to change the underlying numerical solver that is used to compute the QoI, but also has limitations in terms of improper treatment of the temporal dimension when the QoI is time dependent, as the non-intrusive approach is only able to enhance the QoI at a given time instance. As a result, we need to retrain our machine learning algorithm when wanting to enhance the QoI at a different time instance. Therefore, also an intrusive alternative is proposed that uses machine learning inside the numerical solver in order to increase the accuracy of solver quantities at every time step. We will focus in particular on the enhancement of a specific numerical fluid flow solver, i.e., PIC/FLIP. This solver is introduced in detail in the preliminary section of this part.

I. 4 What prior knowledge do you need

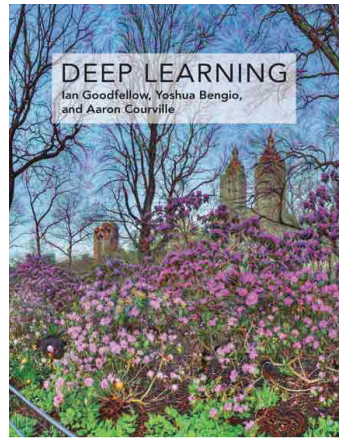
The main topics of this thesis are UQ and machine learning. We assume that the reader possesses basic prior knowledge on these two subjects. Furthermore, the test cases discussed in both Part I and II assume prior knowledge on basic numerical discretisation techniques, in particular finite volume methods. Good references for the assumed prior knowledge are shown below.

UNCERTAINTY QUANTIFICATION



Dongbin Xiu, Numerical Methods for Stochastic Computations, 2010, Princeton Univerity Press

MACHINE LEARNING



Ian Goodfellow et al., Deep Learning, 2016, MIT Press

Part I

efficient sampling

As stated before, for problems which have a complex underlying model, one often uses so-called *non-intrusive* methods, also known as *sampling* methods. The model is solved deterministically a number of times, and a surrogate for the QoI is constructed using these deterministic samples. In this part of the thesis we focus on minimising the required number of samples when constructing a surrogate model. In mathematical form, the goal is to solve the following stochastic problem:

$$\mathscr{L}(v;\mathbf{z}) = 0\ . \tag{P. 1}$$

We define $I_{\mathbf{z}}$ as the support set of the uncertain inputs $\mathbf{z}$, referred to as the *random space*, $\mathbf{z} = (z_1,...,z_d) \in I_{\mathbf{z}}$ the d-dimensional vector containing uncertain inputs, $v = v(\mathbf{z})$ the solution and $\mathscr{L}$ an operator, representing the model. The operator $\mathscr{L}$ can be a non-linear partial differential operator, or any sufficiently smooth mathematical model that relates input $\mathbf{z}$ to the solution $v(\mathbf{z})$. The QoI q is calculated by applying an operator $\mathscr{Q}$ to the solution v, i.e., $q(\mathbf{z}) = \mathscr{Q}(v(\mathbf{z}))$. The uncertain inputs are assumed to be characterised by a joint PDF $\rho(\mathbf{z})$. The stochastic problem is solved non-intrusively by sampling the model (P. 1) at different locations $\mathbf{z}_i$ in a random space, i.e.:

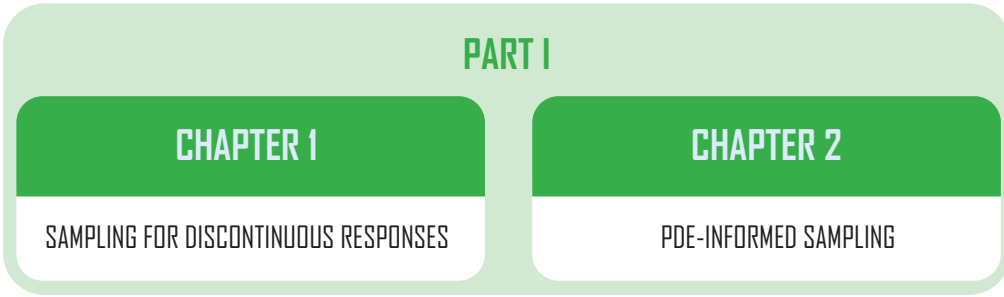
$$\mathscr{L}(v_i;\mathbf{z}_i) = 0 \ \Rightarrow q_i = q(\mathbf{z}_i) = \mathscr{Q}(v_i)\ , \tag{P. 2}$$

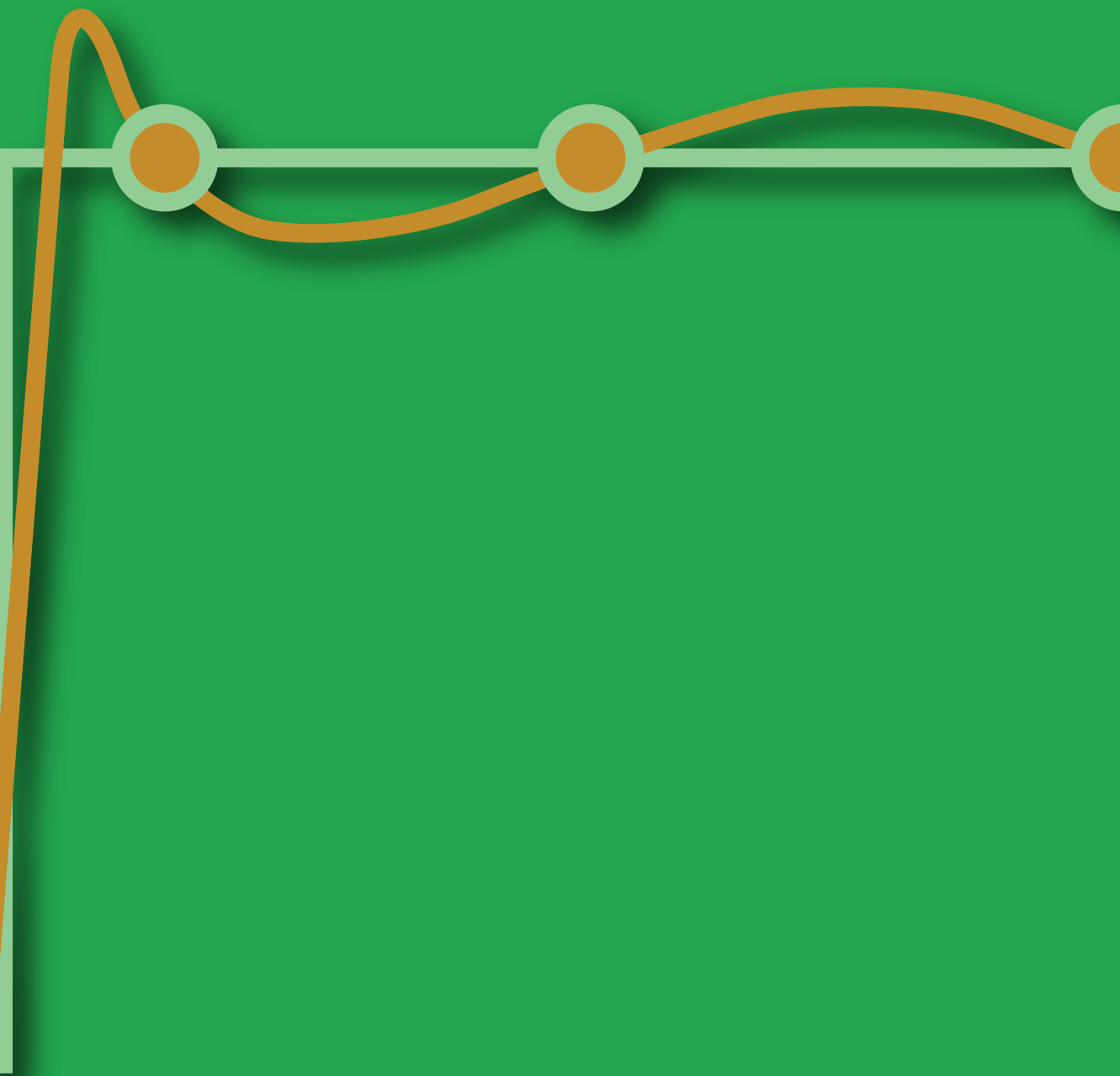
where $v_i = v(\mathbf{z}_i)$ and $q_i = q(\mathbf{z}_i)$ are the sampled solution and QoI at the collocation node $\mathbf{z}_i$, respectively. Since the sampling is non-intrusive, black-box solvers for the operator $\mathscr{L}$ can be used. We are interested in finding the entire functional relation q as a function of the uncertainties $\mathbf{z}$, in terms of a surrogate model. Such a surrogate allows for fast sampling of the QoI in the random space and removes the high computational burden once the surrogate is constructed. A surrogate model $\tilde{q}$ of q may be constructed by interpolation, such that:

$$\tilde{q}(\mathbf{z}) \approx q(\mathbf{z}), \quad \text{for all} \quad \mathbf{z} \in I_{\mathbf{z}}\ . \tag{P. 3}$$

Effectively choosing your sample locations $\mathbf{z}_i$ to reduce the required number of samples is the goal of Part I. In particular, we will focus on effectively choosing sample locations $\mathbf{z}_i$ when the QoI shows a discontinuous response in Chapter 1. Efficiently choosing sample locations by incorporating knowledge of the underlying PDE is discussed in Chapter 2.

An overview of this part of the thesis is given below.





Sampling for discontinuous responses

1

Sampling for discontinuous responses

An adaptive minimum spanning tree multielement method for uncertainty quantification of smooth and discontinuous responses

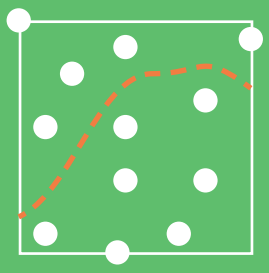
A novel approach for non-intrusive uncertainty propagation is proposed. Our approach overcomes the limitation of many traditional methods, such as generalised polynomial chaos methods, which may lack sufficient accuracy when the QoI depends discontinuously on the input parameters. As a remedy we propose an adaptive sampling algorithm based on minimum spanning trees combined with a domain decomposition method based on support vector machines. The minimum spanning tree determines new sample locations based on both the PDF of the input parameters and the gradient in the QoI. The support vector machine efficiently decomposes the random space in multiple elements, avoiding the appearance of Gibbs phenomena near discontinuities. On each element, local approximations are constructed by means of least orthogonal interpolation, in order to produce stable interpolation on the unstructured sample set. The resulting minimum spanning tree multi-element method does not require initial knowledge of the behaviour of the QoI and automatically detects whether discontinuities are present.

Background

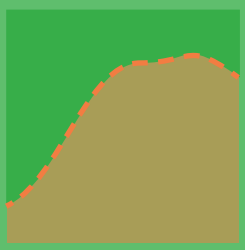
As stated in the introduction, the goal of part I is to reduce the required number of samples when constructing an accurate surrogate for a QoI that is complicated by either possessing discontinuities or lives in a high-dimensional space. In this chapter we are discussing how to effectively construct a surrogate for a QoI in the presence of discontinuities. When propagating input uncertainties through a computational model it is often assumed that the QoI has a smooth response. However, if the QoI is highly non-linear or discontinuous, *Gibbs phenomena* [1] may occur, which may deteriorate the accuracy of a sampling approach globally. To avoid the occurrence of Gibbs phenomena, several methods were introduced [2, 3], but they focus solely on discontinuous QoIs, leading to a significant increase in the number of samples needed for approximating smooth QoIs. One interesting method is the *multi-element stochastic collocation* method [4, 5]. The idea of multi-element stochastic collocation is to decompose the domain, spanned by the uncertainties, into smaller non-overlapping elements, in each of which the QoI is amenable for using a stochastic collocation method. Gibbs phenomena still appear in the elements where there is a discontinuity in the QoI, but they are confined to these specific elements. Improving the multi-element approach is an active field of research and focuses on more efficient and robust domain decomposition. Jakeman et al. [6] proposed the minimal multi-element method, which uses discontinuity detection based on polynomial annihilation to detect discontinuities, and divides the domain along the discontinuities. As a result, the Gibbs phenomena are removed completely. Although this discontinuity detection algorithm is accurate, the total number of samples needed to determine the discontinuity location may easily be still too high if sampling the model is expensive. Therefore, a discontinuity detection algorithm which performs well for a lower number of samples was proposed by Gorodetsky et al. [7]. This discontinuity detection uses polynomial annihilation in combination with *Support Vector Machines* (SVMs) to divide the domain into elements along the so-called SVM classification boundary. Even though both discontinuity detection algorithms [6, 7] perform well, using them for approximating smooth QoIs can become prohibitively expensive, as both methods focus solely on finding the discontinuities. It is often unknown in advance if a QoI is smooth or discontinuous.

Published in SIAM Journal on Scientific Computing

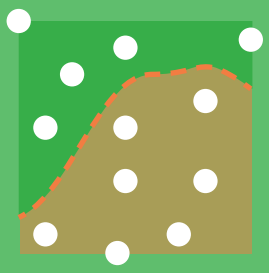
Our Approach



sample the random space to detect and approximate the discontinuity manifold



split into multiple elements by cutting along the approximated discontinuity



interpolate sample values on each element to get multiple local approximations

The highlights of the new approach are:

- *Parametric solutions for PDEs can be efficiently constructed using the proposed method;*
- *The proposed method automatically detects if a quantity of interest is smooth or discontinuous, so no a-priori knowledge of the quantity of interest is required;*
- *A supervised learning algorithm detects the location of discontinuities when present;*
- *Minimum spanning trees refine the sample based on solution gradients and probability density functions.*

1 Preliminaries

As mentioned before, we only consider non-intrusive approaches, which approximate $q(\mathbf{z})$ in (P.3) by using a black-box to sample point values $q(\mathbf{z}_i)$ as in (P.2). These point values are then used to create a surrogate model $\tilde{q}(\mathbf{z})$ for the QoI. When q is smooth, it is possible to construct a surrogate $\tilde{q}$ which converges exponentially fast to the exact QoI q. However, if the QoI exhibits highly non-linear or discontinuous behaviour, then the accuracy of the surrogate deteriorates globally, due to Gibbs phenomena. Multi-element methods divide the random space $I_{\mathbf{z}}$ into a set of N_E smaller elements E_i, such that the negative impact of the Gibbs phenomena is confined to a limited number of elements surrounding the discontinuity. The elements E_i are non-overlapping and span the entire random space, i.e.:

$$\cup_{i=1}^{N_E} E_i = I_{\mathbf{z}} \quad \text{and} \quad E_i \cap E_j = \emptyset \,, \quad \text{if} \quad i \neq j \,. \tag{1.1}$$

A local surrogate $\tilde{q}^{(i)}$ is constructed in each E_i:

$$\tilde{q}^{(i)}(\mathbf{z}) \approx q(\mathbf{z}), \quad \text{for all} \quad \mathbf{z} \in E_i \,. \tag{1.2}$$

The global surrogate is given by patching the local surrogate models:

$$\tilde{q}(\mathbf{z}) = \sum_{i=1}^{N_E} \tilde{q}^{(i)}(\mathbf{z}) \mathscr{I}_{E_i}(\mathbf{z}) \,, \tag{1.3}$$

where $\mathscr{I}_{E_i}(\mathbf{z})$ is the indicator function satisfying $\mathscr{I}_{E_i}(\mathbf{z}) = 1$, if $\mathbf{z} \in E_i$ and 0 otherwise. Standard multi-element methods utilise a tensor construction of hypercubes for defining the elements E_i [4, 5]. While such a tensor construction removes the global effect of Gibbs phenomena, they can still appear locally in several elements.

An example of the standard tensorised multi-element approach for the approximation of a 2D function is shown in figure 1.1. Gibbs phenomena appear in the elements $E_{3,4}$, where a discontinuity is present in the exact function.

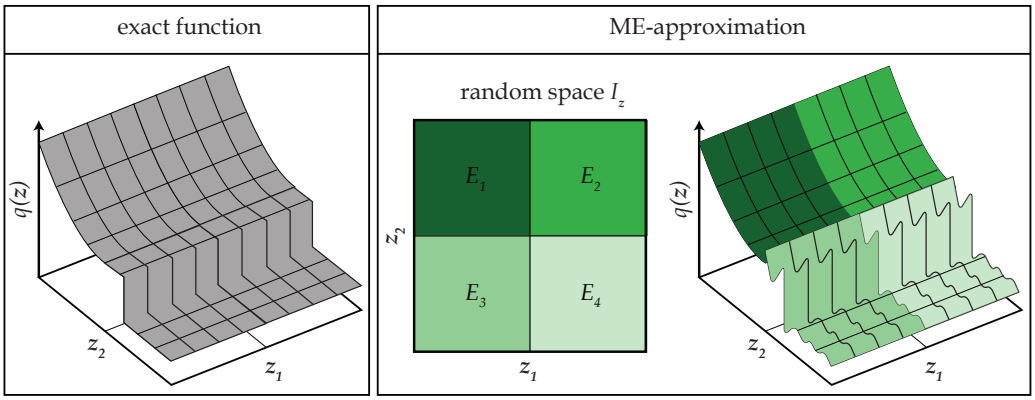


Figure 1.1: *Example of the standard multi-element approach for approximating a 2D function exhibiting a discontinuity.*

2 Our approach in short

The goal of this chapter is to construct an efficient multi-element method that accounts for the possible presence of discontinuities. It is often not known beforehand if the QoI comprises discon-

tinuities and therefore in order for an approach to be robust, our method needs to be able to deal with both smooth and discontinuous responses.

The proposed approach works for both smooth and discontinuous QoIs and requires no initial knowledge about the QoI. A novel domain decomposition method in combination with adaptive sampling of the QoI is therefore proposed for constructing this surrogate. The adaptive sampling procedure in our method is based on Minimum Spanning Trees (MST) [8, 9], which add new sample points at places which are associated with a high PDF and/or where the QoI changes rapidly. The adaptively placed samples are classified and an SVM [7, 10, 11, 12, 13] is used to obtain a classification boundary, which serves as an approximation for the discontinuity location. The decomposition of the random space in this way leads to elements on which each local QoI is amenable for interpolation without Gibbs phenomena. For constructing a surrogate model in each element a least orthogonal interpolant [14] is employed, which is suited for interpolation on the scattered data points that we obtain with our adaptive sampling. Our proposed method is abbreviated as MST-ME (Minimum Spanning Tree Multi-Element) method and is designed mainly for the purpose of uncertainty propagation. However, when assuming uniformly distributed PDFs for the input parameters, the MST-ME method is also well suited to obtain a parametric solution of the PDE under consideration. Although the MST-ME method is designed to work in any number of dimensions, very high-dimensional problems are not the main focus of our work, but rather the accurate treatment of discontinuities.

3 Minimum spanning tree multi-element method

The art of constructing an efficient multi-element stochastic collocation method lies in the choice of sampling points $\mathbf{z}_i$, the choice of the elements E_i, and the reconstruction of the local approximations $\tilde{q}^{(i)}$. These are the foci of this method. Accordingly, the MST-ME method introduced here, is divided into three stages: (I) sampling, (II) domain decomposition and (III) local approximation construction:

I ***choice of*** z_i: adaptive sampling of the QoI while taking into account both smooth and discontinuous regions, and the underlying PDF.

II ***choice of*** E_i: division of the random space into a minimal number of elements, such that the QoI is smooth within each element.

III ***construction of*** $\tilde{q}^{(i)}$: interpolation of the samples, while producing a stable interpolant.

The QoI is adaptively sampled (I) by taking into account both the PDF and the QoI gradient information. The samples are distributed among different classes, such that within each class the local QoI is smooth. The classified samples are given as input to the domain classification algorithm (II). Instead of using a tensor-based domain decomposition, the random space is divided into a minimal number of elements, in which the QoI is amenable for interpolation without Gibbs phenomena. This domain decomposition methodology was already introduced in [6], but in that work the number of samples required for determining proper elements was too high. In contrast, our domain decomposition method (II) uses the samples from the sampling algorithm (I) for determining proper elements, without the need to perform additional sampling. Local approximations (III) are constructed in each element, by using the least orthogonal interpolation method [14]. The global approximation, the surrogate model, is given by the patched local approximations (1.3).

I Sampling algorithm

The main idea behind our adaptive sampling algorithm is that we want to refine our surrogate model based on the QoI behaviour and the associated joint PDF of the random input variables. We achieve this by creating a graph that links the samples, and by assigning weights to edges of this graph, constructing an MST, and then adding samples on the most important edges of this MST.

The methodology is explained for a 2D QoI $q(z_1, z_2)$, but can be generalised easily to a higher dimensional QoI.

Initial sample placement

The sampling procedure is started by placing initial sample points. Straightforward choices of the initial sample locations are shown in figure 1.2. These initial sample configurations are extensible to high-dimensional random spaces. Initial sample placement in both figure 1.2a and figure 1.2b introduces an anisotropy in the placement of subsequent samples. The initial sample placement in figure 1.2c is a good trade-off between the number of initial samples and the isotropy of subsequent samples.

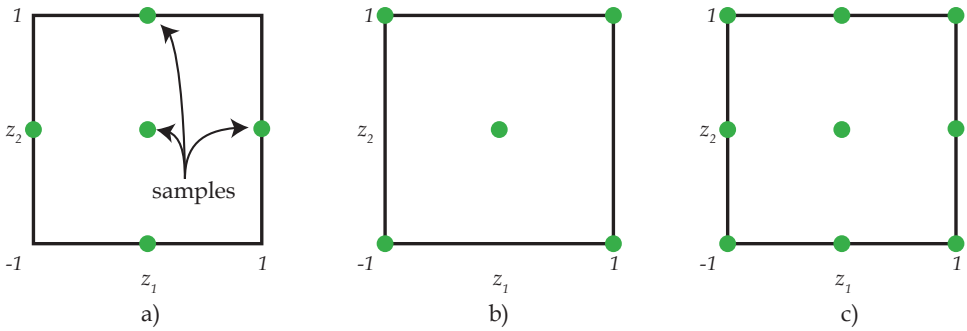


Figure 1.2: *Initial sample location configurations. a) One sample in the middle and one sample at each face centre. b) One sample in the middle and one sample at each corner of the domain. c) The combination of a) and b).*

The initial sample grid shown in figure 1.2c is employed, unless stated otherwise.

Graph construction with neighbouring samples

The existing samples are connected based on Voronoi diagram construction [15]. A Voronoi diagram is a partitioning of a d-dimensional space into regions based on the distance to a specific set of samples [16]. Each Voronoi cell contains one sample, and the cell corresponds to all the points that are closer to this sample than to any other sample. The Voronoi diagram for the initial sample configuration is shown in figure 1.3 (left).

The boundary of each Voronoi cell contains several vertices. At each of these vertices the neighbouring Voronoi cells are determined. The midpoints of these neighbouring cells are then connected to obtain a local graph, see figure 1.3 (middle). This local graph is similar to a Delaunay triangulation, but gives more isotropic behaviour on regular grids. The local graphs are subsequently connected together to obtain a global graph, see figure 1.3 (right).

Assignment of weights to edges

Not all the edges in the total graph are equally attractive for placing new samples. Edges that are long, and have large variation between QoI values and/or are in a region associated with a high PDF value, are good candidates for refinement. A weight function $w(\mathbf{z}_1, \mathbf{z}_2)$ assigns a weight to the edge

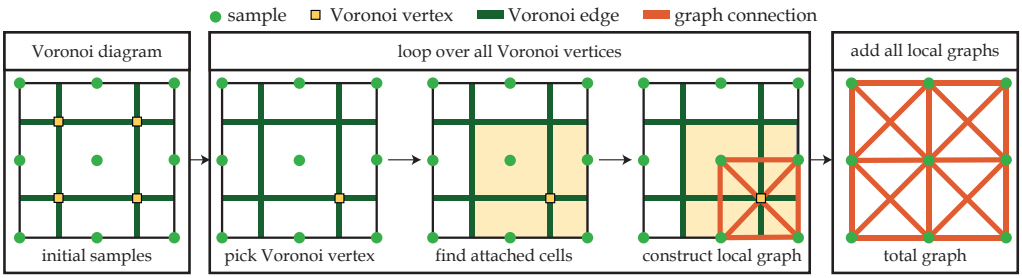


Figure 1.3: *Algorithm for constructing the total graph of a set of sample points. (left) The Voronoi diagram with the corresponding Voronoi vertices. (middle) The local graphs connect the sample points of which the Voronoi cells have a common Voronoi vertex. (right) The total graph combines all the local graphs.*

between the two graph vertices $\mathbf{z}_1$ and $\mathbf{z}_2$. A low weight means that the edge is a good candidate for refinement and vice versa. Weighing based solely on either PDF [17, 18] or gradient [6] is the most straightforward:

$$w_{\text{PDF}}(\mathbf{z}_1,\mathbf{z}_2) = \left(\rho\left(\frac{\mathbf{z}_1+\mathbf{z}_2}{2}\right)||\mathbf{z}_1-\mathbf{z}_2||_2\right)^{-1}, \tag{1.4}$$

$$w_{\text{grad}}(\mathbf{z}_1,\mathbf{z}_2) = \left(\frac{|q(\mathbf{z}_1)-q(\mathbf{z}_2)|}{||\mathbf{z}_1-\mathbf{z}_2||_2}||\mathbf{z}_1-\mathbf{z}_2||_2\right)^{-1}, \tag{1.5}$$

where ρ is the PDF and $||\cdot||_2$ the Euclidean distance. These weight functions are not always efficient, as they place most samples in either the smooth or the discontinuous regions. A PDF-weighed gradient across an edge is more meaningful, as it incorporates both the rate of change of response values and the probability of occurrence. Therefore, when compared to (1.4)-(1.5), a more meaningful measure would be:

$$\left(\int_e |\mathbf{e}\cdot\nabla q(\mathbf{z})|\rho(\mathbf{z})d\mathbf{z}\right)^{-1}, \tag{1.6}$$

where $\mathbf{e}$ is a vector representing an edge. Equation (1.6) can be seen as the reciprocal of the expectation of the gradient across an edge. The expectation of the gradient is an important quantity in stochastic optimisation routines such as stochastic gradient descent [19], where it justifies the use of a subset of samples for determining the gradient, which is used for minimising the objective function. In our case the gradient is not used for finding a descent direction, but it indicates the amount of local variation in the QoI q. By weighing the gradient with the PDF and integrating it over an edge, we get the expected variation across an edge for the given PDF, which indicates if refinement is necessary. The integral in (1.6) can not be computed directly, but can be approximated as:

$$w_{\text{PDF+grad}}(\mathbf{z}_1,\mathbf{z}_2) = \left(\left(\rho\left(\frac{\mathbf{z}_1+\mathbf{z}_2}{2}\right)\frac{|q(\mathbf{z}_1)-q(\mathbf{z}_2)|}{||\mathbf{z}_1-\mathbf{z}_2||_2}\right)||\mathbf{z}_1-\mathbf{z}_2||_2\right)^{-1}. \tag{1.7}$$

To clarify, we multiply each weight function with the distance of the edge $||\mathbf{z}_1-\mathbf{z}_2||_2$ to account for regions in the random space that have a low number of samples. After weighing all the edges, we normalise by dividing by the maximum weight. The different weight functions are compared in section 6.

Minimum spanning tree for refinement of sample grid

New samples are placed at the middles of the edges that have a sufficiently low weight. However,

if all edges with sufficiently low weight are refined, undesirable clustering of samples may occur at early stages of the sampling procedure.

To prevent this, a Minimum Spanning Tree (MST) is used to obtain a subset of edges, such that this subset reaches all the samples, with a minimal total edge weight. The most important edges are contained in this MST, while still exploring a significant portion of the random space. The MST prevents the undesirable sample clustering at early stages. An edge in the MST is refined, if its weight is sufficiently low compared to the minimum weight among all edges, $w_{\min}$:

$$w_i \leq c\, w_{\min} \,, \tag{1.8}$$

where $c > 1$. The value of c determines how many samples are added each iteration, i.e., low values of c result in a low number of samples added and vice versa. Low values of c produce the most accurate results, but many iterations are needed to reach a specified total number of samples. Iterations can become prohibitively expensive in high-dimensional random spaces. Therefore, we set $c = 2$, which is a trade-off between the number of samples added and the number of iterations to be performed. The edge with minimum weight $w_{\min}$ is not necessarily included in the MST, and will be added if it was not already included, to prevent the sampling algorithm from not adding any samples.

Complete sampling algorithm

The complete sampling strategy (I) is an iterative procedure, which is illustrated in figure 1.4. The procedure starts with choosing the initial sample points. Next we loop over the three steps: *graph construction*, *edge weighing* and *MST edge refinement*. The loop is terminated when the specified number of iterations $i_{\max}$ has been performed or when the total number of sample points exceeds a specified threshold $N > N_{\max}$.

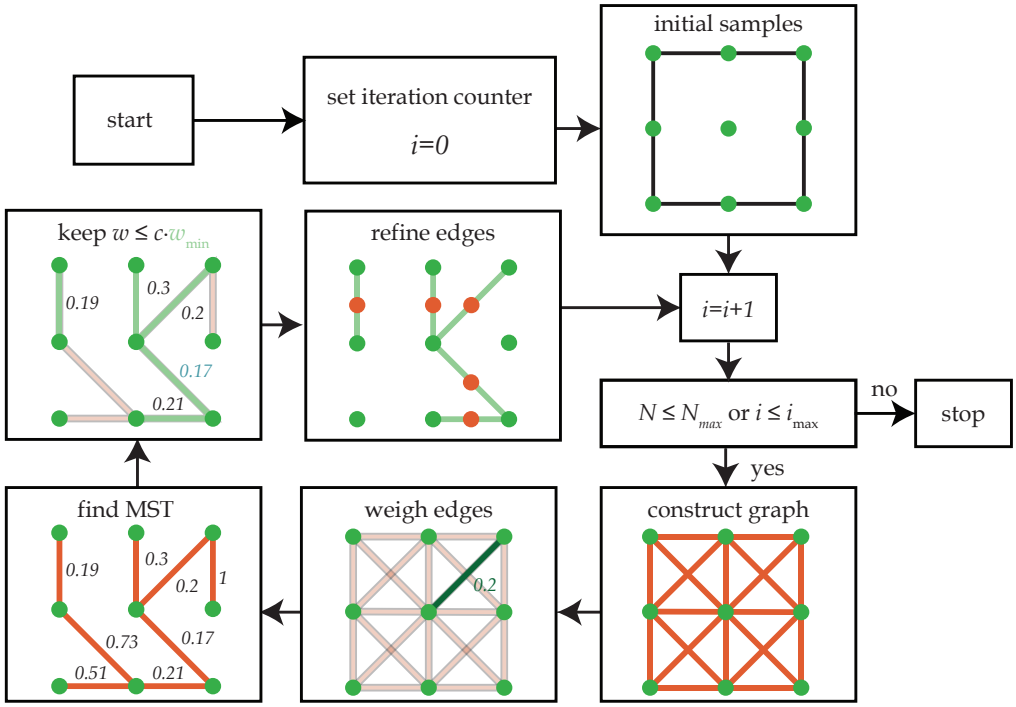


Figure 1.4: *Schematic representation of the adaptive sampling strategy.*

II Domain decomposition

The idea of the domain decomposition step in our method is to divide the random space into non-intersecting elements E_i, such that the sampled QoI values from section I exhibit smooth behaviour locally in each element. The elements are constructed by first classifying the QoI values according to the QoI gradients. Second, the sample classes are separated by means of a classification boundary, based on SVMs. This classification boundary cuts the random space into several elements.

Sample classification based on QoI gradients

An SVM determines a classification boundary based on a set of classified samples. Since a classification boundary is an approximation to a discontinuity in the QoI, the classification is based on the difference in values between two neighbouring samples. To clarify, samples are put into different groups if the jump in values across an edge exceeds a threshold. The choice for the threshold in the procedure is crucial. We have observed that the polynomial annihilation procedure from [7] works very well and a similar methodology is therefore employed. Polynomial annihilation, using all samples close to a straight line crossing an edge, is used to estimate a jump value across an edge and labels two points as the same class if the difference in function values is less than the jump value. An in-depth discussion of polynomial annihilation is discussed in [20]. A schematic representation of the classification procedure is shown in figure 1.5.

The classification procedure shown in figure 1.5 is able to detect multiple discontinuities and able to divide the samples in multiple classes by using only the sample locations and corresponding function values. There is no need for specifying the number of classes beforehand, as the procedure detects the total number of classes automatically. Even though the classification procedure works in many cases, it is not able to consistently detect discontinuities which do not divide the random space into multiple sub-domains. For this reason, we do not consider such discontinuities in the remainder of this manuscript.

Classification boundary from support vector machines

The classes assigned to the different samples in the random space will be used as training data for a supervised machine learning algorithm. SVM is a supervised machine learning technique, used for building a classification boundary between samples that belong to different classes [7, 13]. SVM is used because it is defined by a convex optimisation problem, for which efficient methods are available [21], which makes it viable for high-dimensional problems.

Assume N adaptive samples $Z = \{\mathbf{z}_i\}_{i=1}^{N}$ are classified into N_c different classes c_i, where $c_i \in \{1, \ldots N_c\}$ is the class belonging to sample $\mathbf{z}_i$. The idea behind an SVM is to construct a classifier S_λ of the form:

$$S_\lambda(\mathbf{z}) = \sum_{i \in SV} \alpha_i K(\mathbf{z}, \mathbf{z}_i) \,, \tag{1.9}$$

where α_i is the coefficient associated with the sample point $\mathbf{z}_i$, λ a regularisation parameter, $SV \subset Z$ the set of support vectors and K a kernel [12]. If $\alpha_i > 0$, then $\mathbf{z}_i$ is a support vector. Depending on the application, different kernels are available [13]:

$$K(\mathbf{x}, \mathbf{y}) = <\mathbf{x}, \mathbf{y}> \quad \text{(linear)} \,, \tag{1.10}$$

$$K(\mathbf{x}, \mathbf{y}) = (\gamma <\mathbf{x}, \mathbf{y}> + c_t)^r \quad \text{(polynomial)} \,, \tag{1.11}$$

$$K(\mathbf{x}, \mathbf{y}) = \exp(-\gamma ||\mathbf{x} - \mathbf{y}||_2^2) \quad \text{(radial basis function)} \,, \tag{1.12}$$

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\gamma <\mathbf{x}, \mathbf{y}> + c_t) \quad \text{(sigmoid)} \,, \tag{1.13}$$

where $<\cdot, \cdot>$ is the standard inner product in $\mathbb{R}^d$, γ a regularisation constant, c_t a translation constant and r the polynomial degree. Choosing the proper kernel depends on the regularity of the classification boundary which is often not known beforehand. Radial basis function kernels (1.12)

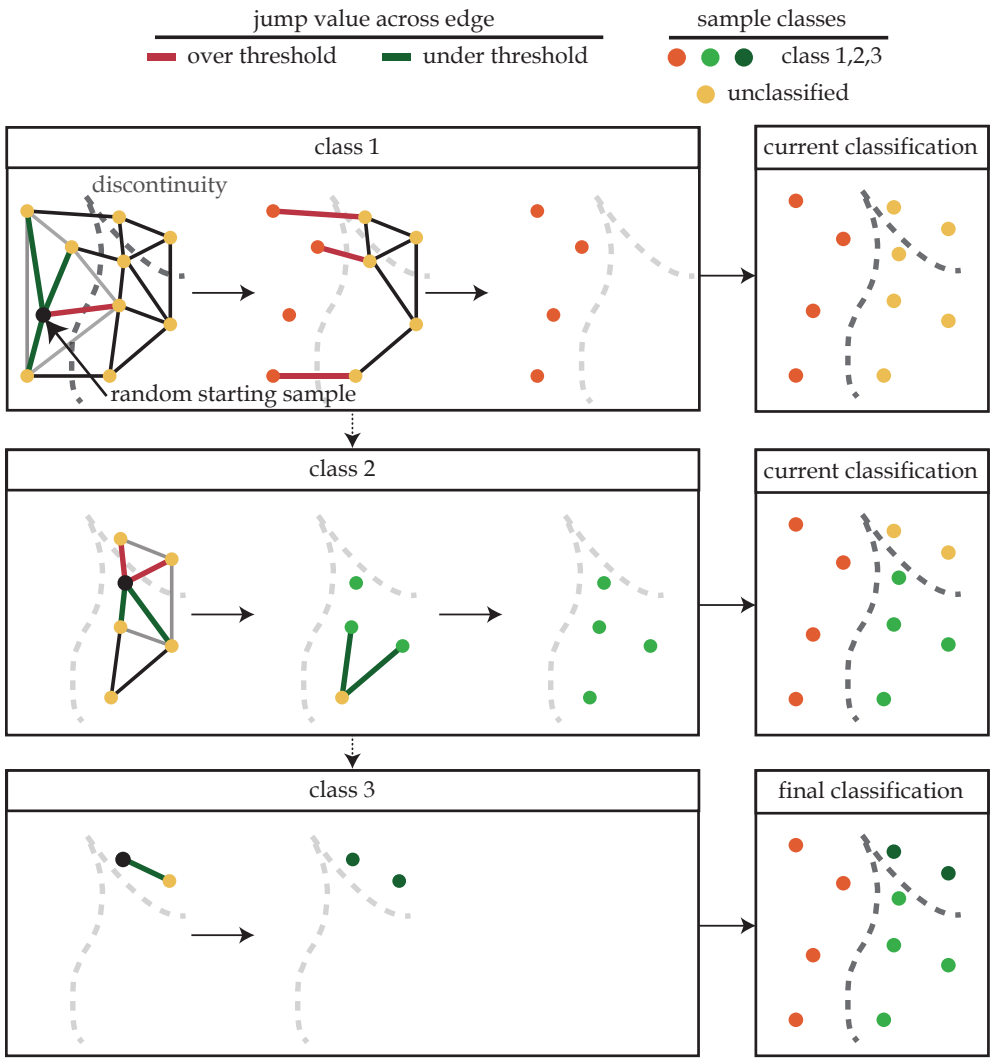


Figure 1.5: *Schematic representation of the classification procedure. The sample connectivity is given by the Voronoi construction performed on the final sampling grid from the adaptive sampling algorithm.*

are a common choice and are also employed. The constant γ is normally advised to be chosen as $1/N_c$ [21], but we will also investigate other choices in the results section. High values for γ result in a classification boundary with a fine resolution, but this may result in overfitting. Low values for γ result in a coarser estimation of the location of the classification boundary. The optimal value for γ differs for different functions. In section 6 an optimal value for γ is found for a specific family of functions, which is important in our test-cases. Once K has been chosen, the classifier S_λ is obtained by solving the following least-squares problem:

$$S_\lambda(\mathbf{z}) = \arg\min_{S \in L_2(I_\mathbf{z})} \left\{ \frac{1}{N} \sum_{i=1}^{N} \max(0, 1 - c_i S(\mathbf{z}_i)) + \lambda ||S||_{L_2(I_\mathbf{z})} \right\} . \tag{1.14}$$

The classification boundary is given by the 0-contour of S_λ. It separates the different classes from each other with a hypersurface and is obtained with the LIBSVM library [21]. The classification

boundary decomposes the domain into several elements. Figure 1.6 shows an example of a classification boundary for two different classes. SVMs can deal with multiple classes, and hence multiple discontinuities, quite easily, which makes it a suitable discontinuity finder for a wide range of QoIs. However, SVMs in combination with the classification procedure are not yet able to properly detect discontinuities that have at least one endpoint which is not intersecting with itself or with the boundaries of the random space.

Gorodetsky et al. [7] show how SVMs can be used to efficiently localise discontinuities in a high-dimensional space. Their approach differs from our approach in the strategy for placing samples which are used by the SVM. They propose sample placement which is solely suited for detecting discontinuities and samples are placed without taking the PDF into account. As opposed to this, our approach incorporates the PDF and makes our sample placement more suitable for surrogate construction and statistical moment calculations, while being slightly less efficient in localising discontinuities.

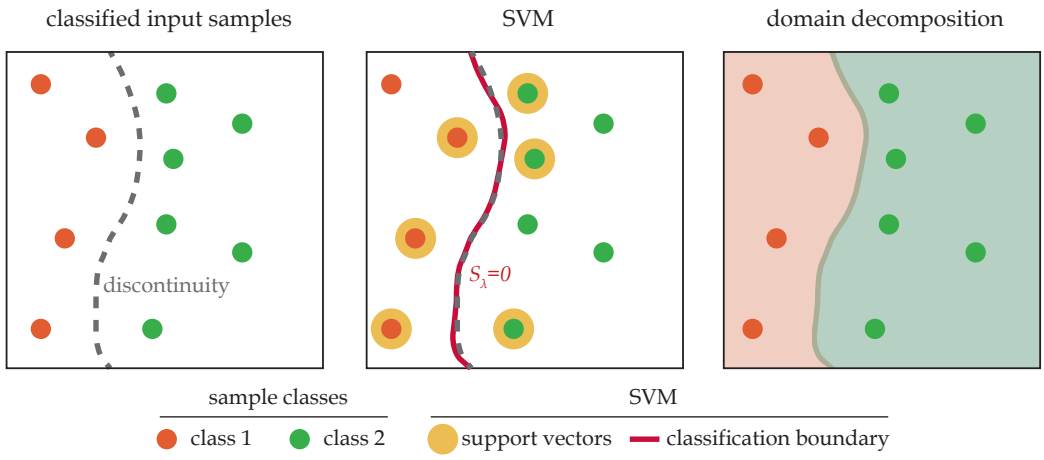


Figure 1.6: *An example of an SVM domain decomposition for two different classes.*

III Local approximations

The elements E_i from the domain decomposition (II) are arbitrarily shaped and the samples $\mathbf{z}_i$ are distributed in such a way that interpolation is not a trivial task. Least orthogonal interpolation is able to perform interpolation on sample distributions on such arbitrarily shaped domains. The sampling strategy (I) does not necessarily choose points that are optimal for interpolation. Therefore, attempting to construct an interpolant on this set of interpolation nodes is not always a good idea and may produce unstable interpolants. We therefore use an extended version of the original least orthogonal interpolation, which selects a subset of samples that is better suited for stable interpolation [6]. This enables the MST-ME method to place sample points in the random space, where we want to further resolve the QoI, without focusing on the stability of the interpolation. In practice, the least orthogonal interpolation can become unstable when an element has a highly irregular shape and in such cases we propose that a different interpolation procedure should be used [22, 23]. In this work we use the least orthogonal interpolation in the remainder of this manuscript because of its fast convergence and flexibility for scattered data sets, and we do not consider highly irregular shapes.

Least orthogonal interpolation

We briefly introduce the least orthogonal interpolation procedure, which is discussed in more detail in [14]. Assume we have sampled our model response at the unstructured locations $\{\mathbf{z}_i\}_{i=1}^N$ in one element $E \in \mathbb{R}^d$ of the SVM domain decomposition. Assume a pdf $\rho(\mathbf{z})$ on E, with corresponding orthonormal polynomials $\{\phi_i(\mathbf{z})\}_{i=1}^\infty$. The orthonormal polynomials are ordered with natural numbers according to the graded reverse lexicographic ordering. We order the orthonormal polynomials by using a multi-index $\mathbf{i} \in \mathbb{N}^d$. Now let Π be the space of all d-variate polynomials and let Π_k be the space of all d-variate polynomials of degree less than or equal to k. For any $q \in \Pi$, define the projection P_k onto Π_k, as

$$q = \sum_{\mathbf{i}} \hat{q}_{\mathbf{i}} \phi_{\mathbf{i}} \Leftrightarrow P_k q = \sum_{||\mathbf{i}|| \leq k} \hat{q}_{\mathbf{i}} \phi_{\mathbf{i}} \ , \tag{1.15}$$

where the expansion coefficients $\hat{q}_{\mathbf{i}}$ are given by

$$\hat{q}_{\mathbf{i}} = < q, \phi_{\mathbf{i}} >_\rho = \int_E q(\mathbf{z}) \phi_{\mathbf{i}}(\mathbf{z}) \rho(\mathbf{z}) d\mathbf{z} \ . \tag{1.16}$$

The goal of Least Orthogonal Interpolation is now: Given the nodes $Z = \{\mathbf{z}_i\}_{i=1}^N$, define a polynomial interpolation space $\Pi_{Z,\rho}$ in which we can uniquely perform interpolation on the sample responses $\{q(\mathbf{z}_i)\}_{i=1}^N$.

To accomplish this, we use the sample locations to define a new set of functions:

$$h_n(\cdot) = \sum_{\mathbf{i}} \phi_{\mathbf{i}}(\mathbf{z}_j) \phi_{\mathbf{i}}(\cdot) \ , \quad j = 1, ..., N \ . \tag{1.17}$$

An operator is introduced that maps the functions h_n to polynomials:

$$h_{\downarrow,\rho} = P_m h \ , \quad m = \min \{ \in \mathbb{N}_0 \ : \ P_k h \neq 0 \} \ , \tag{1.18}$$

where we introduce a subscript ρ to emphasise the fact that the operator P depends on the choice of ρ. We extend the operator $(\cdot)_{\downarrow,\rho}$ to vector spaces H by $H_{\downarrow,\rho} = \text{span} \left\{ h_{\downarrow,\rho} \ : \ h \in H \right\}$. If $H = \text{span} \{h_i\}_{i=1}^N$ one can show that the space of polynomials

$$\Pi_{Z,\rho} = H_{\downarrow,\rho} \ , \tag{1.19}$$

is an $N-$dimensional space of d-variate polynomials, that is isomorphic to interpolation data at the sample locations Z. Therefore, a polynomial $g \in \Pi_{Z,\rho}$, satisfying $g(\mathbf{z}_i) = q(\mathbf{z}_i)$, exists and is unique. The polynomial g is called the *least orthogonal interpolant*. The construction of both the interpolation space and the interpolation coefficients is accomplished by standard linear algebra operations on the Vandermonde matrix. Notice that when using Gauss quadrature sample locations associated with the specified PDF as input for least orthogonal interpolation, the classic orthogonal basis polynomials result, and the procedure boils down to common non-intrusive stochastic collocation. For an in-depth discussion of least orthogonal interpolation, see [14].

Surrogate construction

We denote the least orthogonal interpolation operator by $I[\cdot]$, which operates on a subset of $(\mathbf{z}_i, q(\mathbf{z}_i))_{i=1}^n$, and we assume that the random space is decomposed into N_c elements E_i. Each element E_i comprises a single class c_i, which consists of the samples $(\mathbf{Z}_i, \mathbf{Q}_i)$. The global approximation is given by, see equation (1.3):

$$\tilde{q}(\mathbf{z}) = \sum_{i=1}^{N_c} I[(\mathbf{Z}_i, \mathbf{Q}_i)] \mathscr{I}_{E_i}(\mathbf{z}) \ , \tag{1.20}$$

where $\mathscr{I}_{E_i}(\mathbf{z})$ is the indicator function satisfying $\mathscr{I}_{E_i}(\mathbf{z}) = 1$, if $\mathbf{z} \in E_i$ and 0 otherwise.

IV Complete algorithm

The complete MST-ME method is shown in figure 1.7. Apart from the weight function and sampling threshold, no additional input from the user is required, which makes the method suitable for generic problems.

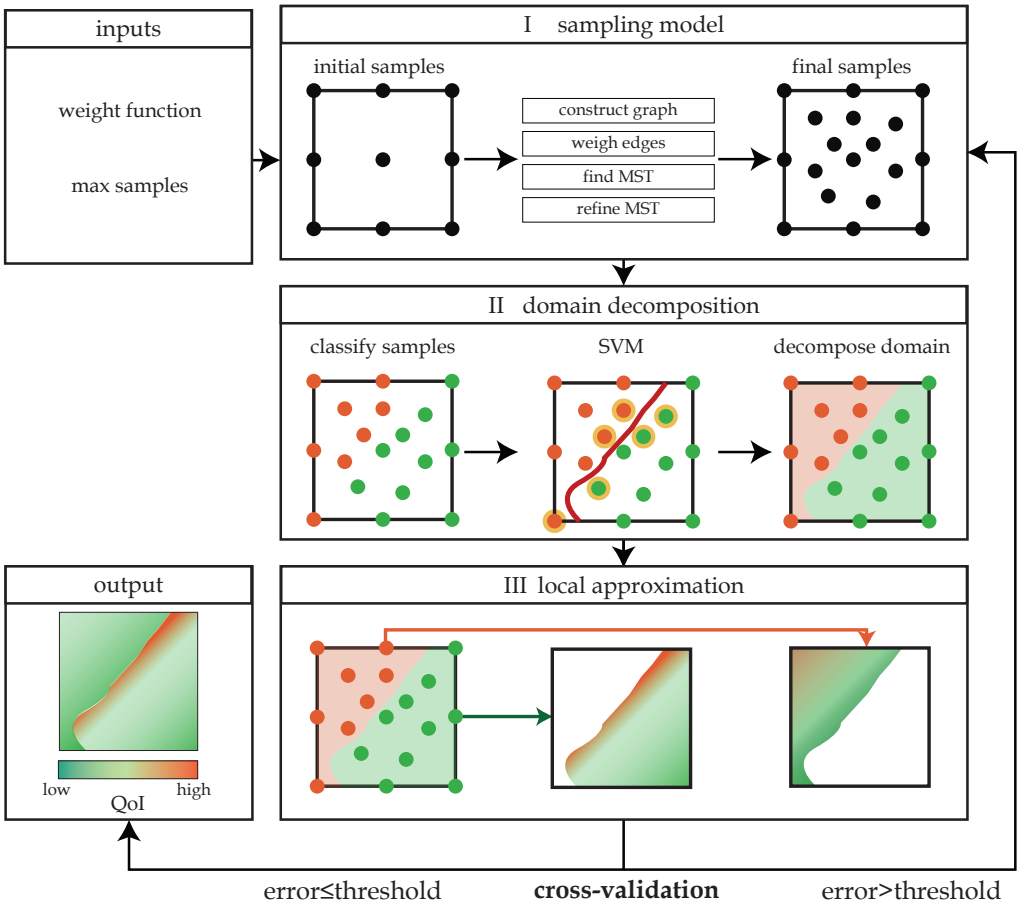


Figure 1.7: *A schematic overview of the complete MST-ME method in a 2D random space.*

A suitable stopping criterion is given by checking if cross-validation errors in the surrogate drop below a specified threshold. Computing the cross-validation errors is straightforward and is based on the leave-one-out principle. The errors in the surrogate given by cross-validation are unbiased [24, 25, 26] and give a good indication on how well the algorithm performs. In fact, as we use the adaptive least orthogonal interpolation, some samples are not used in the interpolation procedure and are left out. These samples can be used to compute the validation error without the need to leave out relevant samples.

Global approximation error

The adaptive nature and incorporation of SVMs in our approach makes it difficult to derive error bounds. However, in this subsection we discuss how the error is composed and how it can be monitored. The global approximation error can be decomposed into two parts.

The first part comprises the error made by the SVM, i.e., the error in the approximation for the

discontinuity location. Strict error bounds are hard to derive for SVMs in general. However, Vapnik and Chapelle showed that the cross-validation error of the SVM gives an unbiased estimate of the error in the domain decomposition step [24], and it yields therefore a proper way to monitor the error of the SVM approximation.

The second part comprises the error made by interpolating the samples. Least orthogonal interpolation is used, which is able to interpolate scattered data sets adaptively. A quality measure of our sample set can be given by the Lebesgue constant, which is a measure of how well an interpolant is in comparison with the best polynomial approximation. In the least orthogonal interpolation method, the Lebesgue constant in 1D is defined as [14]

$$\Lambda_N = \sup_{\mathbf{z} \in I_\mathbf{z}} \lambda(\mathbf{z}) \,, \tag{1.21}$$

where

$$\lambda(\mathbf{z}) = \sum_{i=1}^{N} |l_i(\mathbf{z})| \,, \tag{1.22}$$

where N is the number of samples and $l_i(\mathbf{z})$ is the Lagrange interpolating polynomial corresponding to node i. The Lebesgue constant is related to the error in the interpolant as follows

$$\|q - \tilde{q}\| \leq (1 + \Lambda_N) \|q - q^*\| \,, \tag{1.23}$$

where q, $\tilde{q}$, and q^* are the exact function, the surrogate and the best approximating polynomial amongst the set of polynomials of degree $N-1$, respectively. The adaptively chosen sample sets that are used for interpolation on the elements can be used to compute the Lebesgue constant, which gives an indication of the quality of the nodal locations. However, for the 1D test-cases in section 6, the Lebesgue constant gives no additional insight and is therefore not further discussed.

Computational cost for N samples in d-dimensional random space

The computational cost of the sampling strategy (I) is determined by the cost of computing the Voronoi diagrams and finding the MST. Computing a Voronoi diagram on N samples in $\mathbb{R}^d$ can be done in $\mathcal{O}(N \log(N) + N^{\lceil d/2 \rceil})$ time [27]. In the worst-case scenario, only a single sample is added in each iteration, and hence we have to compute N Voronoi diagrams on N samples. The maximum computational cost is therefore $\mathcal{O}(N^2 \log(N) + N^{\lceil d/2 \rceil + 1})$. Notice that the Voronoi diagrams are not suited for high-dimensional spaces, because of the factor $d/2$ in the exponent. Instead, when Voronoi diagrams are too expensive to compute, a simple radius search algorithm can be used. Computing the MST can be done with Prim's algorithm [28], which has an algorithmic complexity of $\mathcal{O}(|E| \log(|V|))$, where $|E|$ is the total number of edges and $|V|$ the total number of samples. An upper bound for $|E|$ and $|V|$ is given by $N(N+1)/2$ and N, respectively. Again, in the worst case scenario we have to perform N iterations, which can be done in $\mathcal{O}(N^3 \log(N))$ time.

The computational cost of the domain decomposition (II) is based on the complexity of the classification procedure and the SVM. Classification of N samples can be performed in $\mathcal{O}(N^2)$ time. The SVM has a complexity ranging between $\mathcal{O}(N^2)$ and $\mathcal{O}(N^3)$, depending on the number of classes and the kernel [21]. Hence, domain decomposition can be performed in $\mathcal{O}(N^3)$ time and is independent of d.

The local approximation (III) uses LU- and QR-decomposition for determining the interpolant. Computing the QR-decomposition is the dominant factor in (III), it can be done in $\mathcal{O}(N^3)$ time, using the standard implementation in Matlab. If least orthogonal interpolation is employed in an adaptive fashion, the complexity increases to $\mathcal{O}(N^4)$, as N QR-decompositions are computed in the worst case.

The complexity of the complete algorithm is determined by the complexities of (I), (II) and (III). This results in an overall complexity of $\mathcal{O}(N^4)$ if $d \leq 5$ and $\mathcal{O}(N^3 \log(N) + N^{\lceil d/2 \rceil + 1})$ otherwise.

4 Results

In this section we present multiple examples that illustrate the robustness and flexibility of the MST-ME method. For computing the error between the exact surrogate and approximation we use the following weighed $L_{2,\rho}$-norm:

$$\|\tilde{q} - q\|_{2,\rho}^2 = \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \rho(\mathbf{z}_i^{MC}) \cdot |\tilde{q}(\mathbf{z}_i^{MC}) - q(\mathbf{z}_i^{MC})|^2 \,, \tag{1.24}$$

where the surrogate model $\tilde{q}$ is constructed using MST-ME and evaluated at Monte Carlo sample locations $\mathbf{z}_i^{MC}$ drawn from the PDF ρ. The exact solution q is the evaluation of the model sampled at the same Monte Carlo samples. To clarify, the error compares the constructed surrogate with the exact solution at the Monte Carlo sample locations, and this error should converge to 0 if the constructed surrogate converges to the exact response. We multiply the difference between the surrogate and exact solution with the PDF ρ to emphasise on regions in the random space that are likely to occur. Samples that are within a distance from the discontinuity, which is lower than the minimum distance of the adaptive MST-ME samples, are discarded. Discarding samples is motivated by the fact that these lie below the resolution of the SVM discontinuity detection, where the fidelity of the classification is questionable [6].

The first example shows the approximation of three 2D, piecewise constant functions. This example focuses solely on the domain classification and shows convergence of the SVM domain decomposition (II). The second example shows the approximation of 1D and multidimensional Genz functions [29]. The third test-case shows the MST-ME method applied to a more complicated model, which is defined by an underlying PDE, namely, the shallow water equations. Lastly, we apply the MST-ME method to a 3D dam break problem simulated through an incompressible Smoothed Particle Hydrodynamics (SPH) model, to indicate that our methodology can be applied to complex engineering problems.

4.1 Domain classification

The accuracy of the SVM domain decomposition (II) is investigated as a function of the parameter γ.

SVM domain decomposition works best for discontinuities without corners

The SVM domain decomposition (II) is tested by approximating three piecewise constant 2D functions. The functions have a discontinuity in the shape of a circle, a rectangle and a triangle, respectively. Notice that all discontinuities divide the domain into two sub-domains, and therefore are detectable by the classification procedure in II. The sampling algorithm (I), with a weight function that focuses solely on the gradient (1.5), is used to determine the sample locations. The SVM domain decomposition uses the radial basis function kernel (1.12) in which γ is set to the advised value $1/N_c$ [21], where N_c is the number of classes. The sample locations for the circle test-case are shown in figure 1.8 (left), along with the definition of the error measure (the misclassified portion). Clustering of samples appears around the circular-shaped discontinuity location, because the gradient based weight function (1.5) does not lead to refinement if there is no intersection with the discontinuity. The sample locations show symmetry, but after 5 iterations, the symmetry is slightly lost, although this is not noticeable in figure 1.8 (left).
The domain decomposition error as a function of the number of samples is shown in figure 1.8 (right).

The misclassified portion of the domain decreases rapidly with the increasing number of samples, but is in general not monotonically decreasing. As stated in section II, discontinuity lines with sharp corners, in particular the square and triangle, are hard to approximate for the SVM with

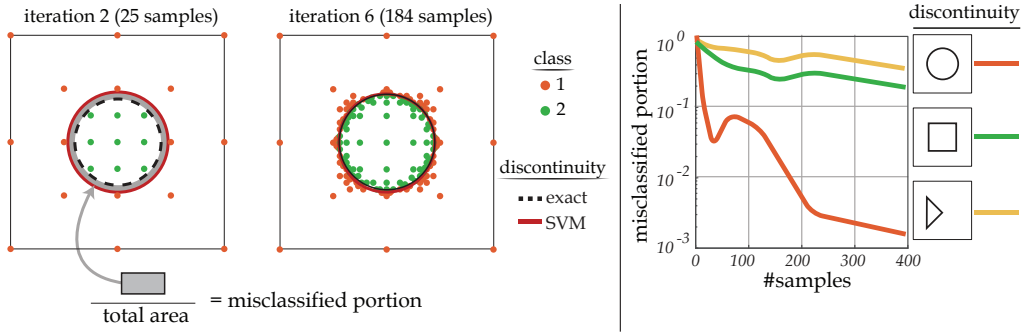


Figure 1.8: *SVM domain decomposition results. (left) Sample locations at different iterations for the circular test-case, with $\gamma = \frac{1}{2}$. (right) Convergence of the misclassified portion for increasing number of samples.*

a radial basis function kernel. However, the regularity of the classification boundary is often not known beforehand and therefore we opt for the commonly used radial basis function kernel. The value for γ however may significantly influence the accuracy of the discontinuity approximation, and the advised value $1/N_c$ is in general not optimal. Therefore we search for a value of γ that is optimal for the remaining test-cases.

The optimal value for γ for our test-cases is $3/N_c$

We now investigate the optimal value for γ by testing several candidate values for a large set of piecewise constant functions. To clarify, the optimal shape parameter γ for the radial basis function kernel (1.12) is sought for a specific family of discontinuous functions, such that the classification procedure works best. The fact that this value may be optimal for the radial basis function kernel, does not necessarily imply that it is optimal for other kernels as well. Kernel (1.12) is used for classification, we focus on the optimal shape parameter for this specific kernel. The parameter γ may influence the accuracy of the SVM domain classification. Many discontinuous QoIs in engineering applications possess a discontinuity without corners, i.e., the exact discontinuity surface is a smooth hypersurface. For such discontinuities, the value of $1/N_c$ might not be optimal and therefore we search for an optimal value among a set of candidates that are obtained by adding perturbations to the advised value: $\{\frac{1}{N_c+4}, \frac{1}{N_c+3}, \frac{1}{N_c+2}, \frac{1}{N_c+1}, \frac{1}{N_c}, \frac{2}{N_c}, \frac{3}{N_c}, \frac{4}{N_c}, \frac{5}{N_c}\}$. A set of 10^6 piecewise constant functions, possessing up to 3 discontinuities, on the domain $[-1,1]^2$, is randomly generated. The discontinuity location is given by up to 3 non-intersecting polynomial parametric curves up to degree 5, which have random coefficients. For each of these functions, an SVM domain decomposition is performed for each of the possible values of γ. This domain decomposition is performed on 50 adaptively sampled points from the MST-ME method, based on the weight function (1.5) and the initial configuration shown in figure 1.2c. We add all correctly classified portions for each γ-value, and divide the resulting sum value by 10^6 to obtain an average correctly classified portion for all the randomly generated functions. The number of adaptive samples influences the average correctly classified portion. A similar trend between the γ-values is obtained. The results are shown in figure 1.9.

Figure 1.9 shows that $\gamma = 3/N_c$ is the most accurate choice for the generated family of piecewise constant functions. The inconsistency with the value for γ suggested in literature $(1/N_c)$ and our value is possibly due to the fact that we limit ourselves to a family of discontinuous functions, which possess no sharp corners in the discontinuity surface. Hence, the value $3/N_c$ might not be the optimal value for other families of discontinuous functions. The discontinuities considered in this chapter have no sharp corners in the discontinuity surface and $\gamma = 3/N_c$ is therefore used there as

Figure 1.9: *Average correctly classified portion for different γ-values.*

well.

4.2 Genz functions approximation

To illustrate the accuracy/efficiency, the proposed MST-ME method is applied to standard benchmark problems, namely, approximation of 1D Genz functions [29].

Edge weighing based on PDF and gradient is most robust
We consider the following Genz functions:

$$g_1(x,\alpha) = \cos(\alpha x)\ , \tag{1.25}$$

$$g_3(x,\alpha,\beta) = \left(\frac{1}{1+\alpha x}\right)^{1+\beta}\ , \tag{1.26}$$

$$g_5(x,\alpha,\beta) = \exp(-(\alpha|x|-\beta))\ , \tag{1.27}$$

$$g_6(x,\alpha,\beta) = \begin{cases} 0, & x>\beta\ , \\ \exp(\alpha x), & \text{otherwise}\ . \end{cases} \tag{1.28}$$

A uniform PDF is assumed on the interval $[-1,1]$ and the initial grid consists of the two end points plus the middle point of this interval. As a reference, the MST-ME solution is compared with the stochastic collocation solution on a Gauss-Legendre grid, which is essentially least orthogonal interpolation using Gauss-Legendre sample locations (see section III).
Figure 1.10 shows that weighing based on the PDF only results in a uniformly spaced sample grid. Interpolation on such a grid is in general not a good idea, as it may produce unstable interpolants. The least orthogonal interpolation method partially circumvents this issue by choosing a subset of these samples in constructing an interpolant. Consequently, the smooth Genz functions g_1 and g_3 are well approximated, but g_5 and g_6 are not. Weighing based on the gradient alone leads to improved results for the discontinuous function g_6, but leads to less accurate results for the smooth functions g_1 and g_3. The standard stochastic collocation method performs well in smooth cases, but converges slowly and also shows an oscillating error in some cases, due to the Gauss-Legendre grid that includes the middle point of the domain only for an odd number of samples. In contrast, the weight function based on both the PDF and the gradient performs the best overall, by keeping track of the discontinuity location, while still maintaining a sample distribution which resolves parts of

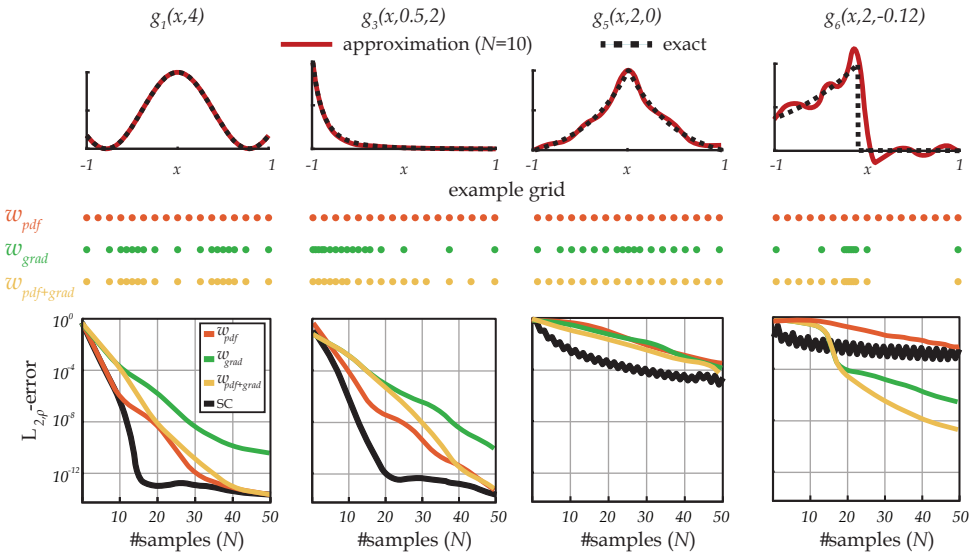


Figure 1.10: *Error of the approximation with respect to the exact function.*

the random space away from the discontinuity. In the remainder of this chapter we therefore use weighing based on both PDF and gradient, equation (1.7).

Notice that all weight functions show slow convergence in the approximation of g_5. This is due to the absence of the second derivative in the weight function (1.7). By basing the classification on the second derivative in the QoI, we can circumvent the slow error convergence in presence of a discontinuity in the first derivative. However, we will not have any discontinuities in the first derivatives for the remaining test-cases. We therefore use classification based on the first derivative only.

The underlying PDF changes the sample grid

The effect of the underlying PDF is now investigated. The two PDFs that we consider are a symmetric and an asymmetric β-distribution, with parameters $(10,10)$ and $(2,7)$, respectively. The support of both PDFs is scaled to $[-1,1]$ and we use the uniform distribution as a reference. The error convergence for g_1 and g_6 is shown in figure 1.11.

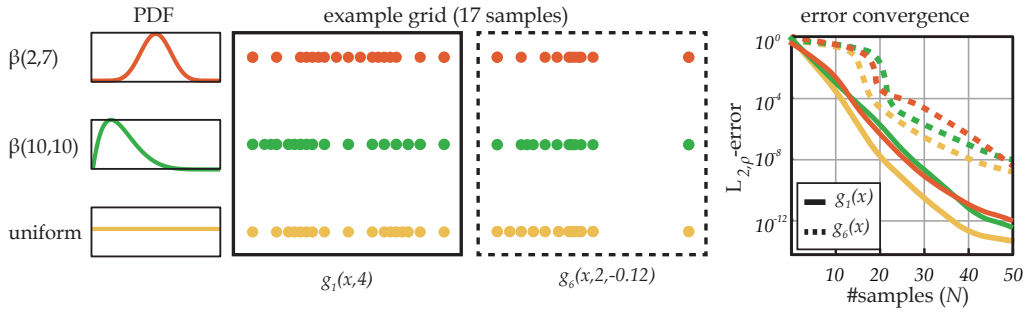


Figure 1.11: *Error convergence for different choices of PDFs.*

The error convergence is similar to the error convergence for the uniform distribution. Again the sample grid is not ideal for interpolation, but the adaptive least orthogonal interpolation circumvents

this by using a subset of nodes.

MST-ME outperforms conventional Latin Hypercube Sampling in multiple dimensions

To investigate how the MST-ME method performs in multiple dimensions, the error is plotted for the smooth Genz function g_1 and the discontinuous Genz function g_6, with increasing dimension d, in figure 1.12 (left). These multidimensional Genz functions are tensor products of the 1D Genz functions. The error for each dimension is based on 1000 adaptively sampled points with an initial sampling configuration equal to the one shown in figure 1.2c. The number of required samples needed to attain a specific accuracy is also plotted as a function of the dimension of the random space (right). As stochastic collocation scales poorly to multiple dimensions when using a tensor grid construction, we compare the MST-ME method with a method which scales better to higher dimensions [30, 31]. As a comparison, we use isotropic Latin Hypercube Sampling (LHS) in combination with the least orthogonal interpolation method, and the results are shown in figure 1.12.

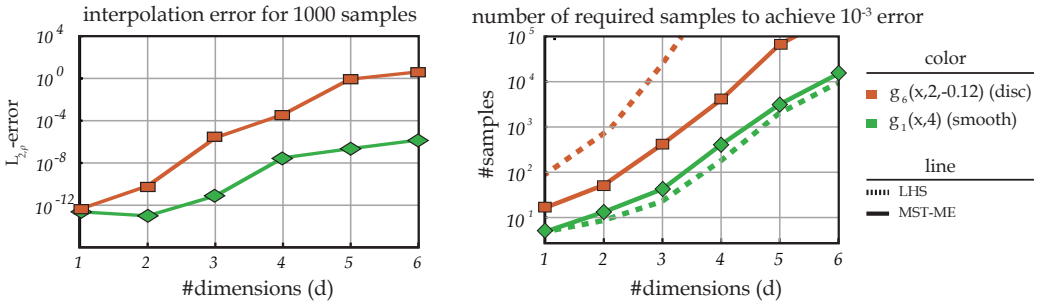


Figure 1.12: *Comparison of the MST-ME method and LHS+least orthogonal interpolation with increasing dimensionality. (left) Error of the approximation with respect to the exact function as a function of the dimension. (right) The number of samples needed to attain a certain accuracy as a function of the dimension.*

We clearly see the exponential increase of the required number of samples for the MST-ME method, which is the well-known curse of dimensionality. The LHS approach scales similarly when compared to the MST-ME for the smooth case, but requires significantly more samples when compared to the MST-ME method.

4.3 Shallow water dam break

We study the performance of the MST-ME method applied to a system of 1D conservation laws. This system consists of the 1D shallow water equations (SWEs), which describe the inviscid flow of a layer of fluid with free surface, under the action of gravity, with the thickness of the fluid layer small compared to the other length scales [32]:

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ hv \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} hv \\ hv^2 + gh^2/2 \end{pmatrix} = \mathbf{0}\,, \tag{1.29}$$

where h is the free surface height (thickness of the fluid layer), v the velocity, and $g = 9.81$ the acceleration of gravity. The initial condition for the system of PDEs is given by:

$$\begin{pmatrix} h \\ v \end{pmatrix}(x, t=0) = \begin{cases} \begin{pmatrix} h_l \\ v_l \end{pmatrix}, & x \leq 0\,, \\ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & x > 0\,, \end{cases} \tag{1.30}$$

leading to a Riemann problem shown in figure 1.13. The solution of the Riemann problem for these initial conditions can be computed exactly when working on an infinite spatial domain [33]. The solution consists of two characteristic waves travelling through the spatial domain, see figure 1.13. Each wave is a shock or rarefaction wave. When solid boundary conditions are imposed at $x = \pm 1$, an exact solution cannot be obtained for all initial solutions. We therefore employ a finite volume method with an exact Riemann solver [34] to compute the cell face fluxes, and solve the SWEs using 256 cells. A Crank-Nicolson scheme is used to integrate the SWEs in time and a ghost-cell method with reflective properties is used for the boundaries. The initial left state (h_l, v_l) is assumed to be uncertain and uniformly distributed $\mathscr{U}$ between $[1.5, 2.5]$ and $[-0.5, 0.5]$, respectively, i.e.:

$$\mathbf{z} = \begin{pmatrix} h_l \\ v_l \end{pmatrix} \sim \begin{pmatrix} \mathscr{U}(1.5, 2.5) \\ \mathscr{U}(-0.5, 0.5) \end{pmatrix} . \tag{1.31}$$

The uncertainty in the initial conditions is large in order to ensure that we get different characteristic behaviours of the QoI. The average thickness of the fluid layer is not shallow compared to the domain length, but this is not important for showing the performance of the MST-ME method. The QoI q is defined as the fluid height at $x = -1$ at a certain time t^*. A schematic representation of this set-up is shown in figure 1.13.

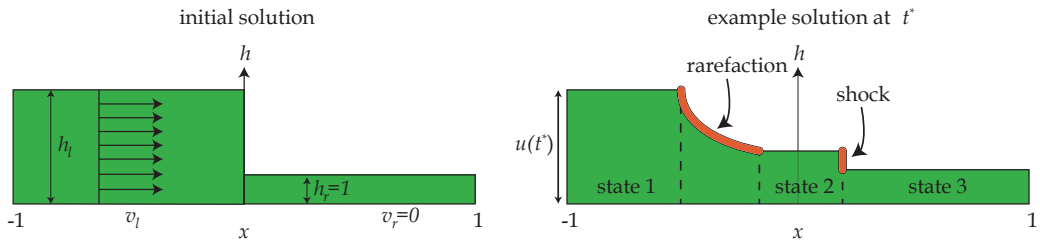


Figure 1.13: *Schematic representation of the shallow water test-case.*

Notice that the QoI is time dependent and that the characteristics of this QoI will change significantly as time progresses. Either a transition between a shock and rarefaction wave, or a difference in wave speeds can result in a discontinuity in q. This allows us to study the robustness of the MST-ME method, as this test-case comprises both smooth and discontinuous QoI responses. We emphasise that the constructed surrogate for the QoI at $t = t^*$ cannot be reused for other time instances, because the MST-ME method uses the QoI at the current time t^* as a measure to place new samples.

MST-ME automatically detects if a function is smooth or discontinuous

The MST-ME method is used for three different QoIs, $u = h(x = -1, t^* \in \{1.67, 4.16, 2.21\})$, which correspond to a mildly non-linear, highly non-linear and close to discontinuous QoI, respectively. The surrogate model and sample grids after 10 iterations are shown in figure 1.14.
The discontinuity in the QoI at time $t^* = 2.21$ is caused by a shock wave, which hits the left boundary for certain values in the random space, but does not yet hit the left boundary for other values in the random space.

To investigate the accuracy of the MST-ME method, we determine the convergence. The error is based on 10^6 Monte Carlo samples. The convergence of the $L_{2,\rho}$-error (1.24) is shown in figure 1.15.
The results show that the error as a function of the samples decays fast for the mildly and highly non-linear case, as expected. The highly non-linear case shows a sudden drop in the error, which is caused by transition in the domain decomposition. First the classification procedure detects a

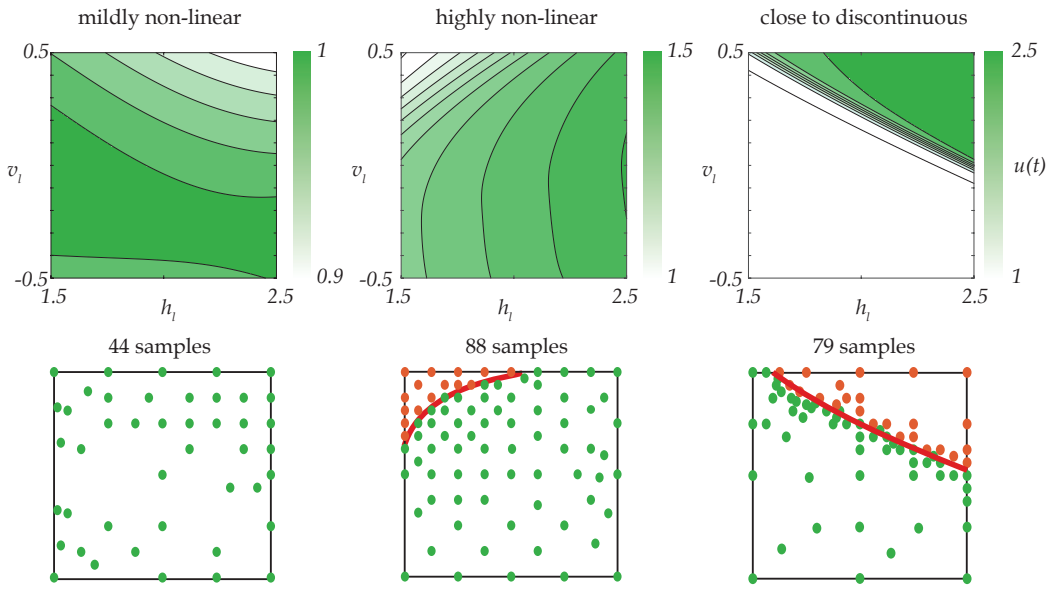


Figure 1.14: *The three QoI surrogate models and corresponding sample grids after 10 iterations.*

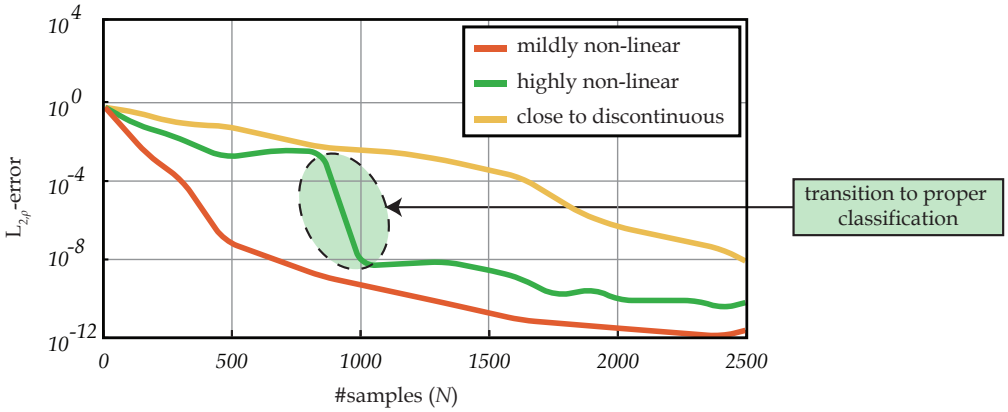


Figure 1.15: *Convergence of the MST-ME solution in the $L_{2,\rho}$-error* (1.24).

large enough jump in the sampled QoI to conclude that there is a discontinuity present in the QoI. As the MST-ME progresses, samples are added in the area of the possible discontinuity, until the jumps in the QoI values become small enough to classify the samples properly. This transition to a correct classification explains the sudden drop in the error for the highly non-linear case. MST-ME automatically detects the smoothness of the QoI, as the number of samples increases.

4.4 3D dam break

As a last test-case, the MST-ME method is applied to a complex engineering problem, namely a 3D fluid dam break problem with parametric initial conditions. This test-case is similar to the shallow water dam break, with the difference that it describes fluid motion in 3D and contains more physics, i.e., viscous effects and no shallow-water assumption. Dam break problems are commonly used as

benchmark in for example sloshing applications [35] and have been studied extensively [35, 36, 37]. MST-ME can be used to gain physical insight for this parametric problem, by constructing a surrogate model in the full parameter space, and this is our goal in this test-case. We do not focus on convergence, as it is computationally infeasible to construct a reliable reference solution. A schematic representation of the test-case is shown in figure 1.16.

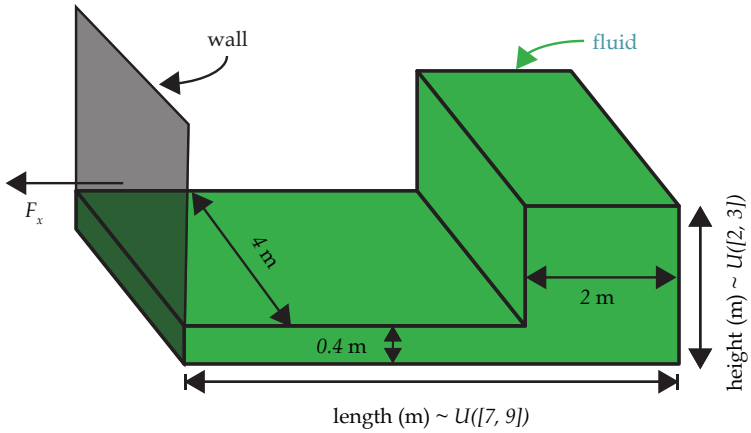


Figure 1.16: *Schematic of initial condition for 3D dam break.*

We consider two uncertain parameters, namely the length of the tank and the height of the right column of liquid. When the fluid is released, it starts to flow to the left side of the domain and impacts the wall, shown in figure 1.16. The QoI is the maximum perpendicular wall force component F_x during first impact. Smoothed particle hydrodynamics (SPH) is used to simulate the free surface flow, induced by the initial condition in figure 1.16. We use an open-source SPH solver, DualSPHysics [38]. The acceleration of gravity is set to $9.81\text{m}/\text{s}^2$ and the kinematic viscosity ν is set to $10^{-6}\text{m}^2/\text{s}$, which is the value for water at room temperature. Surface tension is neglected. Approximately 10^6 particles are used for the simulations, which is considered as medium to high resolution for free surface flows of these length scales. The simulations are performed on a single GPU-unit with 2048 cores and the average simulation time is approximately 14 hours. A typical time-dependent result for a height of 3m and a length of 7m, along with the perpendicular wall force component F_x, is shown in figure 1.17. The four consecutive instances of the example simulation show: initial liquid configuration; wave development due to pressure gradient; breaking wave impact on wall; liquid after impact. Depending on the height difference between both liquid columns, wave breaking may occur before the wave impacts the wall. In reality, a gas pocket may be entrapped during this process. The gas pocket may be compressed and next lead to an inside pressure build-up. However, the simulations performed here are single phase free surface flows and the gas is not taken into account, so the physics in an entrapped gas pocket is ignored.

Obtaining a parametric solution is the target, where both parameters are assumed to be uniformly distributed. The MST-ME method is used to construct the unknown QoI response. A total of 6 iterations of the sampling algorithm is performed with weight function (1.7), which results in 30 samples. We expect the MST-ME method to automatically distinguish smooth and discontinuous behaviour of the QoI, which is important for this problem, since we have limited knowledge of the QoI as a function of the parameters. The results are shown in figure 1.18.

The results indicate a smooth QoI response, which is approximately constant along the lines where height/length is constant. This implies that for single phase free surface flow, the force on the wall depends roughly on the ratio of height and length, and not on their separate values. The wall force increases when this ratio increases, which is intuitive from a physical point of view. Figure 1.18

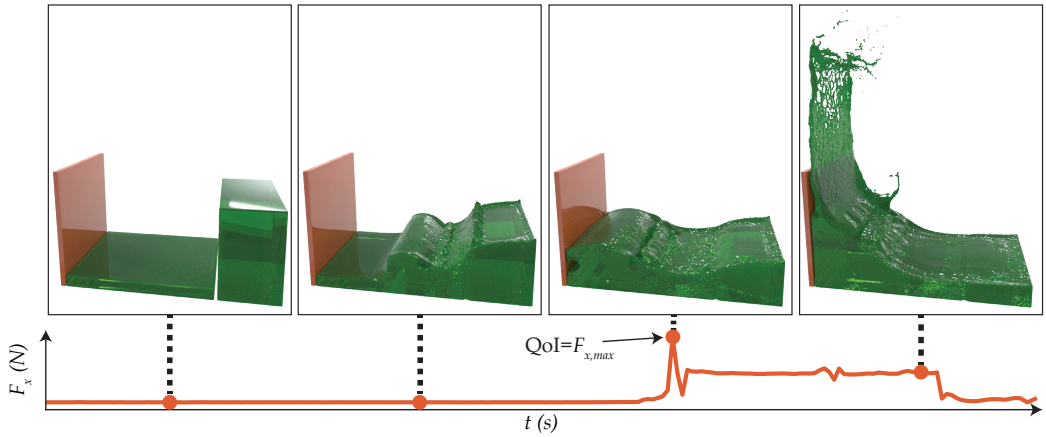


Figure 1.17: *Single simulation with height of 3m and a length of 7m. The QoI is the maximum of the wall force F_x.*

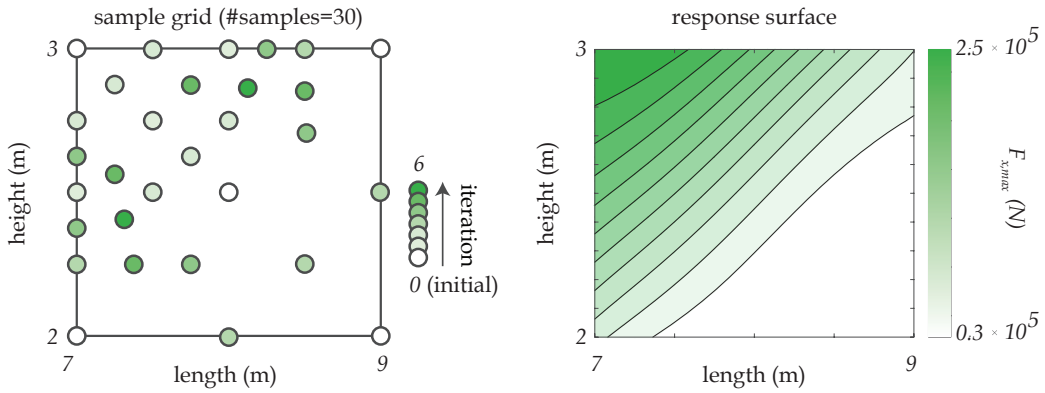


Figure 1.18: *QoI of the 3D dam break problem, obtained by the MST-ME method with 6 iterations.*

(right) shows that this increase is non-linear, which corresponds to results previously reported for dry-bed dam break problems [39].

Interestingly, in contrast to the SWE test-case, the QoI shows no discontinuity in the parameter space. This is possibly due to neglecting the gas phase in the single phase free surface simulations. When simulated with a gas phase, the pressure build-up in the entrapped gas pocket (see figure 1.17) may lead to a discontinuity in the QoI. The strength of our proposed MST-ME method is that we do not require knowledge about the characteristics of the QoI beforehand, as it distinguishes smooth and discontinuous behaviour automatically.

5 Conclusion

In this chapter we presented a novel domain decomposition based interpolation method, the Minimum Spanning Tree Multi-Element (MST-ME) method. The unique property of the MST-ME method is that it adaptively constructs a surrogate model as a function of a set of uncertain parameters for both smooth and discontinuous quantities of interest. The three key ingredients in the MST-ME method are: (I) adaptive sampling based on a minimum spanning tree with a smart weight

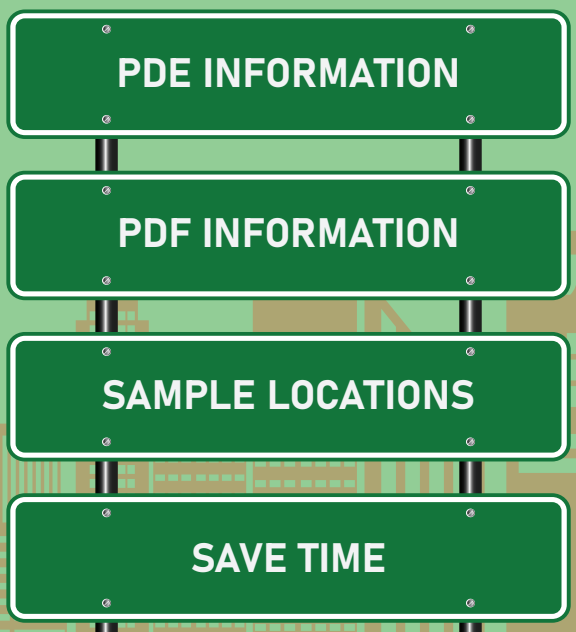
function, (II) discontinuity detection and sample classification with a support vector machine algorithm, and (III) least orthogonal interpolation to construct local approximations. This combination of robust methods makes the MST-ME method a very practical method that is applicable to a wide range of UQ problems.

The MST-ME method has been applied to several numerical examples: domain decomposition, Genz function approximation, 1D shallow water equations, and a 3D dam break problem. Discontinuities present in these test-cases are effectively captured by the method. In all cases, fast convergence is obtained, leading to an accurate surrogate model already at a relatively low number of model runs. This surrogate model can be directly used as a fast tool for uncertainty quantification (for example with Monte Carlo type methods), but it is also a useful tool for the parametric solution of black-box models, including PDEs. We also foresee application of this surrogate model in the solution of inverse problems. The freedom in the weighing function of the minimum spanning tree offers many applications, such as adaptive sampling for reliability analysis, where the weighing function may be adapted such that samples are placed in regions of low probability.

Currently, the MST-ME method does not include the option to detect discontinuities that do not divide the random space into several simply connected subdomains. Furthermore, the MST-ME method can not yet preserve the symmetry in the node distributions, which might be advantageous in certain special cases (e.g. when both the model and the underlying PDF of the random variables are symmetric). Lastly, the least orthogonal interpolation does not necessarily use all sample points in the construction of the local approximation. By adding a term to the weight function of the minimum spanning tree, which accounts for the stability of the interpolant (as is done for example in Leja nodes [40]), the sample locations may be further improved, such that all samples are used in the construction of the interpolant.

What did we achieve?

This chapter discussed how to effectively place your samples in the random space when the QoI possesses a discontinuity in the random space. By effectively placing samples based on the proposed MST-ME method, we are able to construct an accurate surrogate, even when a discontinuity is present, also in high-dimensional spaces.


PDE INFORMATION
PDF INFORMATION
SAMPLE LOCATIONS
SAVE TIME


PDE-informed sampling
for uncertainty quantification

2 PDE-informed sampling

PDE/PDF-informed adaptive sampling for efficient non-intrusive surrogate modelling

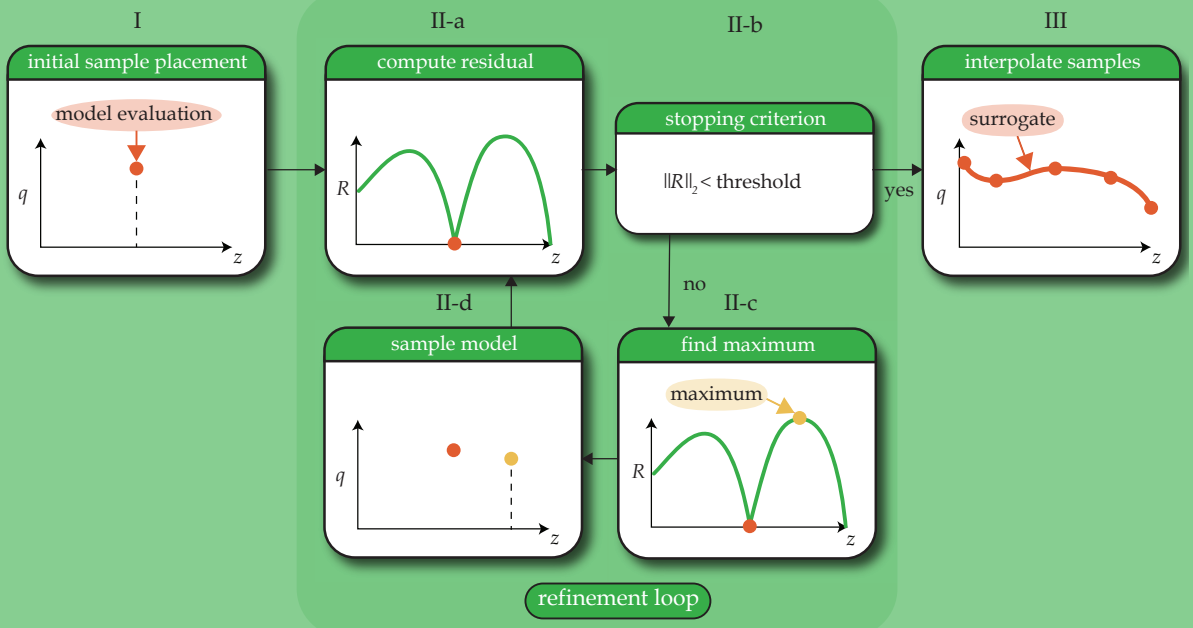
A novel refinement measure for non-intrusive surrogate modelling of PDEs with uncertain parameters is proposed. Our approach uses an empirical interpolation procedure, where the proposed refinement measure is based on both the PDE residual and the PDF. An important strength is that it excludes parts of the PDE solution that are not required to compute the QoI. The PDE residual used in the refinement measure is computed by using all the partial derivatives that enter the PDE separately. The proposed refinement measure is suited for efficient parametric surrogate construction when the underlying PDE is known, even when the parameter space is non-hypercube, and has no restrictions on the type of the discretisation method. Therefore, we are not restricted to conventional discretisation techniques, e.g., finite elements or finite volumes, and the proposed method is shown to be effective when used in combination with recently introduced neural network PDE solvers. We present several numerical examples with increasing complexity that demonstrate accuracy, efficiency and generality of the method.

Background

As stated in the introduction, the goal of part I is to reduce the required number of samples when constructing an accurate surrogate for a QoI that is complicated by either possessing discontinuities or lives in a high-dimensional space. We already discussed how to effectively sample the QoI in the presence of discontinuities. In this chapter we will discuss how to deal with high-dimensional and/or non-rectangular random spaces by incorporating knowledge from the underlying PDE and PDF. Commonly used non-intrusive methods, e.g., stochastic collocation, place samples that are based on the input uncertainties and do not use the sampled QoI to determine samples locations, which may result in placing samples in regions where the QoI does not provide any additional information. In order to place samples more effectively, methods like empirical interpolation [41, 42, 43, 44] and 'best' interpolation [45, 46] were introduced. These alternatives sample the model adaptively in order to reduce the number of samples required for constructing an accurate surrogate model. Furthermore, the empirical interpolation approach enhances adaptive sampling by incorporating knowledge from the underlying model in terms of the PDE residual, which is a measure of how well a discrete approximation satisfies the model. Even though this results in a significant decrease in the number of samples compared to methods that do not take the model into account, the method is still unsuited for uncertainty propagation with a large number of uncertainties. One reason for this is that the empirical interpolation bases sample placement on the entire solution, rather than focusing on the QoI. Furthermore, when interested in the statistical properties of the QoI, e.g., mean and variance, using only the residual as a measure for adaptive sampling placement is not efficient, as it does not utilise the PDF, which is used in the calculation of these quantities.

Published in International Journal for Uncertainty Quantification

Our Approach



The highlights of the new approach are:

- *Refinement procedure for empirical interpolation is proposed which works not only for elliptic PDEs;*
- *Incorporates both PDE information and probability density information of the uncertain parameters;*
- *Suited for efficient surrogate construction on non-hypercube domains;*
- *Faster convergence dependent on the QoI when compared to conventional empirical interpolation;*
- *No restrictions on the underlying discretization, e.g., finite elements or finite volumes;*
- *Perfectly suited to combine with new state-of-the-art neural network based PDE solvers.*

1 Preliminaries

In order for our approach to work, we need to assume a specific form of the underlying PDE (P. 1) (page 16), which is given by:

$$\sum_{l=1}^{n_l} \mathscr{G}_l(\mathbf{z},\mathbf{x})\mathscr{L}_l(v;\mathbf{x}) = \mathscr{S}(\mathbf{z},\mathbf{x}) \ , \quad (\mathbf{x},\mathbf{z}) \in D \times I_{\mathbf{z}} \ , \tag{2.1}$$

supplemented with proper initial and boundary conditions. In (2.1), n_l denotes the number of differential operators in the PDE, $\mathbf{z} \in I_{\mathbf{z}} \subset \mathbb{R}^d$ a d-dimensional vector containing uncertain parameters with corresponding joint PDF $\rho(\mathbf{z})$, $\mathbf{x} \in D$ a vector containing spatial and/or temporal coordinates, $\mathscr{L}_l$ differential operators, $\mathscr{G}_l$ known functions, $\mathscr{S}$ a source term, and $v(\mathbf{z},\mathbf{x})$ the exact solution of the PDE. This particular explicit PDE-form, where the uncertainties enter the equation via the source term $\mathscr{S}(\mathbf{z},\mathbf{x})$ and via the functions $\mathscr{G}_l(\mathbf{z},\mathbf{x})$ as uncertain coefficients in front of differential operators, is more restrictive than the more general form $F(v,\mathbf{x}) = 0$. The explicit form is used because it allows for the definition of a non-zero residual in the random space $I_{\mathbf{z}}$. This PDE-form does not comprise all possible PDEs, but many PDEs with parametric uncertainties, e.g., isotropic diffusion equations, Navier-Stokes equations and advection-diffusion equations, may be written in this form:

$$v_t^e = z_1 v_{xx}^e \ , \tag{2.2}$$

$$v_t^e + z_1 v_x^e = z_2(x) v_{xx}^e \ , \tag{2.3}$$

$$v_t^e + (v^e \cdot \nabla) v^e = -\frac{\nabla p^e}{\rho} + z \Delta v^e \ . \tag{2.4}$$

Because the exact solution of (2.1) is often not available, a discrete solution vector $\mathbf{v}(\mathbf{z},\mathbf{x})$ is computed, e.g., via a finite-difference method or finite-volume method, which satisfies a discretised form of (2.1) for $\mathbf{z} \in I_{\mathbf{z}}$:

$$\sum_{l=1}^{n_l} G_l(\mathbf{z},X) L_l(\mathbf{v}(\mathbf{z})) = \mathbf{S}(\mathbf{z},X) \ , \quad X \subset D, \mathbf{v} \in \mathbb{R}^{N_{PDE}} \ . \tag{2.5}$$

Here $L_l : \mathbb{R}^{N_{\text{PDE}}} \to \mathbb{R}^{N_{\text{PDE}}}$ are the discretised PDE operators, $X = (\mathbf{x}_1, ..., \mathbf{x}_{N_{\text{PDE}}})$ is the computational grid in space and time, consisting of N_{PDE} grid points, $G_l \in \mathbb{R}^{N_{\text{PDE}} \times N_{\text{PDE}}}$ are diagonal matrices with diagonal entries $G_{l,ii}(\mathbf{z},X) = g_l(\mathbf{z},X_i)$, and $\mathbf{S}(\mathbf{z},X) \in \mathbb{R}^{N_{\text{PDE}}}$ is the source term vector. In case of finite difference methods, the source term vector $\mathbf{S}$ is the source term evaluated on the computational grid, but in the case of finite volume or finite element methods the source vector may involve discretisation operations as well. The initial and boundary conditions also enter via the source term $\mathbf{S}$. The uncertainties enter the equations via the source term and the matrices G_l, which comprise the function $\mathscr{G}_l$ evaluated on the computational grid X.

We are again interested in the dependence of the QoI q on the parameters $\mathbf{z}$. In this chapter we also assume a specific form for the operator $\mathscr{Q}$ that is used in (P. 2). The QoI is a vector $\mathbf{q}(\mathbf{z})$ and is assumed to be a set of linear combinations of the discrete solution vector $\mathbf{v}$, i.e., $\mathbf{q}(\mathbf{z}) = Q\mathbf{v}(\mathbf{z})$, where

$$Q : \mathbb{R}^{N_{\text{PDE}}} \to \mathbb{R}^{N_{\text{QoI}}} \ , \tag{2.6}$$

is a matrix that maps the solution vector $\mathbf{v}$ to the QoI $\mathbf{q}$. This assumption allows for a suitable refinement measure for sampling, which is introduced later. By assuming linearity of Q we limit the space of possible QoIs, but this limitation is not too severe as many quantities, e.g., integral quantities and mean quantities, can be written in this form.

2 Our approach in short

In this work, an empirical interpolation procedure related to [42, 44, 45, 47, 48, 49, 50, 51] is proposed, with the main differences that: first, the PDF of the parameters is included in the sampling algorithm, and second, fewer restrictions on the type of the PDE are imposed (when compared to e.g. [50]), and third, the residual is based on the QoI and not on the entire PDE solution. A relation between the error in the surrogate and the PDE residual is given, which justifies the residual as a refinement measure for surrogate construction. Furthermore an alternative refinement measure, which incorporates the PDF, is proposed when interested in the statistical properties of the QoI. Using both the residual and PDF as a measure for defining new sampling locations, leads to accurate statistical properties of the QoI, which converge significantly faster than the procedures in [42, 44, 45]. Additionally, it is shown that the proposed method is suited for efficient surrogate construction on complex topologies, which is a common problem in the case of dependent input uncertainties. Our method does not require a specific type of PDE discretisation, e.g., finite elements or finite volumes, and can in fact also be used in combination with state-of-the-art neural network solvers [52]. A key part of our approach is the use of the PDE residual, which is discussed later in more detail. In order to compute this residual, the black-box solver needs to give not only the solution values, but also derivatives with respect to spatial and/or temporal coordinates and will be referred to as a gray-box. This introduces a small degree of intrusiveness in our approach, although no changes to the model equations are necessary. Our approach is therefore still referred to as a non-intrusive approach. As our approach is still considered to be non-intrusive and uses a combination of the PDE residual and PDF of the uncertain parameters as a refinement measure, we refer to the proposed method as Non-Intrusive PDE/PDF-informed Adaptive Sampling (NIPPAS).

The goal in this work is twofold: either construct an accurate surrogate $\tilde{\mathbf{q}}(\mathbf{z})$ for $\mathbf{q}(\mathbf{z})$ or calculate statistical properties of $\mathbf{q}(\mathbf{z})$ with respect to the uncertain parameters $\mathbf{z}$. The latter, calculating statistical properties, is typically achieved by the former, constructing a surrogate, in combination with Monte Carlo sampling of the surrogate to extract the statistical quantities. However, if we are only interested in the statistical properties of the QoI, the surrogate does not need to be accurate everywhere, as some areas of the random space contribute little when calculating these statistical properties. Nevertheless, whether we are interested in constructing an accurate surrogate to study the dependency of the QoI on the parameters $\mathbf{z}$, or whether we are interested in the statistical properties of the QoI, a surrogate needs to be constructed. We construct a surrogate by solving the discretised PDE (2.5) and by sampling values from $\mathbf{q}(\mathbf{z})$. We sample the QoI $\mathbf{q}(\mathbf{z}_i)$ at $N+1$ locations $\{\mathbf{z}_i\}_{i=0}^{N}$ in the random space $I_{\mathbf{z}}$. The QoI evaluations $\mathbf{q}(\mathbf{z}_i)$ are calculated as follows

$$\underbrace{\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\mathbf{v}(\mathbf{z})) = \mathbf{S}(\mathbf{z},X) \Rightarrow}_{\text{solve PDE for } \mathbf{z}=\mathbf{z}_i} \mathbf{q}(\mathbf{z}_i) = Q\mathbf{v}(\mathbf{z}_i)\ . \tag{2.7}$$

The PDE-solver is assumed to be a gray-box, which means that we supply inputs and receive outputs, with the possibility to observe intermediate calculated quantities, but without the possibility to alter the intermediate steps. After sampling, a surrogate model $\tilde{\mathbf{q}}(\mathbf{z})$ is constructed by means of polynomial interpolation on the samples $\{(\mathbf{z}_i, \mathbf{q}(\mathbf{z}_i))\}_{i=0}^{N}$. The interpolant is constructed individually for each element of $\mathbf{q}$, such that:

$$\tilde{q}_j(\mathbf{z}) \approx q_j(\mathbf{z}), \ \text{ for all } \ \mathbf{z} \in I_{\mathbf{z}}\ , \ \text{ with } \tilde{q}_j(\mathbf{z}_i) = q_j(\mathbf{z}_i) \ \ i = 0,...,N\ , \tag{2.8}$$

where q_j corresponds to the j-th element of the vector $\mathbf{q}$. The element-wise approximation for the entire QoI vector $\mathbf{q}$ is denoted as $\tilde{\mathbf{q}}$. Polynomial interpolation is used instead of polynomial regression as it allows for more efficient adaptive refinement and has extensive theoretical grounding.

When interpolating, placing a new sample ensures that the new surrogate model is accurate in a neighbourhood around the newly added sample and has therefore immediate impact on the surrogate in the sampled area. This ensures improved accuracy near the new sample location, something which is not necessarily the case when using regression. Choosing proper sample locations $\mathbf{z}_i$ is crucial for stable and accurate interpolation and this will be the main focus of this chapter.

Many UQ methods focus either on the PDF [53] or on the PDE residual [42, 44, 45, 50] for adaptive sample placement. The PDF indicates which values for $\mathbf{z}$ are likely to happen and are therefore important to sample. The PDE residual however gives an indication where surrogate refinement is needed in order for the surrogate to satisfy the underlying PDE. In this chapter we propose a novel strategy, in which the importance of both the PDE and PDF is taken into account in determining the sample locations. Finding a set of sample locations, which resembles the importance of both the underlying PDE and the PDF, is the goal of this chapter.

3 Non-intrusive PDE/PDF-informed adaptive sampling

The locations of the interpolation samples determine the accuracy and stability of the surrogate model. We aim for accuracy and stability by constructing a set of interpolation nodes adaptively, by using knowledge from the underlying PDE as refinement measure.

3.1 Residual definition

For this purpose we define the PDE residual, which indicates how well an approximation of the parametric PDE solution satisfies the parametric discretised PDE in the random space $I_{\mathbf{z}}$. An intuitive definition of the residual can be obtained by first constructing approximations $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$ in the random space based on evaluations $L_l(\mathbf{v}(\mathbf{z}_i))$ and interpolation (2.8), and substituting these approximations into the discretised PDE (2.5):

$$\mathbf{R}_v(\mathbf{z}) := \sum_{l=1}^{n_l} G_l(\mathbf{z},X)\widetilde{L_l(\mathbf{v}(\mathbf{z}))} - \mathbf{S}(\mathbf{z},X)\ . \tag{2.9}$$

The quantity $\mathbf{R}_v$ indicates how well the approximations $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$ satisfy the discretised PDE in the random space $I_{\mathbf{z}}$.

3.2 Relation between residual and surrogate error

The residual $\mathbf{R}_v(\mathbf{z})$ is a quantity that indicates the quality of a parametric approximation by substituting the approximation into the discretised PDE and by calculating the error. The next theorem states a relation between the residual $\mathbf{R}_v$ and the error in the surrogate $\tilde{\mathbf{q}}$ for linear PDEs, which is used later to define a suitable refinement measure for placing new samples in the random space.

Theorem 1. *Assume the following:*

- *Bounded random space $I_{\mathbf{z}}$.*
- *Stable discretization of well-posed linear PDE with probability 1 for all $\mathbf{z} \in I_{\mathbf{z}}$, of the form*

$$\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\mathbf{v}(\mathbf{z})) = \mathbf{S}(\mathbf{z},X)\ ,\ \ X \subset D, \mathbf{v} \in \mathbb{R}^{N_{PDE}}\ . \tag{2.10}$$

- *L_l are discretised linear differential operators, i.e., matrices satisfying:*

$$L_l(\mathbf{v}+\mathbf{w}) = L_l\mathbf{v} + L_l\mathbf{w} \,. \tag{2.11}$$

- *Interpolation and matrix multiplication commute, i.e., $\widetilde{L_l\mathbf{z}} = L_l\tilde{\mathbf{z}}$.*
- *$G_l(\mathbf{z},X)$ are known matrices as functions of $\mathbf{z}$ and X.*
- *QoI vector can be written as $\mathbf{q} = Q\mathbf{v}$, where $Q \in \mathbb{R}^{N_{QoI}\times N_{PDE}}$.*

Then the following holds:

$$Q\left(\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l\right)^{-1}(\mathbf{R}_v(\mathbf{z})) = \tilde{\mathbf{q}}(\mathbf{z}) - \mathbf{q}(\mathbf{z}) \,, \tag{2.12}$$

where $\tilde{\mathbf{q}}$ is the interpolant in the random space $I_{\mathbf{z}}$ for each element of the vector $\mathbf{q}$, based on the samples $\{(\mathbf{z}_i, \mathbf{q}(\mathbf{z}_i)\}_{i=0}^N$.

Proof. Substitution of assumption 1 into (2.9) gives

$$\mathbf{R}_v(\mathbf{z}) = \sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\tilde{\mathbf{v}}(\mathbf{z})) - \mathbf{S}(\mathbf{z},X) \,. \tag{2.13}$$

Subtracting equation (2.10), results in

$$\mathbf{R}_v(\mathbf{z}) = \sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\tilde{\mathbf{v}}(\mathbf{z})) - \underbrace{\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\mathbf{v}(\mathbf{z}))}_{\mathbf{S}(\mathbf{z},X)} \,. \tag{2.14}$$

This can be rewritten by using linearity of L as

$$\mathbf{R}_v(\mathbf{z}) = \sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l(\tilde{\mathbf{v}}(\mathbf{z}) - \mathbf{v}(\mathbf{z})) \,. \tag{2.15}$$

Using well-posedness of the underlying discretised PDE, we can write

$$\left(\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l\right)^{-1}(\mathbf{R}_v(\mathbf{z})) = \tilde{\mathbf{v}}(\mathbf{z}) - \mathbf{v}(\mathbf{z}) \,, \tag{2.16}$$

which can be multiplied by the matrix Q to obtain the desired result

$$Q\left(\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l\right)^{-1}(\mathbf{R}_v(\mathbf{z})) = \tilde{\mathbf{q}}(\mathbf{z}) - \mathbf{q}(\mathbf{z}) \,, \tag{2.17}$$

□ □

The commutation assumption (assumption 4) holds for a wide variety of interpolation operators and especially for the one used in this chapter (discussed in section 5.1). Theorem 1 states a relation between the residual and the error in the QoI surrogate. From this relation we can derive the error bound stated in the following corollary.

Corollary 1. *Assume the following:*

- *The conditions of theorem 1 are satisfied, such that* (2.12) *holds.*

Then the error $\|\tilde{\mathbf{q}}(\mathbf{z}) - \mathbf{q}(\mathbf{z})\|_2$ *satisfies*

$$\|\tilde{\mathbf{q}}(\mathbf{z}) - \mathbf{q}(\mathbf{z})\|_2 \leq \|Q\|_2 \left\| \left(\sum_{l=1}^{n_l} G_l(\mathbf{z}, X) L_l \right)^{-1} \right\|_2 \|\mathbf{R}_v(\mathbf{z})\|_2 \,, \tag{2.18}$$

where $\|\cdot\|_2$ *is the vector 2-norm or its induced matrix norm.*

Corollary 3.1 states an upper bound for the error in the surrogate in terms of the residual and the discretised differential operator, which gives an indication where the surrogate $\tilde{\mathbf{q}}$ deviates significantly from the exact solution. Equation (2.18) holds for any induced matrix norm. In this chapter we adopt the 2-norm. For specific types of PDEs, e.g. elliptic PDEs, stricter error bounds can be formulated [50].

3.3 Refinement measure for adaptive sampling

The goal is to sample the QoI in the random space $I_{\mathbf{z}}$ such that we can construct an accurate surrogate model for the QoI by means of polynomial interpolation. Theorem 1 shows that a suitable refinement measure for a linear underlying PDE is given by

$$\mathbf{R}^*(\mathbf{z}) = Q \left(\sum_{l=1}^{n_l} G_l(\mathbf{z}, X) L_l \right)^{-1} \mathbf{R}_v(\mathbf{z}) \,. \tag{2.19}$$

If samples are placed such that $\mathbf{R}^*$ converges to zero, then the error in the surrogate also converges to zero. Using polynomial interpolation for the approximations $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$ in $\mathbf{R}_v$ ensures that $\mathbf{R}_v$ is close to zero in the neighbourhood of the interpolation samples. The quantity $\left(\sum_{l=1}^{n_l} G_l(\mathbf{z}, X) L_l\right)^{-1}$ is a function of $\mathbf{z}$, but does not depend on the choice of interpolation samples and therefore acts as a scaling function for the residual $\mathbf{R}_v$. This justifies using a greedy approach for placing new samples as follows:

$$\mathbf{z}_{\text{new}} = \arg\max_{\mathbf{z} \in I_{\mathbf{z}}} \|\mathbf{R}^*(\mathbf{z})\|_2 \,. \tag{2.20}$$

The construction of $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$ in $\mathbf{R}_v$ is a crucial step in the NIPPAS method and is therefore discussed in detail, in section 4. However, when using $\mathbf{R}^*$ as a refinement measure, there are two issues:

- The term $\left(\sum_{l=1}^{n_l} G_l(\mathbf{z}, X) L_l\right)^{-1}$ is a-priori unknown and expensive to compute.
- Equation (2.19) holds for linear PDEs only.

The inverse of the full discretisation matrix is unknown in general and cannot be used for adaptive sample placement, as it is expensive to compute as a function of $\mathbf{z}$. However, as this term only acts as a scaling function, omitting this term may result in a compact refinement measure for the quantities that are used to compute the QoI. This particular residual is then computed as follows:

$$\mathbf{R}(\mathbf{z}) := Q\mathbf{R}_v(\mathbf{z}) \,. \tag{2.21}$$

Notice that in comparison to $\mathbf{R}^*$, $\mathbf{R}$ can be computed for both linear and non-linear PDEs. Intuitively, we might expect that the refinement measure $\mathbf{R}(\mathbf{z})$ produces accurate and stable interpolants in combination with the greedy sample placement, because large errors and/or instabilities in the surrogate would lead to large errors in the residual, which then triggers new sample placement due

to the greedy approach. A commonly used quantity that is used to indicate the quality of a set of sample locations is the Lebesgue constant [54]. We note that our proposed greedy approach does not necessarily result in sample locations with an optimal Lebesgue constant, but effectively uses the model information to sample regions of interest, similar to other approaches [40, 42, 45, 55].

To summarise, $\mathbf{R}$ is our proposed refinement measure, as it is less expensive to compute and more generally applicable when compared to $\mathbf{R}^*$. A numerical comparison for both refinement measures is given in section 6 for a case where $\mathbf{R}^*$ can still be computed. Furthermore, the effectiveness of using $\mathbf{R}$ as a refinement measure for the case of non-linear PDEs is demonstrated numerically in section 6.

3.4 Incorporating PDF in refinement measure

Although using the residual $\mathbf{R}(\mathbf{z})$ as a refinement measure for adaptive sampling may result in stable and accurate surrogate models, the sampling procedure may put too many resources in areas that contribute little to the statistical quantities. Such areas are of little interest when we compute statistical quantities like the mean

$$\mathbb{E}[\mathbf{u}] := \int_{I_\mathbf{z}} \rho(\mathbf{z})\mathbf{q}(\mathbf{z})\mathrm{d}\mathbf{z} \,. \tag{2.22}$$

These statistical quantities are calculated by weighing the QoI with the PDF and integrating the resulting quantity over the parameter space $I_\mathbf{z}$. Areas of $I_\mathbf{z}$ where both the PDF and QoI are low, contribute little to the integral in (2.22) and therefore an alternative refinement measure is proposed, which is especially suited when one is interested in accurate statistical quantities rather than an accurate surrogate model. The proposed refinement measure is based on the following theorem:

Theorem 2. *Assume the following:*

- *The conditions of theorem 1 are satisfied, such that* (2.12) *holds.*
- $\mathbb{E}[\tilde{\mathbf{q}}]$ *and* $\mathbb{E}[\mathbf{q}]$ *are finite.*

Then the following holds:

$$\|\mathbb{E}[\tilde{\mathbf{q}}-\mathbf{q}]\|_2 \leq vol(I_\mathbf{z}) \sup_{z\in I_\mathbf{z}} s(\mathbf{z})\rho(\mathbf{z})\|Q\|_2\|\mathbf{R}_v(\mathbf{z})\|_2 \,, \tag{2.23}$$

where

$$s(\mathbf{z}) = \left\| \left(\sum_{l=1}^{n_l} G_l(\mathbf{z},X)L_l \right)^{-1} \right\|_2 \,. \tag{2.24}$$

The proof of the theorem follows the proof of theorem 1 and bounds the integral by multiplying the domain volume with a bound for the integrand. The full proof is not shown to avoid repetition.

Theorem 2 states a relation between the residual and the error in the mean of the QoI. The theorem shows that a proper refinement measure again includes the inverse of the full differentiation matrix. As stated before, this value is unknown in general as a function of $\mathbf{z}$ and is expensive to sample. Therefore the proposed refinement measure for calculating statistical quantities is given by

$$\mathbf{R}_\rho(\mathbf{z}) := \rho(\mathbf{z})Q\mathbf{R}_v(\mathbf{z}) = \rho(\mathbf{z})\mathbf{R}(\mathbf{z}) \,, \tag{2.25}$$

and new samples are placed using the greedy approach

$$\mathbf{z}_{\text{new}} = \arg\max_{\mathbf{z}\in I_\mathbf{z}} \|\mathbf{R}_\rho(\mathbf{z})\|_2 \,. \tag{2.26}$$

Refinement measure (2.25) will not place new samples in parts of the random space where the PDF is zero, but is still able to place samples at "rare events" in the random space, i.e., parts where the PDF is small but where the PDE residual is large.

A comparison of the refinement measures (2.21) and (2.25) for convergence of both surrogate construction and statistical quantity calculation is made in section 6.

3.5 Overview of method

Residual definitions (2.21) and (2.25) are used in the NIPPAS method to adaptively refine a surrogate model $\tilde{\mathbf{q}}(\mathbf{z})$. In detail, the NIPPAS method comprises the following steps:

1. initial sample placement
2. refinement loop:
   (a) compute residual $\mathbf{R}(\mathbf{z})$ or $\mathbf{R}_\rho(\mathbf{z})$
   (b) check stopping criterion
   (c) find $\mathbf{z}_{\text{new}} = \arg\max_{\mathbf{z}\in I_\mathbf{z}} \|\mathbf{R}(\mathbf{z})\|_2$ or $\|\mathbf{R}_\rho(\mathbf{z})\|_2$
   (d) sample model at $\mathbf{z}_{\text{new}}$
3. interpolate resulting samples

A schematic representation of the methodology is shown in figure 2.1. The choice of using either refinement measure (2.21) or (2.25) depends on whether the interest is in constructing an accurate surrogate or calculating accurate statistical quantities. The individual steps of the proposed surrogate model construction are discussed in detail in the next section. Before discussing the NIPPAS method steps, the connection is shown between the NIPPAS method and a general empirical interpolation method.

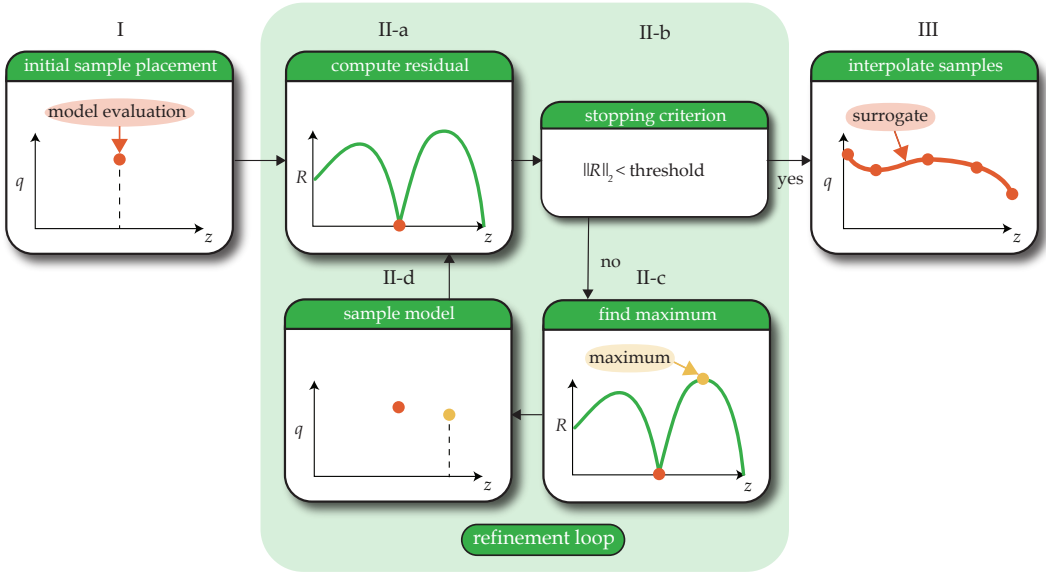


Figure 2.1: *Schematic representation of the methodology.*

The refinement measures were derived from theory for linear PDEs. We will show in section 6 that the proposed refinement measures also work for a non-linear underlying PDE. Stable polynomial interpolating surrogates show clustering samples at the boundary of the domain [54]. Even though stability is not proven for the proposed method, it follows intuitively from using the greedy approach in combination with the residual. If samples do not cluster at the boundaries, then the approximations used to calculate the residual become unstable. These unstable interpolants produce large errors near the boundary of the domain because of the Runge-phenomenon. This results in large residual values, which leads to sample placement near the boundary of the domain. Lastly, note that the sample locations from the NIPPAS method are in general scattered, which complicates constructing an interpolant on these samples. This aspect will be discussed in more detail in section 5

3.6 Connection with empirical interpolation

Our NIPPAS method has similarities to the classical empirical interpolation procedure [42, 44, 45, 50]. The main difference between general empirical interpolation and the NIPPAS method is the way in which the residual is constructed. In empirical interpolation the residual is based on the entire spatial/temporal surrogate $\tilde{\mathbf{v}}$, rather than on the QoI $\tilde{\mathbf{q}}$. Consequently, the residual in empirical interpolation is defined as

$$\mathbf{R}_{EI}(\mathbf{z}) = \sum_{l=1}^{n_l} G_l(\mathbf{z},X) L_l(\widetilde{\mathbf{v}}(\mathbf{z})) - \mathbf{S}(\mathbf{z},X) \ , \tag{2.27}$$

which has the advantage that one only needs to approximate a single term, i.e., $\tilde{\mathbf{v}}(\mathbf{z})$, instead of constructing several approximations $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$. However, the disadvantage is that the refinement measure $\mathbf{R}_{EI}$ adds a degree of intrusiveness as it assumes that the operators L_l are known completely [43], which is not always the case and restricts the applicability.

An advantage of the NIPPAS method is that it focuses the sample placement on regions where $\tilde{\mathbf{q}}$ needs to be refined, rather than where $\tilde{\mathbf{v}}$ needs to be refined. The NIPPAS method therefore focuses on areas in the parameter space that are most relevant for constructing an accurate surrogate for $\mathbf{q}$, which results in faster convergence. The disadvantage is that the NIPPAS method requires the observation of calculated quantities in the PDE-solver, i.e., calculated partial derivatives, which will be discussed in section 4.

4 Algorithm in detail

The Non-Intrusive PDE/PDF-informed Adaptive Sampling (NIPPAS) method when using refinement measure $\mathbf{R}$ is shown in this section. The procedure when using $\mathbf{R}_\rho$ is analogous. A schematic overview of the NIPPAS method is shown in figure 2.1.

I. Placement of one random initial sample

The adaptive algorithm starts by performing initial sampling in the random space. Multiple initial sample configurations can be considered. It is advised to start the method with a single Monte-Carlo sample in the random space. Other configurations that initially place multiple samples in the random space are also possible. However, this introduces additional degrees of freedom in the initialisation phase, which are in principle unneeded, as the method is perfectly capable of choosing proper sample locations from the start with only one initial sample. This initialisation is used in the remainder of this chapter. The initial sample location and corresponding model evaluation is

denoted as $(Z_0, V_{Q,0}) = (z_0, \mathbf{q}(z_0))$. More generic, a subscript i indicates the number of adaptively placed samples, i.e., $Z_i = \{\mathbf{z}_j\}_{j=0}^{j=i}$ and $V_{Q,i} = \{\mathbf{q}(\mathbf{z}_j)\}_{j=0}^{j=i}$.

II-a. Non-intrusive computation of residual

The refinement loop starts with computing the residual (2.21).

The residual requires the approximations $\widetilde{L_l(\mathbf{v}(\mathbf{z}))}$. These approximations can be easily constructed for a given sample set Z_i by applying an interpolation operator P (to be discussed in section 5.1) on the values $\{L_l(\mathbf{v}(\mathbf{z}_i))\}_{j=0}^{i}$. *However, the gray-box solver in general only returns the solution values $\mathbf{v}(\mathbf{z}_i)$, and should be altered such that it also returns the values of the individual discretised differential operators $L_l(\mathbf{v}(\mathbf{z}_i))$. These discretised partial derivatives are used to compute the solution values within the solver and may be output alongside the solution when sampling PDE solutions.* This requires only a slight alteration of the solver, without changing the underlying PDE, and therefore keeps the methodology non-intrusive. To summarise, the approximations of the differential operator terms are given by

$$L_l(\mathbf{v}(\mathbf{z})) \approx \widetilde{L_l(\mathbf{v}(\mathbf{z}))} = P[(Z_i, (L_l V)_i] \ , \ \ l = 1, ..., n_l \ , \tag{2.28}$$

where

$$(L_l V)_i = \{L_l(\mathbf{v}(\mathbf{z}_0)), ..., L_l(\mathbf{v}(\mathbf{z}_i))\} \ , \ \ l = 1, ..., n_l \ , \tag{2.29}$$

which are the values that should be returned by the gray-box solver alongside the solution values $\mathbf{v}$. After approximating each differential operator term in the random space through interpolation, the interpolants are substituted into (2.21), which returns a function in the variable $\mathbf{z}$. Notice that if the gray-box solves (2.5) with negligible round-off and iteration error, then we satisfy

$$\mathbf{R}(\mathbf{z}_j) = 0, \ \ j = 0, ..., i \ , \tag{2.30}$$

which attains local maxima between the samples, as is shown in figure 2.1.

II-b. Stopping criterion based on the residual

The refinement loop has to be terminated after a number of iterations. For this purpose we need a stopping criterion, which reflects the quality of the surrogate model. Equation (2.18) can be used as a stopping criterion. However, (2.18) does not hold for non-linear PDEs and computing the required norms would be intractable in general. Ideally, the residual will show a similar convergence as the error in the surrogate. If this is the case, the magnitude of the residual is not only useful to adaptively sample the gray-box, but it also serves as a reliable indication for the quality of the surrogate. Therefore, the refinement loop is stopped when the following criterion is met:

$$\|\mathbf{R}(\mathbf{z})\|_2 < \varepsilon \ , \tag{2.31}$$

where ε is a specified threshold. This norm is approximated using Monte Carlo sampling.

II-c. Find the location where the residual attains maximum

If criterion (2.31) is not met, the surrogate is refined by placing an extra sample according to (2.21). In other words, we need to solve the following global optimisation problem in $I_{\mathbf{z}}$:

$$\mathbf{z}_{\max} = \arg\max_{\mathbf{z} \in I_z} \|\mathbf{R}(\mathbf{z})\|_2 \ . \tag{2.32}$$

The complexity of this global optimisation problem depends on characteristics of the objective function $\mathbf{R}(\mathbf{z})$. The residual is in general not smooth and has multiple local maxima. Therefore, in order to solve (2.32), a particle swarm method from the Global optimisation toolbox in Matlab is used, which is able to solve non-smooth global optimisation problems.

The solution of the global optimisation problem (2.32) is denoted $\mathbf{z}_{\max}$ and is used in the next step to refine the surrogate.

II-d. Sample the model at the new sample location

To refine the surrogate model, we add the new sample $\mathbf{z}_{\max}$ to our current sample set Z_i:

$$Z_{i+1} = Z_i \cup \mathbf{z}_{\max} \ , \tag{2.33}$$

$$V_{Q,i+1} = V_{Q,i} \cup \mathbf{q}(\mathbf{z}_{\max}) \ , \tag{2.34}$$

$$(L_l V)_{i+1} = (L_l V)_i \cup L_l(\mathbf{v}(\mathbf{z}_{\max})) \ , \quad l = 1, \ldots, n_l \ . \tag{2.35}$$

The new sample set $(L_l V)_{i+1}$ is suited for constructing an improved approximation $\widetilde{L_l(\mathbf{v})}$, see equation (2.28), which is used in the next iteration of the refinement loop.

III Final surrogate is constructed by interpolation

After the stopping criterion (2.31) has been met, the refinement loop terminates and we end up with a sample set Z_n and an evaluation set Q_n. In order to construct the final surrogate, we apply the interpolation operator P on the final sample set, leading to

$$\widetilde{\mathbf{q}}(\mathbf{z}) = P[(Z_n, V_{Q,n})] \ , \tag{2.36}$$

where $\widetilde{\mathbf{q}}$ is expected to be an accurate approximation to $\mathbf{q}$. If wanted, statistical quantities like (2.22) can be computed by using this surrogate to compute the desired integrals.

5 Implementation details

In this section some implementation details are discussed.

5.1 Surrogate modelling by polynomial interpolation for scalar QoIs

Interpolation for a scalar QoI is discussed here. Interpolation for a vector QoI is achieved by applying the interpolation operator to each element of the QoI vector individually.

Assume we have a sample set $Z = \{\mathbf{z}_i\}_{i=0}^N$ and corresponding QoI values $V_Q = \{q(\mathbf{z}_i)\}_{i=0}^N$. As mentioned in section 4, we denote by P the interpolation operator which acts on the set (Z, V_Q). Polynomial interpolation aims to find a polynomial $\tilde{q}$, such that:

$$\tilde{q}(\mathbf{z}_i) = P[(Z, V_Q)](\mathbf{z}_i) = q(\mathbf{z}_i) \ . \tag{2.37}$$

The polynomial $\tilde{q}$ serves as a surrogate for $q(\mathbf{z})$. Finding $\tilde{q}$ is straightforward in a 1D random space, but interpolation in multi-dimensional random spaces is not unique, and the result is influenced by the choice of interpolation basis. Additionally, the sample locations from our adaptive sampling are scattered, which further complicates the interpolation procedure. In order to make the interpolation basis unique, graded lexicographic ordered Chebyshev polynomials $\{\phi_i(\mathbf{z})\}_{i=0}^N$ are used, which are

known for their stability when using them for interpolation [54]. The interpolant is found by solving a Vandermonde system [53]:

$$\underbrace{\begin{pmatrix} \phi_0(\mathbf{z_0}) & \dots & \phi_N(\mathbf{z_0}) \\ \vdots & & \vdots \\ \phi_0(\mathbf{z_N}) & \dots & \phi_N(\mathbf{z_N}) \end{pmatrix}}_{A} \begin{pmatrix} c_0 \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} q(\mathbf{z}_0) \\ \vdots \\ q(\mathbf{z}_N) \end{pmatrix} , \tag{2.38}$$

which results in the interpolant

$$\tilde{q}(\mathbf{z}) = \sum_{i=0}^{N} c_i \phi_i(\mathbf{z}) . \tag{2.39}$$

The Vandermonde matrix A can become ill-conditioned when increasing the number of samples [56], which makes solving (2.38) difficult. Nevertheless, other approaches that circumvent solving (2.38) [14, 57, 58] do not have the flexibility to reuse calculations done in the previous iterations for solving the larger system at the current iteration. This leads to a significant increase in computational expense as the adaptive sampling placement is done iteratively. Therefore, Vandermonde interpolation is used in the NIPPAS method, and a robust procedure for solving the possibly ill-conditioned system (2.38) is used.

Several methods exist for solving ill-conditioned systems of equations: regularisation [59], singular value decomposition [60], pivoted LU-factorisation [61], and pseudo-inversion [62]. In practice, computing the pseudo-inverse is often not advised due to its computational cost, which is $O(N^3)$. However, our adaptive sampling algorithm requires the solution of multiple linear systems with the same Vandermonde matrix A at each iteration, which makes the use of a pseudo-inverse beneficial. Nevertheless, computing the full pseudo-inverse at each iteration would be inefficient and therefore Greville's algorithm [63] is used to update the pseudo-inverse after adding a new row/column to A. Consequently, interpolating polynomials can be constructed without solving the full linear system (2.38) and without computing the full pseudo-inverse each time the interpolation operator P is applied. The implementation of Greville's algorithm is discussed in more detail in the next subsection.

5.2 Rank-one update of pseudo-inverse

As mentioned before, when adding a new sample to the sample set, an extra row and column are appended to the existing Vandermonde matrix (2.38). Therefore, a new pseudo-inverse needs to be determined for this new Vandermonde matrix [63]. Instead of calculating the new entire inverse, Greville's algorithm is used to iteratively compute the pseudo-inverse of a given matrix A.

Assume we have a matrix $A \in \mathbb{R}^{N \times N}$ and its corresponding pseudo-inverse G. When a column is added to A, i.e.,

$$A' = [A \; a] , \tag{2.40}$$

then Greville's algorithm computes the pseudo-inverse of the new matrix A' recursively. In more detail, the new pseudo-inverse G' of A' is given by [64]

$$G' = \begin{pmatrix} G - db^T \\ b^T \end{pmatrix} , \tag{2.41}$$

where

$$a^{(1)} = AGa\ ,\quad a^{(2)} = a - a^{(1)}\ , \tag{2.42}$$

$$d = Ga\ , \tag{2.43}$$

$$b^T = \begin{cases} \frac{\left(a^{(2)}\right)^H}{\left(a^{(2)}\right)^H a^{(2)}}\ , & \text{if } a^{(2)} \neq \mathbf{0}\ , \\ (1 + d^T d)^{-1} d^T G\ , & \text{if } a^{(2)} = \mathbf{0}\ , \end{cases} \tag{2.44}$$

where the H denotes the conjugate transpose. Notice that our refinement loop adds both a row and a column to the Vandermonde matrix (2.38), each time a new sample is added. Therefore the procedure described above has to be performed twice, first adding a column $A' = [A\ c]$, then adding a column r^T to the transpose of the resulting matrix, i.e., $(A'')^T = ((A')^T\ r^T)$.

A recursive algorithm for calculating the pseudo-inverse is necessary to reduce the algorithmic complexity of the adaptive sampling procedure. For an $N \times N$ matrix, Greville's algorithm performs an inverse update with complexity $O(8N^2)$, while computing the full inverse directly has complexity $O(N^3)$ [63] and becomes infeasible quickly in high dimensional spaces $I_{\mathbf{z}}$, where the number of samples required for constructing an accurate surrogate is high.

5.3 Algorithmic complexity scales well for high dimensions

The algorithmic complexity gives an estimate on how the computational effort scales with the number of samples and the dimension of the random space.

- The computational complexity is determined by the complexity of the individual parts; updating pseudo-inverse, interpolation, and particle swarm optimisation. First we discuss the algorithmic complexity of a single iteration in the refinement loop.
- Updating the pseudo-inverse is achieved by using Greville's algorithm. Assume a row and column are added to an existing pseudo-inverse of dimensions $i \times i$. Using Greville's algorithm, this can be done in $O(8i^2)$ operations [63] (discussed in section 5).
- The interpolation procedure is fairly cheap when the pseudo-inverse is available. In detail, when interpolating on i sample locations, solving (2.38) only requires a matrix-vector multiplication of $O(i^2)$ operations.
- The complexity of the particle swarm optimisation is determined by the number of particles used and the maximum number of iterations. In our case we use a particle swarm of N_p particles and a maximum of N_{it} iterations. Typical values for N_p and N_{it} are $100\dim(I_{\mathbf{z}})$ and $200\dim(I_{\mathbf{z}})$, respectively. In the worst case, the number of maximum iterations is reached and we have to evaluate the residual at $N_p N_{it}$ locations, without the guarantee to have found an optimum. Moreover, if we have i samples, then the residual is a polynomial of degree $i-1$ and can be evaluated with Horner's method [56] in $O(i)$ operations. This results in a total cost of the particle swarm optimisation for a single iteration of the surrogate construction of $O(i\dim(I_{\mathbf{z}})^2 n_l)$, where n_l is the number of terms in the PDE (2.5).

The total cost of each iteration in the refinement loop is now given by:

$$O(\underbrace{i^2}_{\text{Greville}} + \underbrace{i^2}_{\text{interpolation}} + \underbrace{i\dim(I_{\mathbf{z}})^2 n_l}_{\text{particle swarm optimisation}})\ . \tag{2.45}$$

Hence, if we run the refinement loop N times, a conservative upper bound for the total algorithm complexity becomes

$$O(N^3 + N^2 \dim(I_{\mathbf{z}})^2 n_l)\ . \tag{2.46}$$

Notice that the complexity depends quadratically on the dimension of the random space. However, the number of samples necessary for achieving a specified accuracy in higher-dimensional random space will generally increase with the number of dimensions, and computational cost will therefore increase faster than quadratic with the dimension of the random space.

6 Results

In this section we present multiple examples that illustrate the efficiency of our method. In order to study and compare convergence properties, two error measures are defined, one for the error in the statistical quantities, i.e., mean, variance, etc., and one for the error in the surrogate.

The error in the k-th statistical moment is the 2-norm of the difference vector between the exact and the approximate statistical moment as computed in (2.22):

$$e_{\rho}^{(k)} := \|\mathbb{E}[\mathbf{q}^k - \tilde{\mathbf{q}}^k]\|_2 = \left\| \int_{I_\mathbf{z}} \rho(\mathbf{z})(\mathbf{q}^k(\mathbf{z}) - \tilde{\mathbf{q}}^k(\mathbf{z}))d\mathbf{z} \right\|_2 , \tag{2.47}$$

where the integral is calculated using numerical integration with negligible error, i.e., a tensor based Gauss quadrature rule with 100 nodes in each direction. The uncertainties are assumed to be uniformly distributed, $\rho(\mathbf{z}) = 1$, unless stated otherwise.

Secondly, the error in the surrogate is computed as follows:

$$e := \frac{1}{N_{MC}} \sum_{i=1}^{N_{MC}} \|\mathbf{q}(\mathbf{z}_i) - \tilde{\mathbf{q}}(\mathbf{z}_i)\|_2 , \tag{2.48}$$

where the sum is taken over a large number of uniform Monte-Carlo samples (N_{MC}) in the random space.

The first two test-cases consider a steady and unsteady advection-diffusion equation, respectively. They demonstrate the difference between our approach and conventional empirical interpolation, effect of discretisation on error convergence, difference between refinement measures (2.21) and (2.25), and applicability to non-hypercube domains. The third and last test-case considers the non-linear shallow water equations and demonstrates how our method can be used in combination with a new state-of-the-art neural network based PDE solver.

6.1 Steady-state advection-diffusion equation

We first consider a test-case such that the assumptions in theorem 1 hold and we study the following:

- Comparison of refinement measure (2.21) and (2.19).
- Asymptotic sample distribution.
- Comparison between the NIPPAS method and conventional empirical interpolation.

The underlying PDE is the dimensionless steady state advection-diffusion problem given by:

$$Re(z)v_x - v_{xx} = 0 \ , \ v(0,z) = 0, \ v(1,z) = 1, \ x \in [0,1] \ , \tag{2.49}$$

where $Re(z)$ is the Reynolds number, which is assumed to be uncertain and a function of z. The equations are discretised using a finite-difference approach on an equidistant grid with a resolution of Δx with N_{PDE} grid points. The solution vector on the computational grid $\mathbf{v}(z) \approx v(\mathbf{x})$, with $x_i = i\Delta x$ for $i = 1, ..., N_{PDE}$, is obtained by solving the following linear system:

$$Re(z)L_1\mathbf{v} - L_2\mathbf{v} = \mathbf{S}(z) \ , \tag{2.50}$$

where

$$L_1 = \frac{1}{2\Delta x}\begin{pmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & 0 & -1 & 0 \end{pmatrix}, \; L_2 = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & 0 & 1 & -2 \end{pmatrix} \tag{2.51}$$

$$\mathbf{S}(z) = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ -\frac{Re(z)}{2\Delta x} - \frac{1}{\Delta x^2} \end{pmatrix}, \tag{2.52}$$

where the boundary conditions enter the discretised equation via the vector $\mathbf{S}$. Notice that the discretised PDE satisfies the assumptions of theorem 1. Example solutions for different Reynolds numbers are shown in figure 2.2.

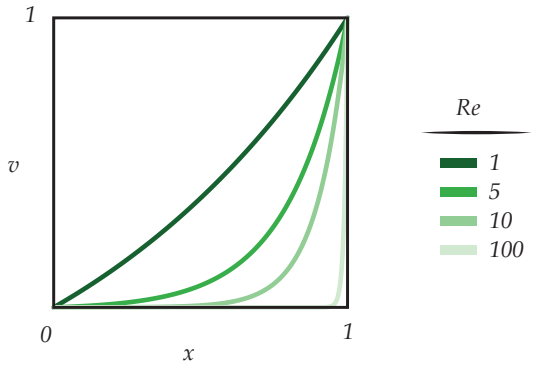


Figure 2.2: *Example solutions for different Reynolds number. The solution are computed on a computational grid with* $\Delta x = 10^{-3}$.

In order for the discretisation to produce stable results, the cell Reynolds number $Re_{\Delta x} = Re\Delta x$ should satisfy $Re_{\Delta x} < 2$, which is satisfied by picking Δx sufficiently small, which is $\Delta x = 10^{-3}$ in our case ($N_{PDE} = 1000$).

***There is no significant difference in convergence between refinement measures* R *and* R***

The difference between refinement measures $\mathbf{R}$ and $\mathbf{R}^*$ is the incorporation of the scaling term $\left(\sum_{l=1}^{n_l} G_l(z,X)L_l\right)^{-1}$ in $\mathbf{R}^*$. This scaling term is in general expensive to compute as a function of z and is therefore often infeasible to incorporate in the refinement measure. However, for this simple test-case we can compute this scaling term as a function of z and are able to study its effect on sample placement. In order to compare the two refinement measures $\mathbf{R}$ (2.21) and $\mathbf{R}^*$ (2.19), three different functional forms for $Re(z)$ are used, where z is uniformly distributed between $[0,1]$. The three different functional forms are given by:

$$Re_1(z) = 99z + 1\,, \tag{2.53}$$

$$Re_2(z) = 10^{2z}\,, \tag{2.54}$$

$$Re_3(z) = 10^{-2(z-1)}\,, \tag{2.55}$$

and are shown in figure 2.3. All three functions range from 1 to 100 for $z \in [0,1]$, but have completely different shapes, which influence the scaling function and therefore the sample locations when using refinement measure $\mathbf{R}^*$.

A comparison between the two refinement measures $\mathbf{R}$ and $\mathbf{R}^*$ is made by choosing $Q = I$, i.e., we are interested in the entire solution $\mathbf{q} = \mathbf{v}$ as a function of z. The adaptive sample placement starts by placing a single uniformly distributed sample in the random space. The solution values and derivative values $\mathbf{v}_x := L_1\mathbf{v}$ and $\mathbf{v}_{xx} := L_2\mathbf{v}$ are sampled at the sample location. These values are used to construct the interpolants which are required for computing $\mathbf{R}_v$. The convergence of the error in the surrogate $\tilde{\mathbf{v}}(z)$ for both refinement measures and the different Re_i is shown in figure 2.3.

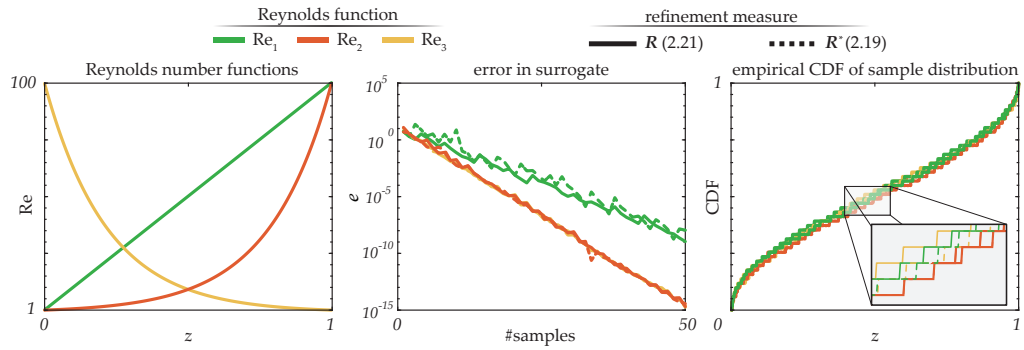


Figure 2.3: *The results for the steady-state advection-diffusion equation for the two refinement measures* $\mathbf{R}$ *and* $\mathbf{R}^*$ *and three different Reynolds functions in the random space. The solutions are computed using* $\Delta x = 10^{-3}$.

The error in the surrogate converges exponentially fast to machine precision for both refinement measures without any significant difference. Samples are placed at similar locations for both refinement measures, which is shown in the empirical CDF plot in figure 2.3. Regarding stability of the interpolant, a clustering of samples occurs at the boundaries of the random space, which stimulates stable interpolation.

Refinement measure $\mathbf{R}^*$ uses all the terms that depend on z and should be optimal for sample placement, see equations (2.12) and (2.19). However, the error in the surrogate is not only determined by sample placement, but also by the polynomial interpolation procedure. Consequently the best refinement measure does not necessarily lead to the best convergence in the error. Including the scaling term in $\mathbf{R}^*$ appears to have little effect on the sample placement, at least for this simple test-case. In fact, for this test-case one could refine directly on $\|\tilde{\mathbf{q}} - \mathbf{q}\|_2$ (because $\mathbf{q}$ is explicitly known), and this gives the same results as refining based on $\mathbf{R}^*$. As there is no significant difference between the two refinement measures, we use refinement measure $\mathbf{R}$ in the remainder of this chapter, as it is more generally applicable.

NIPPAS method converges faster than conventional empirical interpolation

We compare convergence of the surrogate for the NIPPAS method based on (2.21) with the conventional empirical interpolation based on (2.27). The main difference (in absence of incorporation of the PDF) is the fact that empirical interpolation places new adaptive samples based on a residual which is based on the entire solution $\mathbf{v}$, whereas the NIPPAS method uses a residual based only on the QoI $\mathbf{q}$. To compare both methods, the QoI is set to $q = (\mathbf{v})_{500}$ (the solution computed at the middle of the computational grid, with $Q = (0,...,0,1,0,...,0)$), and the Reynolds number is given by $Re_1(z)$. The convergence comparison is shown in figure 2.4.

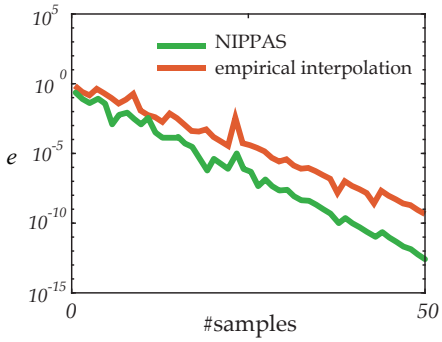


Figure 2.4: *Convergence comparison for the NIPPAS method and conventional empirical interpolation for the steady-state advection-diffusion equation.*

The NIPPAS method enhances the surrogate by focusing on locations that are relevant for the QoI. This leads to faster convergence for the NIPPAS method when compared to conventional empirical interpolation. However, in case more solution values from the solution vector $\mathbf{v}$ would be incorporated, i.e., more non-zero entries in the columns of Q, the convergence speed-up will decrease and eventually the convergence coincides with the one from empirical interpolation when the QoI uses the entire solution vector $\mathbf{v}$.

6.2 Unsteady advection-diffusion equation

In the previous example we applied the NIPPAS method to a steady-state PDE for a 1D random space in order to compare refinement measures and to compare with conventional empirical interpolation. In this section we study the following:

- The effect of the discretisation method applied to the PDE.
- The accuracy of the NIPPAS method for approximating statistical moments in a 2D random space.
- The construction of a surrogate model on non-hypercube domains.

Therefore, we thoroughly study the NIPPAS method for an unsteady advection-diffusion equation with two parameter uncertainties. We start with a 2D hypercube random space and then gradually increase the complexity of the test-case.

The underlying PDE is the 1D advection-diffusion equation, given by

$$v_t + z_1 v_x = z_2 v_{xx} \ , \tag{2.56}$$

where the advection parameter z_1 and the diffusion parameter z_2 are assumed to be uncertain and uniformly distributed between $[0, 2\pi]$. For the problem (2.56) to be well-posed, initial and boundary conditions are required. A spectral spatial discretisation method [1] is used for solving (2.56) and the boundary conditions are taken periodic for $x \in [0, 2\pi]$ and the initial condition is given by

$$v(x,0) = v_0(x) = \sin(x) \ . \tag{2.57}$$

The spectral spatial discretisation is performed on an equidistant grid $x_i = (2\pi i)/N_x$ for $i = 0, ..., N_x$,

where $N_x = 256$. This results in a solution vector $\mathbf{v}(t)$:

$$\mathbf{v}(t) = \begin{pmatrix} v_0(t) \\ \vdots \\ v_{N_x}(t) \end{pmatrix} , \tag{2.58}$$

where $v_i(t)$ is the approximate solution at grid point x_i. Additionally, taking derivatives can be written in terms of a matrix-vector multiplication

$$v_x \approx D_x \mathbf{v} , \tag{2.59}$$

$$v_{xx} \approx (D_x)^2 \mathbf{v} , \tag{2.60}$$

where D_x is the spectral differentiation matrix [1]. As a result, the solution vector $\mathbf{v}(t)$ can be obtained by solving the semi-discrete problem

$$\mathbf{v}_t + z_1 D_x \mathbf{v} = z_2 D_x^2 \mathbf{v} , \tag{2.61}$$

which can be rewritten in the form (2.5) by discretising the time-derivative and formulating the resulting set of equations in matrix form. Notice that this results in a block-diagonal system.

A surrogate is created for the quantity $q(z_1, z_2) = v_{N_x}(1)$, which is the solution at the right boundary of the physical domain at $t = 1$.

Effect of discretisation method is small

As we use a spectral method with a fine resolution for the spatial discretisation and the problem is linear, the time integration error is expected to be dominant, and therefore the effect of the time discretisation method is studied. The semi-discrete problem (2.61) is solved using different time discretisation schemes, i.e., backward-Euler, Crank-Nicolson and fourth-order explicit Runge-Kutta (RK4), where we fix the time step at $\Delta t = 10^{-5}$.

To demonstrate the efficiency of the NIPPAS method, convergence is compared to an adaptive Smolyak approach [65]. The errors in the surrogate for both the NIPPAS method and the Smolyak solution are computed by using a Monte-Carlo reference solution based on 5000 samples. Notice that the reference solution changes, depending on the time discretisation. The reference solution for a Crank-Nicolson time discretisation is shown in figure 2.5. The choice of the time discretisation method affects the accuracy of the gray-box solver, and changes the shape of the surrogate slightly, as different time discretisations do not give the same QoI at the exact same location in random space. As a result, when changing the discretisation, a new reference solution has to be computed to study convergence. To further clarify, there is a difference between the exact/wanted surrogate, which is obtained by solving (2.1), and the discrete exact surrogate, obtained by solving the discretised equations (2.5). This difference is shown in the following equation:

$$\underbrace{\|q - \tilde{q}_{N_x}\|_2}_{\text{error w.r.t. exact solution}} \leq \underbrace{\|q - q_{N_x}\|_2}_{\text{discretisation error}} + \underbrace{\|q_{N_x} - \tilde{q}_{N_x}\|_2}_{\text{error w.r.t. discrete solution}} , \tag{2.62}$$

where q is the exact solution (solution of (2.1)), q_{N_x} is the discrete solution computed using $N_x + 1$ grid points (solution of (2.5)), and $\tilde{q}_{N_x}$ is the surrogate based on the discrete solutions. For this reason, we plot both the error in the surrogate with respect to the exact solution and the discrete solution. The results are shown in figure 2.6. The error with respect to the discrete solution converges to zero, with a convergence rate that is significantly faster than that for the adpative Smolyak sparse grid. Convergence is non-monotonic, which is common for adaptive sampling methods [40]. The error with respect to the exact solution stalls before machine precision is reached, which is due to

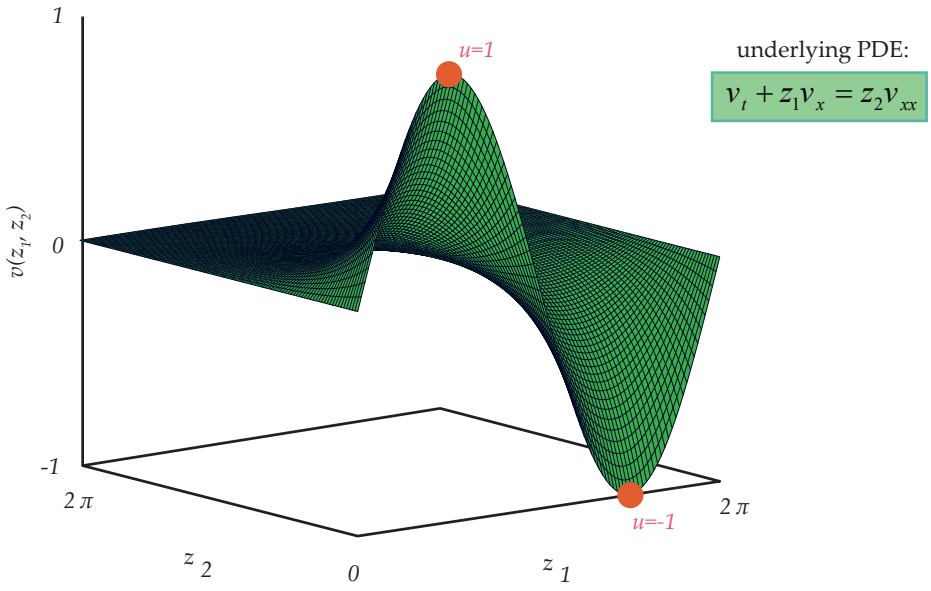


Figure 2.5: *Reference solution for the advection-diffusion equation for the quantity* $q(z_1, z_2) = v_{N_x}(1)$ *based on 5000 samples.*

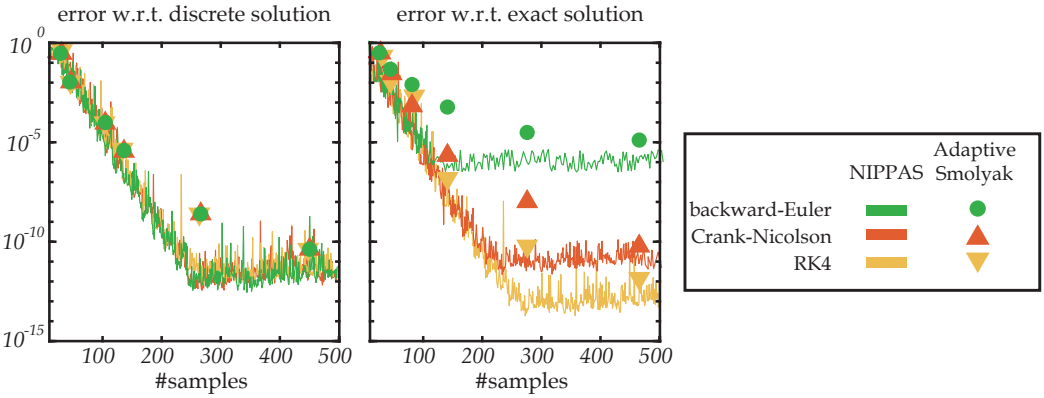


Figure 2.6: *Error* (2.48) *comparison between the Smolyak sparse grid surrogate and the NIPPAS surrogate for different time discretisation methods with* $\Delta t = 10^{-5}$ *and* $N_x = 256$*. The errors are plotted with respect to both the discrete solution of* (2.56) *and the exact solution of* (2.61).

the discretisation error. To clarify, the error with respect to the discrete solution converges to zero and equation (2.62) therefore states that the observed error after stalling is the discretisation error.

The convergence behaviour of the error with respect to the discrete solution is not dependent on the time discretisation, which indicates that the performance of our method does not depend on the underlying discretisation.

Faster convergence for statistical quantities with PDF weighing

Next, to study the effect of the two different refinement measures (2.21) and (2.25), we assume

independent β-distributed input uncertainties z_1 and z_2:

$$\rho(z_1,z_2) = c\cdot(2\pi z_1)^{\alpha_1}(2\pi z_2)^{\alpha_2}(1-2\pi z_1)^{\beta_1}(1-2\pi z_2)^{\beta_2}\ , \tag{2.63}$$

where (α_i,β_i) are parameters that characterise the PDF and the constant c is chosen such that the PDF has total probability 1 on $[0,2\pi]^2$. Convergence behaviour is studied as a function of the shape of the underlying PDF and therefore we vary the PDF parameters as follows:

$$\alpha_i \in \{1,2,3,4,5\},\ \beta_i \in \{1,2,3,4,5\},\ i=1,2\ , \tag{2.64}$$

which results in a total of 625 PDFs with totally different shapes. The convergence statistics of the mean, variance and the entire surrogate are shown in figure 2.7.

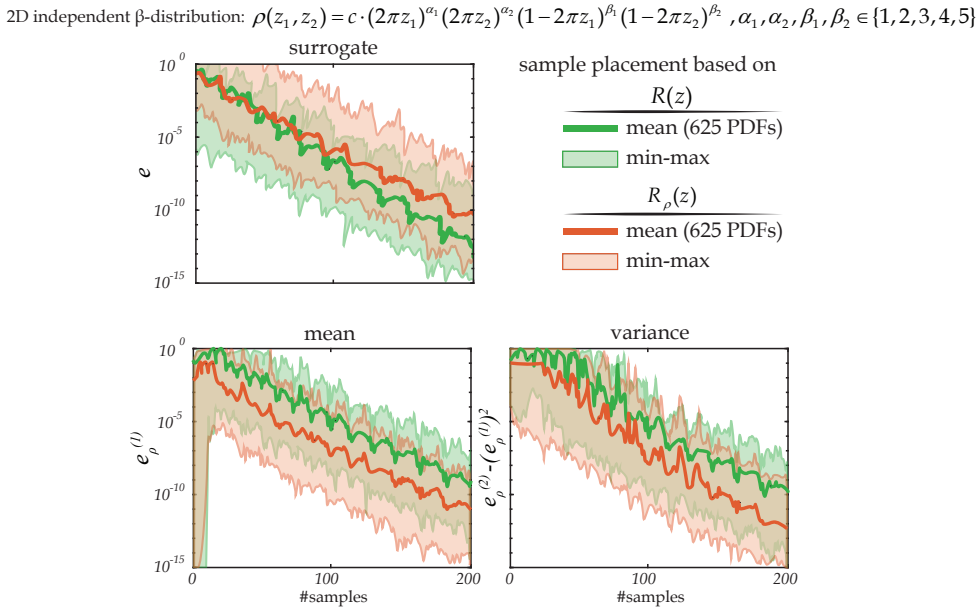


Figure 2.7: *Convergence in advection-diffusion surrogate, mean and variance for the two refinement measures* (2.21) *and* (2.25)*. The error in the surrogate is computed with* (2.48) *with 5000 Monte-Carlo samples. The errors in the mean and variance are calculated with* (2.47)*.*

Figure 2.7 shows the average convergence behaviour taken over the 625 different PDFs. From the figure we may conclude that weighing the residual with the PDF results in slower convergence for the surrogate model, see equation (2.48), but leads to faster convergence in both the mean and the variance, which are computed using (2.47). Furthermore, the variation in convergence over all 625 different PDFs, given by the shaded area around the mean line, has similar width for both refinement measures (2.21) and (2.25). The choice of refinement measures thus depends on the type of convergence the user requires, i.e., convergence in the surrogate or convergence in the statistical quantities. Notice that the shaded area for the convergence in the mean has a minimum that starts at 0 initially, which is due to the fact that for some PDFs the exact mean is zero, and as the initial sample is placed at a location in the random space where the response is also 0, we start with an approximate mean that is equal to the exact mean. The sample placement does not stop immediately, as the stopping criterion (2.31) is not met.

Method shows fast convergence for non-hypercube domains

The 2D β-distribution (2.63) just discussed, comprises two independent uncertainties and therefore the space in which we construct a surrogate is a hypercube. However, when dealing with dependent uncertainties, the space in which the surrogate is constructed, is in general not a hypercube, and many existing UQ methods fail when dealing with a complex random space.

To study the performance of the NIPPAS method on more general geometries, we assume a non-hypercube domain $I_{\mathbf{z}} \subset \mathbb{R}^d$ with associated uniform PDF $\rho(\mathbf{z})$. In order to sample the model on the domain $I_{\mathbf{z}}$, we restrict the residual (2.21) to $I_{\mathbf{z}}$ by multiplying it with an indicator function:

$$R_{I_{\mathbf{z}}}(\mathbf{z}) = R(\mathbf{z})\mathbb{I}_{I_{\mathbf{z}}}(\mathbf{z}) \ , \tag{2.65}$$

where

$$\mathbb{I}_{I_{\mathbf{z}}}(\mathbf{z}) = \begin{cases} 1, & \text{if } z \in I_{\mathbf{z}} \ , \\ 0, & \text{otherwise} \ . \end{cases} \tag{2.66}$$

After altering the residual definition slightly and placing an initial sample randomly in the non-hypercube random space, the NIPPAS method can be applied straightforwardly. Note that the basis functions are the same for the hypercube case, they are given by the Chebyshev polynomials defined on the smallest hypercube that comprises $I_{\mathbf{z}}$. In order to show the efficiency of our method for non-hypercube domains, we again construct a surrogate for the response shown in figure 2.5, but now on more complexly shaped domains. Three different geometries with different characteristics are chosen:

- sharp corners:

$$(z_2 \geq 0) \cap (z_2 - \sqrt{3}z_1 \leq 0) \cap (\sqrt{3}z_1 + z_2 \leq 2\pi\sqrt{3}) \ ,$$

- smooth boundary:

$$(z_1 - \pi)^2 + (z_2 - \pi)^2 \leq \pi^2,$$

- domain with holes:

$$\begin{aligned} &(z_1, z_2) \in [0, 2\pi]^2, \text{ but} \\ &\neg \left((z_1 - \pi)^2 + (z_2 - \pi)^2 \leq \frac{\pi^2}{4} \right) \cup \\ &\neg \left((z_1 - \frac{\pi}{3})^2 + (z_2 - \frac{\pi}{3})^2 \leq \frac{\pi^2}{25} \right) \cup \\ &\neg \left((z_1 - \frac{5\pi}{3})^2 + (z_2 - \frac{5\pi}{3})^2 \leq \frac{\pi^2}{9} \right) \ . \end{aligned}$$

The convergence results are shown in figure 2.8.
The results show an exponential convergence behaviour for all three different geometries, but the convergence rates slightly differ, which is likely caused by the choice of basis, which is suboptimal for all domains. Furthermore, weighing the residual with the PDF in case of non-uniformly distributed uncertainties is again possible, but results are similar to the results shown in figure 2.7 and are not shown to avoid repetition.

It has to be pointed out that the choice of a lexicographically ordered Chebyshev basis is far from optimal on non-hypercube domains. Nevertheless, good convergence behaviour is still achieved.

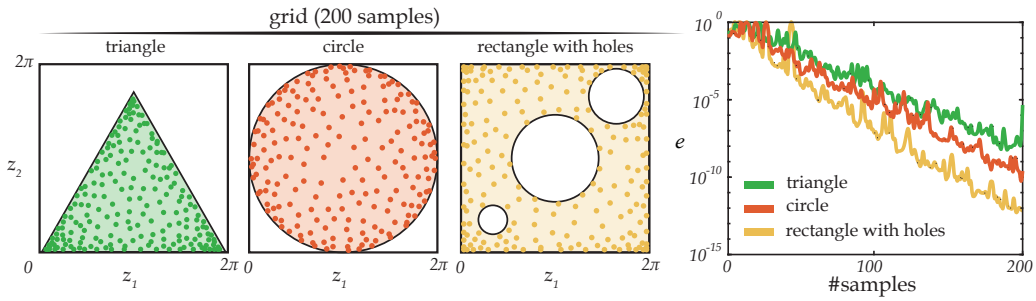


Figure 2.8: *Convergence of surrogate for three different non-hypercube domains.*

Method scales well to higher dimensions

In order to study the effect of an increase in dimensionality, we pose the advection parameter z_1 as a random field $z_1(\xi,t)$. We demonstrate our approach by choosing the most basic random field, i.e., we define $z_1(\xi,t)$ as a random polynomial function:

$$z_1(\xi,t) = \sum_{i=1}^{N_z-1} \xi_i t^i, \ \xi_i \in [0,1] \ . \tag{2.67}$$

The influence of ξ_i decreases when i increases and an efficient sampling approach would place less samples in the ξ_i direction when i increases. A surrogate for the QoI $v_{N_x}(1)$ can be constructed in the high-dimensional random space spanned by $(\xi_i)_{i=1}^{N_z-1}$ and z_2. To show scalability of our approach, we plot the required number of samples such that

$$\|\mathbf{q}-\mathbf{q}_{\text{ref}}\|_2 < 10^{-3} \ , \tag{2.68}$$

where both $\mathbf{q}$ and $\mathbf{q}_{\text{ref}}$ are the constructed surrogate values and exact solution values, respectively, sampled at 10000 uniformly distributed samples in the random space $(\xi_1,...,\xi_{N_z},z_2) \in [0,1]^{N_z-1} \times [0,2\pi]$. The results are shown in figure 2.9.

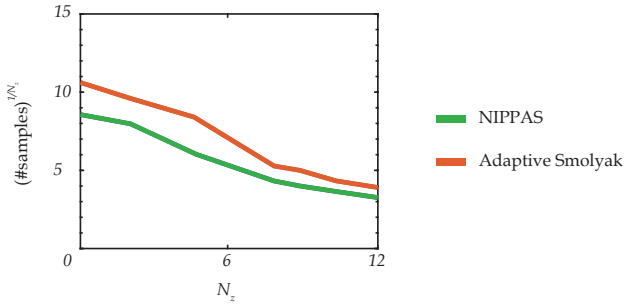


Figure 2.9: *Comparison between the NIPPAS approach and adaptive Smolyak approach in terms of the required number of samples to achieve* $\|\mathbf{q}-\mathbf{q}_{ref}\|_2 < 10^{-3}$. *The required number of samples is post-processed to show the required number of samples per dimension.*

Notice that the NIPPAS approach scales well to high-dimensional random spaces, i.e., increasing N_z, and outperforms the adaptive Smolyak approach up to $N_z = 12$. However, the required number of samples per dimension for the Smolyak approach decays slightly faster than the NIPPAS method, which is likely caused by sub-optimal sample locations resulting from the particle swarm optimisation in high-dimensional spaces. In order to circumvent this issue, one might resort to other

optimisation routines, which are more suitable for finding suitable optima for sample placement in high dimensions. As we need to compute a solution of a PDE at each sample location, it became infeasible to run simulations with $N_z > 12$.

6.3 Shallow water equations with dependent random inputs

As last test-case, we study the performance of the NIPPAS method for a hyperbolic non-linear PDE with dependent random inputs in a non-hypercube random space. In this test-case the underlying model is non-linear and comprises three uncertain parameters which lie on a 2D triangular manifold in a 3D space and therefore this test-case combines the difficult aspects from all previous test-cases. The underlying model is a system of conservation laws, namely the 1D shallow water equations (SWEs):

$$\frac{\partial}{\partial t}\begin{pmatrix} h \\ hv \end{pmatrix} + \frac{\partial}{\partial x}\begin{pmatrix} hv \\ hv^2 + gh^2/2 \end{pmatrix} = \mathbf{0}\,, \tag{2.69}$$

where h is the free surface height (thickness of the fluid layer), v the velocity, g the acceleration of gravity, and where we have assumed that the bottom is flat. Reflective boundary conditions are imposed at $x = \pm 1$ and the initial condition for the system of PDEs is given by a Riemann problem:

$$\begin{pmatrix} h \\ v \end{pmatrix}(x, t=0) = \begin{cases} \begin{pmatrix} h_l \\ v_l \end{pmatrix}, & x \leq 0\,, \\ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, & x > 0\,, \end{cases} \tag{2.70}$$

leading to a dambreak problem. The three uncertain inputs are $z_1 = g$, $z_2 = h_l$ and $z_3 = v_l$, and are assumed to jointly follow a Dirichlet distribution, which is the multivariate generalisation of the 1D β-distribution [66], and which is often used as a prior to a discrete categorical distribution in Bayesian statistics. The PDF of the $3D$ Dirichlet distribution with shape parameters $(\alpha_1, \alpha_2, \alpha_3)$ is given by:

$$\rho(z_1, z_2, z_3; \alpha_1, \alpha_2, \alpha_3) = \frac{\Gamma(\sum_{i=1}^3 \alpha_i)}{\prod_{i=1}^3 \Gamma(\alpha_i)} \prod_{i=1}^3 z_i^{\alpha_i - 1}\,, \tag{2.71}$$

which is defined on the unit simplex

$$\sum_{i=1}^3 z_i = 1, \text{ and } z_i \geq 0,\ \forall i\,. \tag{2.72}$$

For this specific test-case we scale/translate the unit simplex to a triangle with corner points

$$(g, h_l, v_l) = \{(12, 0.5, -0.5), (8, 1.5, -0.5), (8, 0.5, 0.5)\}\,. \tag{2.73}$$

For testing the efficiency of refinement measure (2.25), we consider the shape parameter set $(\alpha_1, \alpha_2, \alpha_3) = (5, 2, 2)$. This specific shape parameter set corresponds to an asymmetric PDF and is shown in figure 2.10.

The Dirichlet distribution is defined on a 2D triangular manifold in 3D space and is used for testing the NIPPAS method efficiency for non-hypercube random spaces. The interpolation basis is the set of Chebyshev polynomials defined on the smallest hypercube that comprises the 2D manifold. As mentioned before, improvements on the interpolation basis are possible, but are outside the scope of this chapter. Furthermore, the QoI $q(z_1, z_2, z_3)$ is the free surface height h at the left boundary $x = -1$ at $t = 1$.

The NIPPAS method samples the solution of the dambreak problem for multiple inputs in order to construct a surrogate. A commonly used method for solving the SWEs is a Riemann solver

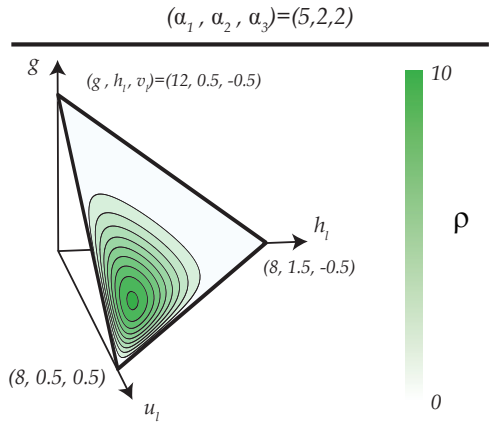


Figure 2.10: *Dirichlet PDF on the scaled unit simplex.*

based finite-volume discretisation [33], which can determine accurate solutions efficiently. In this chapter instead we demonstrate the effectiveness of the NIPPAS method in combination with a more recently developed numerical method. Instead of using a Riemann solver, a neural network is used to solve the SWEs [52]. An advantage of using neural networks for solving PDEs is that *the solution is given in terms of a functional form, from which derivatives can be directly computed analytically*. This functional form allows us to calculate the residual, without requiring alterations to the code output. A multi-output neural network with 7 hidden layers and 40 neurons in each hidden layer is used to solve the SWEs, which is trained on a total of 10^5 collocation nodes in space and time, which are the places where the neural network tries to enforce the PDE. This particular combination of number of hidden layers and neurons was found using straightforward hyperparameter tuning and shows the best results. The trained neural network produces a solution of the PDE. After the residual is computed and a new sample location in random space is determined, the neural network is retrained to produce a solution of the PDE for this new set of parameter values. Previously trained neural networks closest to the new sample location are used as initial starting point for training the new neural network in order to significantly speed-up the training process.

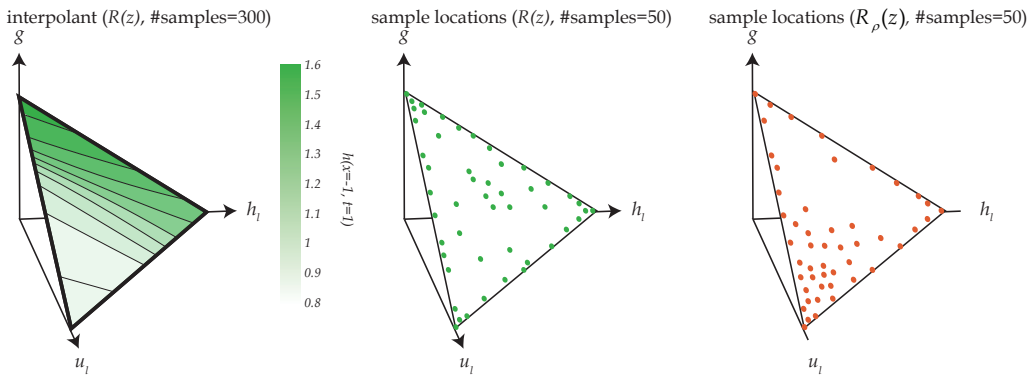


Figure 2.11: *(left) Surrogate model based on 300 samples for refinement measure* (2.21). *(right) Sample locations for refinement measures* (2.21) *and* (2.25)*, respectively.*

The surrogate and sample locations for both refinement measures (2.21) and (2.25) are shown in

figure 2.11. The sample locations show clustering at the boundaries, to produce a stable interpolant. As mentioned before, if the surrogate tends to become unstable and grows at the boundaries of the random space, the residual becomes large at the boundaries as well and causes refinement of the surrogate near the boundaries. At early stages of the refinement process the surrogate can still show irregular oscillations, which is due to insufficient refinement at the boundaries, but these disappear upon further refinement. When taking the PDF into account, clustering also occurs in the region of high probability, as expected. This clustering deteriorates the accuracy of the surrogate in regions of low probabilities, but leads to improved estimation of statistical quantities. Convergence comparison for refinement measures (2.21) and (2.25) are shown in figure 2.12.

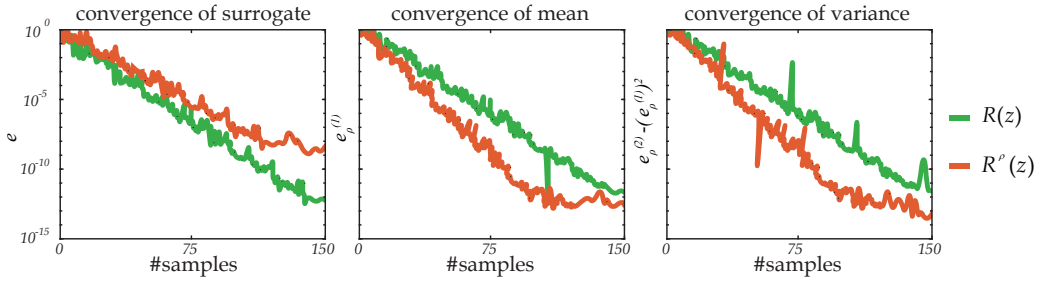


Figure 2.12: *Results for the dambreak problem with random inputs. The convergence plot shows a comparison of the two different refinement measures* (2.21) *and* (2.25)*. The error in the surrogate is computed with* (2.48) *with 5000 Monte-Carlo samples. The errors in the mean and variance are calculated with* (2.47)*.*

The results show indeed faster convergence in statistical quantities when accounting for the PDF in the refinement measure, which was also shown in figure 2.7.

7 Conclusion

In this chapter we have presented a novel approach for parametric surrogate construction when the underlying PDE is known. Our technique, the Non-Intrusive PDE/PDF-informed Adaptive Sampling (NIPPAS), is suited for surrogate construction on non-hypercube parametric spaces. Non-hypercube parametric spaces occur when the underlying PDF is dependent, e.g., Dirichlet-distributed, and significantly complicates surrogate construction when using common stochastic collocation methods, e.g., sparse grid interpolation. The key ingredient of the proposed empirical interpolation procedure is refinement which is based on the PDE-residual and/or the PDF. Sampling based on the PDE-residual leads to stable interpolation, even on non-hypercube domains, due to sample clustering at the boundaries of the domain. At the same time, the incorporation of the PDF in the refinement procedure samples the parametric space in regions of high probability, which ensures fast convergence of statistical quantities. This combination makes the NIPPAS method an efficient and flexible method that is applicable to a wide range of UQ problems. As an extension to the previously proposed refinement measures, a convex combination of both the residual and PDF-based residual can be defined, which attempt to balance the trade-offs between the two types of residuals, simultaneously. However, such a refinement measure does not show any improvement over the two previously proposed refinement measures.

The NIPPAS method has been applied to several numerical examples: 1D and 2D surrogate construction on a hypercube with linear underlying PDE, 2D surrogate construction on complex domains, and 3D surrogate construction with non-linear underlying PDE on a complex domain. In all cases, exponential convergence is obtained, leading to an accurate surrogate model. This

surrogate model can be directly used as a tool for uncertainty quantification (for example with Monte-Carlo type methods), but it is also a great tool for the parametric solution of gray-box models.

Currently, the interpolation basis for non-hypercube domains is a Chebyshev basis defined on the smallest hypercube that comprises the parametric space. Several improvements could be made, for instance by constructing a suitable basis based on the sample locations [14]. Furthermore, the global minimisation problem to be solved at each iteration does not scale well to high-dimensional random space, and alternatives for the particle swarm optimisation may be used [67].

What did we achieve?

This chapter discussed how to effectively place your when the QoI lives in a high-dimensional and possibly non-hypercube random space. The NIPPAS method provides an efficient way to sample your QoI in such spaces by basing the sample locations on the PDE residual and/or the PDF. Whether to choose sample location based on the PDE, PDE+PDF, depends on if you are interested in either convergence of the surrogate, or convergence of the QoI statistics, i.e., mean and variance.

References

[1] C. Canuto, M. Y. Hussaini, A. M. Quarteroni, T. A. Zang, Jr, Spectral Methods in Fluid Dynamics, Springer Science & Business Media, 2012.

[2] J. A. S. Witteveen, A. Loeven, H. Bijl, An adaptive stochastic finite elements approach based on Newton-Cotes quadrature in simplex elements, Computers & Fluids 38 (6) (2009) 1270–1288.

[3] J. D. Jakeman, S. G. Roberts, Local and dimension adaptive sparse grid interpolation and quadrature, (Sep 2011).arXiv:1110.0010.

[4] X. Wan, G. Karniadakis, Multi-element generalized polynomial chaos for arbitrary probability measures, SIAM Journal on Scientific Computing 28 (3) (2006) 901–928.

[5] J. Foo, X. Wan, G. E. Karniadakis, The multi-element probabilistic collocation method (ME-PCM): Error analysis and applications, Journal of Computational Physics 227 (22) (2008) 9572–9595.

[6] J. D. Jakeman, A. Narayan, D. Xiu, Minimal multi-element stochastic collocation for uncertainty quantification of discontinuous functions, Journal of Computational Physics 242 (2013) 790–808.

[7] A. Gorodetsky, Y. Marzouk, Efficient localization of discontinuities in complex computational simulations, SIAM Journal on Scientific Computing 36 (6) (2014) A2584–A2610.

[8] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical Society 7 (1) (1956) 48–50.

[9] R. L. Graham, P. Hell, On the history of the minimum spanning tree problem, Annals of the History of Computing 7 (1) (1985) 43–57.

[10] G. S. El-tawel, A. K. Helmy, An edge detection scheme based on least squares support vector machine in a contourlet HMT domain, Applied Soft Computing 26 (2015) 418–427.

[11] M. K. S. Varma, N. K. K. Rao, K. K. Raju, G. P. S. Varma, Pixel-based classification using support vector machine classifier, in: 2016 IEEE 6th International Conference on Advanced Computing (IACC), 2016, pp. 51–55.

[12] C. J. C. Burges, A tutorial on support vector machines for pattern recognition, Data Mining and Knowledge Discovery 2 (2) (1998) 121–167.

[13] B. Scholkopf, A. J. Smola, Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, MIT Press, 2001.

[14] A. Narayan, D. Xiu, Stochastic collocation methods on unstructured grids in high dimensions via interpolation, SIAM Journal on Scientific Computing 34 (3) (2012) A1729–A1752.

[15] C. B. Barber, D. P. Dobkin, H. Huhdanpaa, The quickhull algorithm for convex hulls, ACM Transactions on Mathematical Software 22 (4) (1996) 469–483.

[16] J. E. Goodman, J. O'Rourke (Eds.), Handbook of Discrete and Computational Geometry, CRC Press, Inc., 1997.

[17] J. Witteveen, G. Iaccarino, Simplex elements stochastic collocation for uncertainty propagation in robust design optimization, in: 48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, American Institute of Aeronautics and Astronautics, 2010.

[18] J. Witteveen, G. Iaccarino, Simplex stochastic collocation with random sampling and extrapolation for nonhypercube probability spaces, SIAM Journal on Scientific Computing 34 (2) (2012) A814–A838.

[19] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, `http://www.deeplearningbook.org`.

[20] J. D. Jakeman, R. Archibald, D. Xiu, Characterization of discontinuities in high-dimensional stochastic problems on adaptive sparse grids, Journal of Computational Physics 230 (10) (2011) 3977–3997.

[21] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Transactions on Intelligent Systems and Technology 2 (3) (2011) 27:1–27:27.

[22] Y. Ohtake, A. Belyaev, H.-P. Seidel, 3D scattered data interpolation and approximation with multilevel compactly supported RBFs, Graphical Models 67 (3) (2005) 150 – 165.

[23] A. Gorodetsky, Y. Marzouk, Mercer kernels and integrated variance experimental design: connections between Gaussian process regression and polynomial approximation, SIAM/ASA Journal on Uncertainty Quantification 4 (1) (2016) 796–828.

[24] V. Vapnik, O. Chapelle, Bounds on error expectation for support vector machines, Neural Computation 12 (9) (2000) 2013–2036.

[25] D. W. Scott, G. R. Terrell, Biased and unbiased cross-validation in density estimation, Journal of the American Statistical Association 82 (400) (1987) 1131–1146.

[26] G. Blatman, Adaptive sparse polynomial chaos expansions for uncertainty propagation and sensitivity analysis, Ph.D. thesis, Université Blaise Pascal, Clermont-Ferrand, France (2009).

[27] S. Fortune, A Sweepline Algorithm for Voronoi Diagrams, SCG '86, ACM, 1986, pp. 313–322.

[28] R. C. Prim, Shortest connection networks and some generalizations, The Bell System Technical Journal 36 (6) (1957) 1389–1401.

[29] A. Genz, Testing multidimensional integration routines, in: Proc. of International Conference on Tools, Methods and Languages for Scientific and Engineering Computation, Elsevier North-Holland, Inc., 1984, pp. 81–94.

[30] A. Narayan, J. D. Jakeman, T. Zhou, A Christoffel function weighted least squares algorithm for collocation approximations, Mathematics of Computation 86 (306) (2017) 1913–1947.

[31] J. Helton, F. Davis, Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems, Reliability Engineering & System Safety 81 (1) (2003) 23 – 69.

[32] C. B. Vreugdenhil, Numerical Methods for Shallow-Water Flow, Springer Verlag, 2013.

[33] R. J. LeVeque, Finite Volume Methods for Hyperbolic Problems, Cambridge University Press, 2002.

[34] E. F. Toro, Riemann Solvers and Numerical Methods for Fluid Dynamics: A Practical Introduction, Springer Science & Business Media, 2013.

[35] E. J. Hopfinger, V. Baumbach, Liquid sloshing in cylindrical fuel tanks, in: Progress in Propulsion Physics, Vol. 1, EDP Sciences, pp. 279–292.

[36] R. Marsooli, W. Wu, 3-D finite-volume model of dam-break flow over uneven beds based on VOF method, Advances in Water Resources 70 (Supplement C) (2014) 104–117.

[37] L. A. Larocque, J. Imran, M. H. Chaudhry, 3-D numerical simulation of partial breach dam-break flow using the LES and k- turbulence models, Journal of Hydraulic Research 51 (2) (2013) 145–157.

[38] A. J. C. Crespo, J. M. Dominguez, B. D. Rogers, M. Gomez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. Garcia-Feal, DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH), Computer Physics Communications 187 (Supplement C) (2015) 204–216.

[39] L. Lobovský, E. Botia-Vera, F. Castellana, J. Mas-Soler, A. Souto-Iglesias, Experimental investigation of dynamic pressure loads during dam break, Journal of Fluids and Structures 48 (Supplement C) (2014) 407–434.

[40] A. Narayan, J. Jakeman, Adaptive Leja sparse grid constructions for stochastic collocation and high-dimensional approximation, SIAM Journal on Scientific Computing 36 (6) (2014) A2952–A2983.

[41] M. Barrault, Y. Maday, N. C. Nguyen, A. T. Patera, An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations, Comptes Rendus Mathématique 339 (9) (2004) 667 – 672.

[42] J. S. Hesthaven, G. Rozza, B. Stamm, The empirical interpolation method, in: Certified Reduced Basis Methods for Parametrized Partial Differential Equations, Springer Briefs in Mathematics, Springer, 2016, pp. 67–85.

[43] J. S. Hesthaven, B. Stamm, S. Zhang, Efficient greedy algorithms for high-dimensional parameter spaces with applications to empirical interpolation and reduced basis methods, ESAIM: Mathematical Modelling and Numerical Analysis 48 (1) (2014) 259–283.

[44] M. Ohlberger, S. Rave, Reduced basis methods: Success, limitations and future challenges, arXiv:1511.02021.

[45] N. C. Nguyen, A. T. Patera, J. Peraire, A 'best points' interpolation method for efficient approximation of parametrized functions, International Journal for Numerical Methods in Engineering 73 (4) (2008) 521–543.

[46] Y. Maday, N. Nguyen, A. Patera, G. S. H. Pau, A general multipurpose interpolation procedure: The magic points, Communications on Pure and Applied Analysis 8.

[47] P. Chen, A. Quarteroni, G. Rozza, Comparison between reduced basis and stochastic collocation methods for elliptic problems, Journal of Scientific Computing 59 (1) (2014) 187–216.

[48] P. Chen, A. Quarteroni, G. Rozza, A weighted empirical interpolation method: A priori convergence analysis and applications, ESAIM Mathematical Modelling and Numerical Analysis 48 (2014) 943–953.

[49] P. Chen, A. Quarteroni, Weighted reduced basis method for stochastic optimal control problems with elliptic PDE constraint, SIAM/ASA Journal on Uncertainty Quantification 2 (2014) 364–396.

[50] P. Chen, A. Quarteroni, G. Rozza, A weighted reduced basis method for elliptic partial differential equations with random input data, SIAM Journal on Numerical Analysis 51 (6) (2013) 3163–3185.

[51] S. Boyaval, C. Bris, T. Lelièvre, Y. Maday, N. Nguyen, A. Patera, Reduced basis techniques for stochastic problems, Archives of Computational Methods in Engineering 17 (2010) 435–454.

[52] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, (Nov 2017), arXiv:1711.10561.

[53] D. Xiu, Numerical Methods for Stochastic Computations: A Spectral Method Approach, Princeton University Press, 2010.

[54] L. N. Trefethen, Approximation Theory and Approximation Practice, SIAM, 2013.

[55] D. Loukrezis, U. Römer, H. De Gersem, Numerical Comparison of Leja and Clenshaw-Curtis Dimension-Adaptive Collocation for Stochastic Parametric Electromagnetic Field Problems, arXiv:1712.07223.

[56] N. Higham, Accuracy and Stability of Numerical Algorithms, Other Titles in Applied Mathematics, SIAM, 2002.

[57] C. d. Boor, A. Ron, On multivariate polynomial interpolation, Constructive Approximation 6 (3) (1990) 287–302.

[58] T. Sauer, Polynomial interpolation of minimal degree, Numerische Mathematik 78 (1) (1997) 59–85.

[59] A. Neumaier, Solving ill-conditioned and singular linear systems: A tutorial on regularization, SIAM Review 40 (3) (1998) 636–666.

[60] J. Varah, On the numerical solution of ill-conditioned linear systems with applications to ill-posed problems, SIAM Journal on Numerical Analysis 10 (2) (1973) 257–267.

[61] N. Higham, The accuracy of solutions to triangular systems, SIAM Journal on Numerical Analysis 26 (5) (1989) 1252–1265.

[62] A. Klinger, Approximate pseudoinverse solutions to ill-conditioned linear systems, Journal of Optimization Theory and Applications 2 (2) (1968) 117–124.

[63] S. Mohideen, V. Cherkassky, On recursive calculation of the generalized inverse of a matrix, ACM Transactions on Mathematical Software 17 (1) (1991) 130–147.

[64] N. Shinozaki, M. Sibuya, K. Tanabe, Numerical algorithms for the Moore-Penrose inverse of a matrix: Direct methods, Annals of the Institute of Statistical Mathematics 24 (1) (1972) 193–203.

[65] A. Klimke, B. Wohlmuth, Algorithm 847: Spinterp: Piecewise multilinear hierarchical sparse grid interpolation in MATLAB, ACM Transactions on Mathematical Software 31 (2005) 561–579.

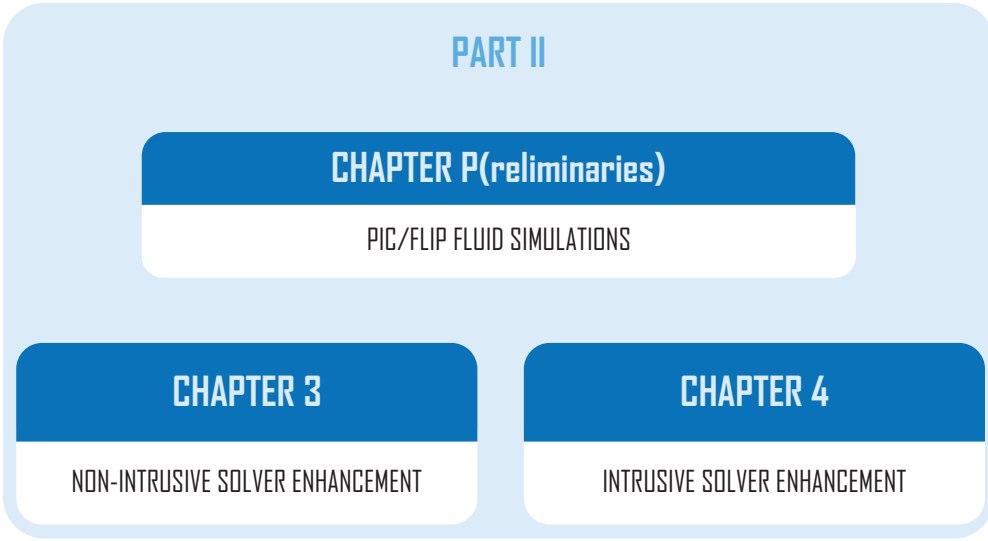
[66] M. Hazewinkel, Encyclopaedia of Mathematics, Springer Science & Business Media, 1990.

[67] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

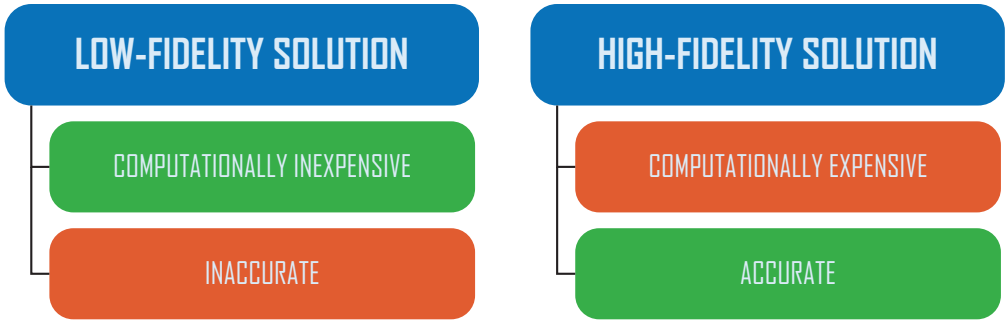
Part II

solver enhancement

In Part I of this thesis, it is discussed how to efficiently sample the QoI in order to significantly reduce the computational cost when performing UQ. An alternative for reducing the overall computational cost when performing UQ is to speed-up the underlying numerical solver that is used to sample the QoI, either by reducing the computational time of computationally expensive and accurate solutions, or to increase the accuracy of computationally cheap but inaccurate solutions. We will look at how to effectively increase the accuracy of computationally cheap but inaccurate numerical solutions in both a non-intrusive and intrusive manner. Our non-intrusive approach functions as a shell around a low-fidelity solver. The non-intrusive approach takes solutions coming from the low-fidelity solver and enhances these in a post-processing stage. The intrusive approach alters the underlying low-fidelity solver by adding a neural network that maps quantities that are defined on a coarse grid to an accurate counterpart on a high-resolution grid. An overview of this part of the thesis is given below.



In order to make a clear distinction between the level of accuracy and the computational time it takes to compute a solution, we define low- and high-fidelity solutions in more detail in the figure below. Using a low-fidelity solver alleviates the computational burden by lowering the computational time



required for computing a solution, but may drop the accuracy of this same solution. Using a high-fidelity solver may increase accuracy, but at the cost of a significant increase in computational time.

As a remedy, a multi-fidelity approach utilises both low- and high-fidelity solutions to construct an accurate, but fast solution.
In this part we focus in particular on enhancing low-fidelity fluid simulations, i.e., Particle In Cell/FLuid Implicit Particle (PIC/FLIP) simulations, but as is shown in chapter 3, the methodologies can be used for other applications as well. The PIC/FLIP method is commonly used to simulate free-surface flows, due to its easy parallel implementation and flexibility when it comes to placing obstacles in the computational domain. The inner and outer workings of the PIC/FLIP solver are discussed in the next chapter. Then, in chapter 3 and 4 these PIC/FLIP simulations are enhanced using a new non-intrusive approach and an intrusive approach.

P

PIC/FLIP simulations for fluid flow

Yous van Halder

We discuss the fluid flow solver in more detail. The governing equations for incompressible fluid flow are the Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0 \,, \tag{P.1a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{\nabla p}{\rho} + \mathbf{F} + \nu \nabla^2 \mathbf{u} \,, \tag{P.1b}$$

where $\mathbf{u}$ is the velocity field, p the pressure, ρ the density, $\mathbf{F}$ the body-force vector, and ν the kinematic viscosity. The first equation (P.1a) is known as the incompressibility condition, while the second set of equations (P.1b) is known as the momentum equation. When solving these equations, we can resort to grid-based methods or particle-based methods. In this chapter we use a PIC/FLIP solver for predicting single-phase free-surface flows, as it allows for real-time computation when implemented efficiently on a GPU. This numerical method is a combination of a grid-based method and a particle-based method, and is often used to simulate fluid motions in movies or computer games due to its easy parallel implementation. The particles represent the fluid, and the positions of these particles are evolved over time by using velocity values that are computed on a staggered grid.

Based on the work of Bridson [1, 2]

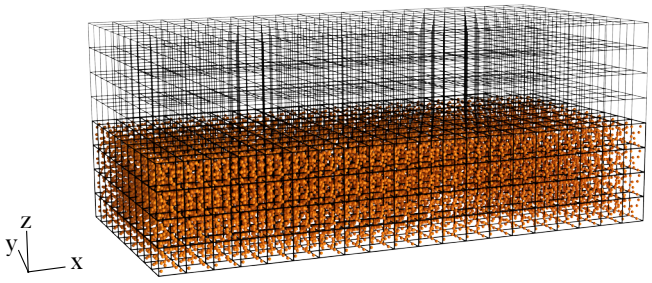


Figure P.2: *Grid and particle initialisation with* $(N_x, N_y, N_z) = (20, 8, 8)$. *The top half of the grid comprises dry cells while the cells in the bottom half are filled with particles.*

P. 2.1 Particle to grid transfer

A weighted average of particle velocities is used to compute the velocities at the grid-cell faces. In order to compute the velocity at a given face centre, particles that lie within a sphere with a $2\Delta s$ radius, centred around this face centre, are used for computing the new velocity value. Consequently, the new face centre velocities are given by:

$$u_{i+\frac{1}{2},j,k} = \frac{\sum_{i=1}^{N_p} p_u^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i+\frac{1}{2},j,k})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i+\frac{1}{2},j,k})} , \tag{P.2a}$$

$$v_{i,j+\frac{1}{2},k} = \frac{\sum_{i=1}^{N_p} p_v^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j+\frac{1}{2},k})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j+\frac{1}{2},k})} , \tag{P.2b}$$

$$w_{i,j,k+\frac{1}{2}} = \frac{\sum_{i=1}^{N_p} p_w^{(i)} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j,k+\frac{1}{2}})}{\sum_{i=1}^{N_p} h(\mathbf{p}^{(i)} - \mathbf{x}_{i,j,k+\frac{1}{2}})} , \tag{P.2c}$$

where the vectors $\mathbf{x}$ correspond to the locations of the face centres, and where h is the so-called kernel. The choice of kernel h depends on the application. In general the cubic kernel is considered to be robust and it results in a smoothed out fluid surface without irregular jumps when post-processed [1]:

$$k(\mathbf{r}) \begin{cases} (4\Delta s^2 - \|\mathbf{r}\|_2^2)^3 & , \|\mathbf{r}\|_2 \leq 2\Delta s , \\ 0 & , \text{otherwise} . \end{cases} \tag{P.3}$$

The particles that are used for the grid transfer for a single face velocity are schematically shown in figure P.3.

Store velocities

After the particle velocities have been transferred to the grid, the resulting face velocities are copied for later use, i.e., we define:

$$u^*_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k}, \quad v^*_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k}, \quad w^*_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} . \tag{P.4}$$

The copied velocities remain unaltered for the rest of the time step.

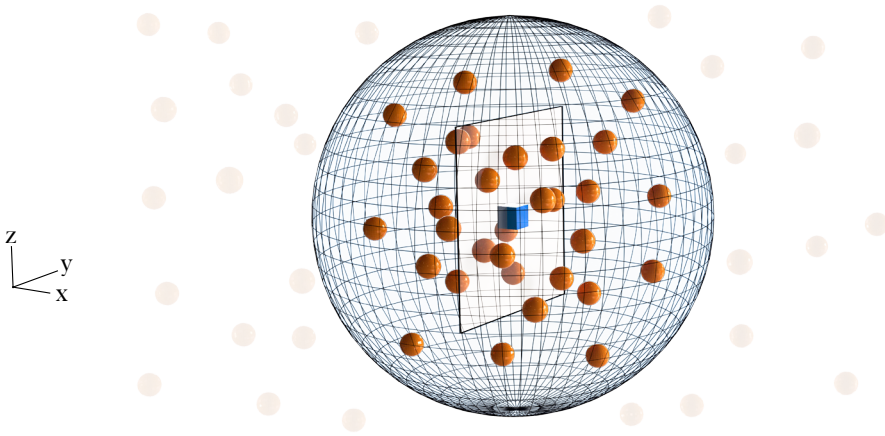


Figure P.3: *Particles that are used for computing one face velocity.*

P. 2.2 Add body forces

The external forces, such as gravity, are added to the velocity field by a simple forward Euler time-integration:

$$u_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k} + \Delta t F_x(\mathbf{x}_{i+\frac{1}{2},j,k}) \,, \tag{P.5a}$$

$$v_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k} + \Delta t F_y(\mathbf{x}_{i,j+\frac{1}{2},k}) \,, \tag{P.5b}$$

$$w_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} + \Delta t F_z(\mathbf{x}_{i,j,k+\frac{1}{2}}) \,, \tag{P.5c}$$

where F_x, F_y and F_z are the components of the body force acting on the fluid in each coordinate direction, calculated at the corresponding face centres. A more accurate time-integration scheme may be used, but forward Euler is very easy to implement and is therefore considered as the standard in PIC/FLIP [2, 1]. When Δs is set, we need to choose Δt accordingly such that the forward Euler time integration remains stable, which is often not an issue when using PIC/FLIP due to the often coarse computational grid.

P. 2.3 Enforce boundary conditions

We assume that the domain boundaries and obstacles are solid, and therefore solid-wall boundary conditions are imposed at these locations, which state that fluid is not allowed to flow out of the domain. As a result, the boundary condition is given by:

$$\mathbf{u} \cdot \mathbf{n} = \mathbf{u}_{\text{boundary}} \cdot \mathbf{n} \,, \tag{P.6}$$

where $\mathbf{n}$ is the normal vector pointing outside the domain. Furthermore, this boundary condition can be simplified when using a staggered grid, where the faces intersect with the domain boundaries:

$$u_{-\frac{1}{2},j,k} = u_{\text{boundary},0}, \quad u_{N_x-\frac{1}{2},j,k} = u_{\text{boundary},L_x} \,, \tag{P.7a}$$

$$v_{i,-\frac{1}{2},k} = v_{\text{boundary},0}, \quad v_{i,N_y-\frac{1}{2},k} = v_{\text{boundary},L_y} \,, \tag{P.7b}$$

$$w_{i,j,-\frac{1}{2}} = w_{\text{boundary},0}, \quad w_{i,j,N_z-\frac{1}{2}} = w_{\text{boundary},L_z} \,, \tag{P.7c}$$

where we assumed that the domain does not deform and the boundaries have a uniform velocity, i.e., $u_{\text{boundary},0} = u_{\text{boundary},L_x}$, $v_{\text{boundary},0} = v_{\text{boundary},L_y}$, $w_{\text{boundary},0} = w_{\text{boundary},L_z}$.

P. 2.4 Determine which cells contain particles

The cells that contain at least one particle are referred to as wet cells. We define a new quantity $m_{i,j,k}$:

$$m_{i,j,k} \begin{cases} 1 & \text{, if cell } (i,j,k) \text{ contains at least one particle ,} \\ 0 & \text{, otherwise .} \end{cases} \tag{P.8}$$

P. 2.5 Pressure correction

The PIC/FLIP method simulates incompressible fluid flow, which means that the velocities at the face centres of all i, j, k have to satisfy the following discrete version of incompressibility constraint (P.1a):

$$(\nabla \cdot \mathbf{u})_{i,j,k} \rightarrow \frac{u_{i+\frac{1}{2},j,k} - u_{i-\frac{1}{2},j,k}}{\Delta s} + \frac{v_{i,j+\frac{1}{2},k} - v_{i,j-\frac{1}{2},k}}{\Delta s} + \frac{w_{i,j,k+\frac{1}{2}} - w_{i,j,k-\frac{1}{2}}}{\Delta s} = 0 \,, \tag{P.9}$$

which states that the central second-order accurate divergence, computed at each cell centre, should be zero. In general, the face velocities from (P.5a)-(P.5c) do not satisfy this constraint. Therefore, the velocities are corrected by subtracting a pressure gradient. In order to calculate this pressure gradient, we solve the pressure equation in the cells that are wet ($m_{i,j,k} = 1$):

$$(\nabla^2 p)_{i,j,k} = \frac{1}{\Delta t} (\nabla \cdot \mathbf{u})_{i,j,k} \,, \tag{P.10}$$

where

$$(\nabla^2 p)_{i,j,k} \rightarrow \frac{p_{i+1,j,k} + p_{i-1,j,k}}{\Delta s^2} + \frac{p_{i,j+1,k} + p_{i,j-1,k}}{\Delta s^2} + \frac{p_{i,j,k+1} + p_{i,j,k-1}}{\Delta s^2} - \frac{6 p_{i,j,k}}{\Delta s^2} \,, \tag{P.11}$$

and set $p_{i,j,k} = 0$ in dry cells. Notice that near the domain boundaries or obstacles we require the pressures located outside the domain, which are not defined, and a different formulation is necessary. To enforce that there is no flow through solid boundaries, we employ the commonly used homogeneous Neumann boundary condition for the pressure at these boundaries. Even though the homogeneous Neumann boundary condition may translate to a non-zero tangential velocity component along the boundaries, it is the easiest to implement while still enforcing that no fluid flows outside the domain. Other boundary conditions, e.g., strict no-slip boundary conditions, are not straightforward to implement on a staggered grid and are not considered in this paper. The homogeneous Neumann boundary conditions set the pressure coefficient of the cells outside the domain or inside an obstacle to zero and the central pressure coefficient 6 in (P.11) is decreased by the number of cells outside the domain or inside an obstacle, which translated to a pressure gradient that is zero on the boundary [1]. All this results in a sparse linear system of the form:

$$A \begin{pmatrix} p_{0,0,0} \\ p_{1,0,0} \\ \vdots \\ p_{N_x-1,N_y-1,N_z-1} \end{pmatrix} = \frac{1}{\Delta t} \begin{pmatrix} (\nabla \cdot \mathbf{u})_{0,0,0} \\ (\nabla \cdot \mathbf{u})_{1,0,0} \\ \vdots \\ (\nabla \cdot \mathbf{u})_{N_x-1,N_y-1,N_z-1} \end{pmatrix} , \tag{P.12}$$

where A is the pressure Poisson matrix. This system is solved using the preconditioned conjugate gradient method, as it does not require the explicit storage of the coefficient matrix A, and still exhibits relatively fast convergence for symmetric matrices A when compared to Jacobi iteration.

After the pressures have been computed, we correct the face velocities, that do not coincide with a domain boundary, as follows:

$$u_{i+\frac{1}{2},j,k} = u_{i+\frac{1}{2},j,k} - \frac{\Delta t}{\Delta s}\left(p_{i+1,j,k} - p_{i,j,k}\right), \tag{P.13a}$$

$$v_{i,j+\frac{1}{2},k} = v_{i,j+\frac{1}{2},k} - \frac{\Delta t}{\Delta s}\left(p_{i,j+1,k} - p_{i,j,k}\right), \tag{P.13b}$$

$$w_{i,j,k+\frac{1}{2}} = w_{i,j,k+\frac{1}{2}} - \frac{\Delta t}{\Delta s}\left(p_{i,j,k+1} - p_{i,j,k}\right), \tag{P.13c}$$

which now satisfy the constraint (P.9) [1].

P. 2.6 Grid to particle transfer

Once the grid velocities are known, they have to be transferred back to the particles in order to advect the particles. To calculate the new particle velocities, we use ***trilinear interpolation***. We will discuss how to compute the $u-$component of the particle velocity $p_u^{(i)}$, and the remaining two components can be computed in a similar fashion.

The face centre velocities that are used for the trilinear interpolation procedure for computing the $u-$component of the velocity $p_u^{(i)}$ for a single particle are shown in figure P.4.

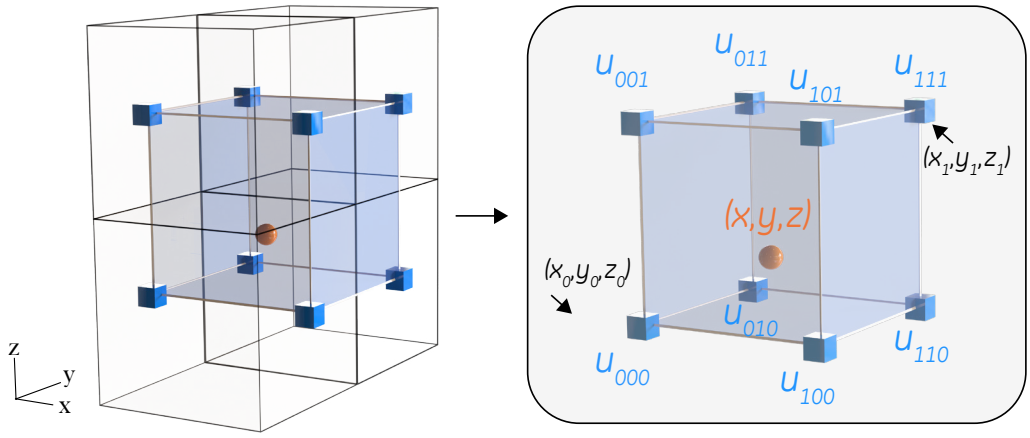


Figure P.4: *The face velocities that are used for the trilinear interpolation procedure to calculate the $u-$component of the particle velocity.*

The $u-$component of the velocity is now calculated as:

$$x^* = \frac{x - x_0}{x_1 - x_0}, \quad y^* = \frac{y - y_0}{y_1 - y_0}, \quad z^* = \frac{z - z_0}{z_1 - z_0}\ , \tag{P.14a}$$

$$\begin{aligned} u_{00} &= u_{000}(1-x^*) + u_{100}x^*, \quad u_{10} = u_{001}(1-x^*) + u_{101}x^*\ , \\ u_{01} &= u_{010}(1-x^*) + u_{110}x^*, \quad u_{11} = u_{011}(1-x^*) + u_{111}x^*\ , \end{aligned} \tag{P.14b}$$

$$\begin{aligned} u_0 &= u_{00}(1-y^*) + u_{01}y^*\ , \\ u_1 &= u_{10}(1-y^*) + u_{11}y^*\ , \end{aligned} \tag{P.14c}$$

$$p_u^{(i)} = u_0(1-z^*) + u_1 z^*\ . \tag{P.14d}$$

This trilinear interpolation is performed two times to perform the PIC/FLIP velocity update; with the new grid velocities $u_{i+\frac{1}{2},j,k}$ from (P.13b), and the old grid velocities $u^*_{i+\frac{1}{2},j,k}$ from (P.4). As a result, two particle velocities are obtained $p^{(i)}_{\text{new},u}$ and $p^{(i)}_{\text{old},u}$. The new particle velocity is calculated as follows:

$$p_u^{(i)} \leftarrow (1-f)(p_u^{(i)} - p^{(i)}_{\text{old},u}) + p^{(i)}_{\text{new},u}\ , \tag{P.15}$$

where $p_u^{(i)}$ is the particle velocity from the previous time step, and $f \in [0,1]$ the so-called PICness parameter. If $f = 1$, then the velocity update is known as the PIC update, while setting $f = 0$ corresponds to the FLIP update. PIC is known for its good stability properties, but suffers from severe numerical diffusion. On the other hand, FLIP does not suffer from numerical diffusion, but can become unstable. Therefore, a weighted average between PIC and FLIP is taken, $f \in (0,1)$, to obtain a stable, but less diffusive solution. Notice that we did not include physical diffusion induced by the viscosity term in the method so far. In PIC/FLIP methods, the physical diffusion term in the Navier-Stokes equations is approximated by tweaking the numerical viscosity by choosing a specific value for the PICness parameter f, namely, f can be related to the kinematic viscosity as:

$$f = \max\left\{1, \frac{6\Delta t\, \nu}{\Delta s^2}\right\}\ , \tag{P.16}$$

and the value for ν should be set to the desired kinematic viscosity accordingly. It is common practice to use $f = 0.99$, which yields good results for water flow [1] and is used in the remainder of this thesis unless stated otherwise.

P. 2.7 Advect particles

A suitable time-integrator is used to advect the particles. A good trade-off between computational expense and accuracy is the second-order accurate Runge-Kutta method (RK2) which mimics the midpoint rule. The procedure for advecting the particles is given by:

1. Advect the particles for the duration of a half time step: $\mathbf{p}^{(i)}_{\text{halfway}} = \mathbf{p}^{(i)} + \frac{\Delta t}{2}\mathbf{p}^{(i)}_{\mathbf{u}}$,
2. Compute new particle velocities using trilinear interpolation, using the grid velocities $u_{i+\frac{1}{2},j,k}$ to obtain $\mathbf{p}^{(i)}_{\text{halfway},\mathbf{u}}$,
3. Compute new particle positions as $\mathbf{p}^{(i)} = \mathbf{p}^{(i)} + \Delta t\, \mathbf{p}^{(i)}_{\text{halfway},\mathbf{u}}$.

After computing new particle positions $\mathbf{p}^{(i)}_{\text{halfway}}$ and $\mathbf{p}^{(i)}$, it may appear that some particles have crossed the boundary of the domain. Particles that hit the domain boundary are reflected back into the domain to prevent particles leaving the domain.

P. 3 PIC/FLIP for fluid sloshing

We will consider fluid sloshing simulations in 3D. These sloshing simulations can be performed in a number of ways and we choose the one that is easiest to implement. In order to induce fluid sloshing, the rectangular tank can be rotated along the x-axis and y-axis independently. This rotation of the tank will alter the orientation of the gravity vector. To clarify, the rotation of the tank enters the fluid solver in the body force as $\mathbf{F} = \mathbf{g} = 9.81(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$, where ψ and ϕ are the rotation angles along the x and y axis, respectively. Rotating the gravity vector mimics rotation of the tank without the need to complicate the boundary conditions of the tank boundary to incorporate rotational accelerations.

P. 4 Accuracy of PIC/FLIP

The accuracy of a PIC/FLIP simulation is determined by the number of particles N_p and the grid resolution Δs. As the particles do not react with each other, the steps that involve particles are easily implemented on a GPU. The number of particles can be chosen to be large (millions) while still being able to run in real-time. Most computational work occurs in the computations on the grid, especially in the solution of equation (P.12) to compute the pressures at the cell centres. It is not straightforward to efficiently implement these computations on a GPU. Solving the pressure equation is done in $O(N_c \log(N_c))$ time, where N_c is the total number of grid cells. Decreasing the grid resolution by a factor 2 in each coordinate direction, i.e., $\Delta s \to 2\Delta s$, results in a pressure solve that is approximately 10 times faster. As a result, when choosing a coarse grid, we can still easily simulate up to millions of particles in real-time, but computing solutions on a coarse grid results in a significant drop in accuracy of the particle positions when compared to simulating flows on a fine grid with the same number of particles. This drop in accuracy is mainly caused by the inaccurate incompressible velocity field that is computed on a the coarse grid, and the trilinear interpolation that is used to transfer these grid velocities to the particles. For instance, the error in the trilinear interpolation is composed of three linear interpolation errors R_{int}, which are bounded by [3]

$$R_{\text{int}} \leq C\Delta s^2 \,, \tag{P.17}$$

where C is a constant depending on the smoothness of the velocity field that is transferred to the particles. Equation (P.17) expresses that the error increases quadratically when coarsening the grid. Insufficiently accurate solutions may quickly arise when coarsening the computational grid. This quadratic scaling of errors holds for other grid computations as well (divergence computation (P.9), pressure computation (P.12)).


$\int e^u du = e^u + C$
$(\ln u)' = \frac{1}{u} \cdot u'$
$y = x^3$
$y = \arcsin$
$X^2 = 2pz$
$(c)' = 0$
$(\sqrt{u})' = \frac{1}{2\sqrt{u}} \cdot u'$
$e^{iy} = \cos$
$\int u^\alpha du =$
$|\vec{a}| = \sqrt{a_x^2 + a_y^2}$
$\cos u \cdot u'$
$\frac{1}{\cos^2 u} \cdot u'$
$e^u \cdot u'$
$\frac{\pi}{2}$
$-\frac{\pi}{2}$
-1
0
1
x
y
z

Non-intrusive solver enhancement

3 Non-intrusive solver enhancement

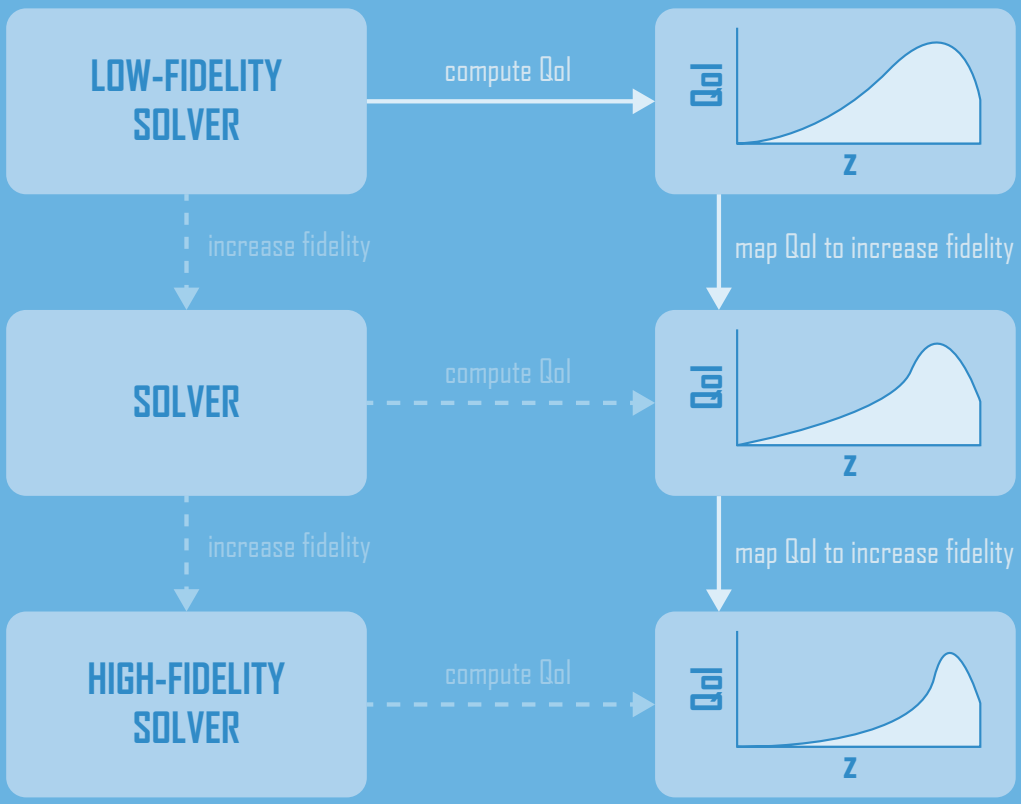
Multi-level neural networks for PDEs with uncertain parameters

A novel multi-level method for partial differential equations with uncertain parameters is proposed. The principle behind the method is that the error between grid levels in multi-level methods has a spatial structure that is by good approximation independent of the actual grid level. Our method learns this structure by employing a sequence of convolutional neural networks, that are well-suited to automatically detect local error features as latent quantities of the solution. Furthermore, by using the concept of transfer learning, the information of coarse grid levels is reused on fine grid levels in order to minimize the required number of samples on fine levels. The method outperforms state-of-the-art multi-level methods, especially in the case of complex PDEs (such as single-phase and free-surface flow problems), or when high accuracy is required.

Background

As stated in the introduction, the goal of part II is to increase the accuracy of fast, but inaccurate solutions. The resulting low-fidelity solver with enhanced accuracy can then be used to efficiently sample a QoI, even in high-dimensional random spaces. This chapter discusses a non-intrusive way to enhance such a low-fidelity solver. The ***curse of dimensionality*** limits the applicability of UQ algorithms, especially when the black-box is computationally expensive to sample from. Using a low-fidelity black-box alleviates the computational burden by lowering the computational time per sample, but may drop the accuracy of the deterministic samples. Using a high-fidelity black-box may increase accuracy, but at the cost of a significant increase in computational time. As a remedy, multi-fidelity approaches were introduced, which use high-fidelity samples to construct the surrogate model and low-fidelity samples to explore the parameter space, and to see where to place a new high-fidelity sample [4, 5, 6, 7, 8, 9, 10, 11]. These approaches significantly reduce the number of high-fidelity samples required to construct an accurate surrogate model. Multi-Level Stochastic Collocation (MLSC) [12, 13, 14, 15, 16, 17] extends this approach by using a hierarchy of grid resolutions, often referred to as levels, and places samples on each grid level to approximate the difference between two consecutive grid resolutions. The underlying concept of variance decay (of the difference in the solution between subsequent levels) ensures that the number of samples required to approximate the difference between two consecutive levels, reduces with increasing level (variance decay [18]). This results in a significant reduction of required number of high-fidelity samples.

Our Approach



The highlights of the new approach are:

- *The error between grid levels is trained in terms of the solution through a neural network;*
- *Convolutional neural networks are used to learn local error features as latent quantities;*
- *Transfer learning is used to exploit the similarities between errors at different levels.*

1 Preliminaries: Multi-level UQ

We introduce notation for the different levels of solutions. The discretised operators corresponding to different levels are labelled with a superscript and are denoted as $L^{(i)}$, and the corresponding computational grids consisting of $N^{(i)}$ grid points are denoted as $X^{(i)}$. The solutions are denoted as $\mathbf{u}^{(i)}(\mathbf{z}) \in \mathbb{R}^{N^{(i)}}$ and are functions of the input uncertainties. In a multi-level method we use a hierarchy of grid resolutions consisting of a total of N_L grids. We denote with $\mathbf{u}^{(1)}(\mathbf{z})$ the solution which is computed on the coarsest grid level $X^{(1)}$, while $\mathbf{u}^{(N_L)}(\mathbf{z})$ is the solution computed on the finest grid level $X^{(N_L)}$. For simplicity the computational grids are assumed to be nested:

$$X^{(1)} \subset X^{(2)} \subset \ldots \subset X^{(N_L)} \,. \tag{3.1}$$

Several multi-level strategies have been presented that use a hierarchy in grid resolutions [4], which reduces the number of high-fidelity solves required to construct an accurate surrogate model or extract accurate statistics. Consider restriction of the fine-grid solution to the coarsest grid:

$$\mathbf{u}^{(N_L)}(\mathbf{z}) \to \mathbf{u}^{(N_L)}(\mathbf{z})|_{X^{(1)}} \,, \tag{3.2}$$

where $\cdot|_{X^{(1)}}$ is the restriction operator which computes values on the coarse grid $X^{(1)}$, which may incorporate sub sampling and interpolation. The multi-level methods described in [12, 14, 15] rewrite $\mathbf{u}^{(N_L)}(\mathbf{z})$, using the telescoping-sum identity:

$$\mathbf{u}^{(N_L)}(\mathbf{z})|_{X^{(1)}} = \mathbf{u}^{(1)}(\mathbf{z}) + \sum_{i=2}^{N_L} (\mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}) \,. \tag{3.3}$$

In case of nested grids, (3.1), the restriction operator solely uses sub-sampling. Restricting to the coarsest level is not necessary, as we can also prolongate the coarse solutions to the finest level computational grid $X^{(N_L)}$, but in many cases the coarsest grid suffices to accurately calculate the QoI. To circumvent interpolation of coarse-grid solution values to the fine grid and reduce the degrees of freedom in the neural networks that are used later, we opt to restrict solution values to the coarsest level. For convenience, we denote the error between two levels as

$$\mathbf{e}^{(i)}(\mathbf{z}) := \mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}} \,. \tag{3.4}$$

The goal of a multi-level method is to approximate the error between two levels $\mathbf{e}^{(i)}$, where we assume that $\mathbf{e}^{(i)} \to 0$ as $i \to \infty$. The variance decay in the magnitude of the terms in (3.3) decreases the number of samples required to approximate $\mathbf{e}^{(i)}(\mathbf{z})$ with increasing level i [12, 19], which reduces the total computational cost for constructing an accurate surrogate.

2 Our approach in short

Our method utilises the variance decay and is based on the fact that *subsequent errors themselves typically exhibit similar behaviour (as a function of the spatial variables) with increasing grid resolution*. This is a key insight that we will directly exploit in this chapter as it allows us to construct multi-level methods based on the concepts of convolutional neural networks and transfer learning. Our proposed method is as follows. Like in the MLSC approach [12], we use the telescopic-sum identity to express the solution in terms of the solution differences between consecutive grid levels ('relative errors', in the following mostly shortly 'errors'). Our method distinguishes itself in that we use the similarity of this error between consecutive grid levels to reduce the required number of samples on each level. We achieve this through an original combination of several ingredients.

First, we express the relative error between grid levels as a function of the solution at the coarser level through the use of a neural network. Convolutional neural networks enable this step. Convolutional neural networks are well-suited to learn local or low-level features ('latent quantities') in the solution structure, that turn out to be similar to the truncation error commonly encountered in numerical discretisation methods for PDEs. In connection with a second, fully connected neural network, the convolutional neural network is able to learn the error between grid levels as a function of the solution. Finally, we use transfer learning to reuse the weights and biases from previously trained neural networks at coarse levels to quickly train the neural networks at finer grid levels, thereby significantly reducing the required number of samples of high-fidelity solutions. In the remainder of this chapter we refer to the proposed approach as the Multi-Level Neural Network (MLNN) method.

3 Multi-level neural network surrogate construction

The MLNN method is also based on identity (3.3), but does not use interpolation for constructing the approximations for $\mathbf{e}^{(i)}$ as used in [12]. Instead we use a convolutional neural network that approximates $\mathbf{e}^{(i)}$ in terms of a set of local low-level features, also known as latent quantities. To show that a representation of $\mathbf{e}^{(i)}$ in a set of latent quantities is justified, the error $\mathbf{e}^{(i)}$ can be written as:

$$\begin{aligned}\mathbf{e}^{(i)} &= \mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}} \\ &= (\mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}_{\text{exc}}(\mathbf{z})|_{X^{(1)}}) - (\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}_{\text{exc}}(\mathbf{z})|_{X^{(1)}}) \\ &= \mathbf{E}^{(i)}|_{X^{(1)}} - \mathbf{E}^{(i-1)}|_{X^{(1)}} \,, \end{aligned} \tag{3.5}$$

where $\mathbf{E}^{(i)} = \mathbf{u}^{(i)}(\mathbf{z}) - \mathbf{u}_{\text{exc}}(\mathbf{z})|_{X^{(i)}}$ denotes the global error between the solution on level i and the exact solution (sub-sampled on $X^{(i)}$). The exact solution is often not known and therefore (3.5) is not directly useful. However, for a linear discretisation operator $L^{(i)}$, we can write:

$$\begin{aligned}L^{(i)}\mathbf{E}^{(i)} &= L^{(i)}(\mathbf{u}^{(i)} - \mathbf{u}_{\text{exc}}|_{X^{(i)}}) \\ &= L^{(i)}\mathbf{u}^{(i)} - L^{(i)}(\mathbf{u}_{\text{exc}}|_{X^{(i)}}) \\ &= -L^{(i)}(\mathbf{u}_{\text{exc}}|_{X^{(i)}}) \\ &= -\tau^{(i)} \,, \end{aligned} \tag{3.6}$$

where τ is the local truncation error. This can be rewritten as:

$$\mathbf{E}^{(i)} = -\left(L^{(i)}\right)^{-1}\tau^{(i)} \,, \tag{3.7}$$

The local truncation error is the error when substituting the exact solution into the discretised operator $L^{(i)}$ and is computed locally, for instance by performing Taylor-series expansions on the schemes that are used to discretise the differential operator $\mathscr{L}$. The inverse operator $\left(L^{(i)}\right)^{-1}$ maps the local truncation error to the global error $E^{(i)}$. For linear discretisation of linear PDEs this results in the form:

$$\tau^{(i)} = \sum_{k=1}^{\infty} c_k \, (\mathbf{u}_{\text{exc}})_{\partial,k}(\mathbf{z})|_{X^{(i)}} \,, \tag{3.8}$$

where $(\mathbf{u}_{\text{exc}})_{\partial,k}$ is a sequence of partial derivatives of the exact solution with respect to spatial/temporal coordinates. The truncation error can be interpreted as a set of latent quantities, which are

local or low-level features that can be used to effectively express the error in the solution. However, this only holds for linear differential equations and is therefore a rather restrictive assumption.

Our key idea is to construct an approximation for $\mathbf{e}^{(i)}$ in terms of a set of latent quantities. This set of latent quantities is obtained by applying a non-linear transformation of the solution values on a coarser level. For linear differential equations, the truncation error is a good candidate for such a set of latent quantities, but it does not generalise to non-linear problems. Therefore, we propose a new algorithm that approximates $\mathbf{e}^{(i)}(\mathbf{z}),\ i = 2,...,N_L$ using a neural network approach. The ***How*** and ***Why*** of this approach are discussed in more detail next.

How a neural network is used to approximate $\mathbf{e}^{(i)}(\mathbf{z})$

The neural network architecture that is used for approximating $\mathbf{e}^{(i)}(\mathbf{z})$ is shown in figure 3.1. The neural network consists of two stages:

Stage 1: Obtain latent/inferred quantities from the solution vector $\mathbf{u}^{(i)}(\mathbf{z})$***, that can be mapped efficiently to*** $\mathbf{e}^{(i)}(\mathbf{z})$

Stage 2: Construct mapping from latent quantities to $\mathbf{e}^{(i)}(\mathbf{z})$

The neural network takes as input the sub-sampled solution values $\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}$ and is trained such that it outputs $\mathbf{e}^{(i)}(\mathbf{z})$.

Stage 1, How?

The first part of the neural network that corresponds to stage 1 consists of a number of sequential convolutional layers. The filters that are used in convolutional neural networks are equivalent to a sequence of local finite difference operators whose coefficients are determined during the training procedure. As a result, the convolutional layers are used to efficiently combine the solution values into latent quantities.

Stage 1, Why?

Instead of constructing an approximation for $\mathbf{e}^{(i)}(\mathbf{z})$ directly (as done in for example [12]), we first construct a set of latent quantities from the solution vector $\mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}})$, which are then mapped to $\mathbf{e}^{(i)}(\mathbf{z})$. We learn from equations (3.6)-(3.7), that a mapping between the solution vector and $\mathbf{e}^{(i)}(\mathbf{z})$ exists in terms of the truncation error, which consists of a set of latent quantities. As a result, the MLNN method uses a sequence of convolutional layers to find a set of latent quantities that can be used by stage 2 of the neural network to efficiently construct an approximation for $\mathbf{e}^{(i)}(\mathbf{z})$.

Stage 2, How?

After propagating the input $\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}$ through the convolutional layers, the output is flattened into a single vector and feeds the second stage of the neural network. The second stage consists of multiple fully-connected layers that apply a non-linear transformation that maps the previously obtained latent quantities to the desired quantity $\mathbf{e}^{(i)}(\mathbf{z})$. Furthermore, as we want to approximate the dependency of $\mathbf{e}^{(i)}$ on $\mathbf{z}$, $\mathbf{z}$ is included in the neural network by concatenating the values of $\mathbf{z}$ to the flattened convolutional output.

Stage 2, Why?

Solely using convolutional layers is not preferred due to their inherent local nature, making them only useful for extracting low-level features. The latent quantities obtained in stage 1 need to be mapped to the desired quantity $\mathbf{e}^{(i)}(\mathbf{z})$. As in equation (12), this requires a global and possibly non-linear transformation. It is known that fully-connected neural networks are so-called universal-function approximators [20, 21, 22] and are therefore perfectly suited for constructing the non-linear

mapping between the latent quantities and $\mathbf{e}^{(i)}(\mathbf{z})$.

Stage 1 + Stage 2

The full architecture for a scalar partial differential equation with two space dimensions and uncertainties $\mathbf{z}$ is shown schematically in figure 3.1.

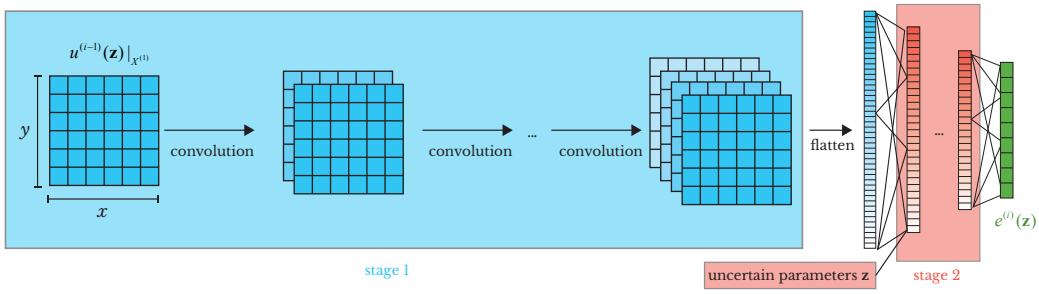


Figure 3.1: *The convolutional and fully-connected network architecture for approximating* $\mathbf{e}^{(i)}$ *for a scalar partial differential equation with two space dimensions and uncertainties* $\mathbf{z}$.

The architecture for the coupled vector partial differential equation case requires a multi-channel input, i.e., one channel for each of the calculated quantities, which is explained in more detail in section 4. We denote the neural network that maps $\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}$ to $\mathbf{e}^{(i)}(\mathbf{z})$ as,

$$\mathbf{e}^{(i)}(\mathbf{z}) \approx P^{(i)}(\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}) \,. \tag{3.9}$$

To clarify, the neural network $P^{(i)}$ requires a training set that comprises pairs of solutions

$$(\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}, \mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}}) \,, \tag{3.10}$$

where $\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}$ is used as input for the neural network, while $\mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}}$ enters the cost function during training. This is discussed in more detail in section 4. In order to utilise the telescopic-sum identity (3.3), we need approximations for $\mathbf{e}^{(i)}(\mathbf{z})$ with $i = 2, ..., N_L$ and therefore we train $N_L - 1$ neural networks $(P^{(2)}, ..., P^{(N_L)})$ of the form shown in figure 3.1. After training the neural networks, we can approximate the high-fidelity solution as:

$$\mathbf{u}^{(N_L)}(\mathbf{z})|_{X^{(1)}} \approx \mathbf{u}^{(1)}(\mathbf{z}) + \sum_{i=2}^{N_L} P^{(i)} \left(\mathbf{u}^{(1)}(\mathbf{z}) + \sum_{j=2}^{i-1} P^{(j)} \left(\mathbf{u}^{(1)}(\mathbf{z}) + \sum_{k=2}^{j-1} P^{(k)}(...) \right) \right) \,, \tag{3.11}$$

where we used (3.3) and recursively used:

$$\mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} = \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}} + \mathbf{e}^{(i)} \approx \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}} + P^{(i)}(\mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}) \,. \tag{3.12}$$

We denote the approximate high-fidelity solution as $\tilde{\mathbf{u}}^{(N_L)}(\mathbf{z}) \approx \mathbf{u}^{(N_L)}(\mathbf{z})|_{X^{(1)}}$. Notice that we only need $\mathbf{u}^{(1)}(\mathbf{z})|_{X^{(1)}}$ to construct an approximation for $\mathbf{u}^{(N_L)}(\mathbf{z})|_{X^{(1)}}$. The architecture, hyperparameters and training procedure of the neural networks will be discussed in more detail in section 4.

How the required number of samples for training $P^{(i)}$ is reduced

The combination of convolutional layers and fully-connected layers allows for learning complex non-linear mappings. The filter coefficients that are determined during training of the convolutional layers, have significantly less degrees of freedom when compared to fully-connected layers with

similar input-output dimensions. As a result, the number of unknowns that need to be determined during training is significantly lower than in a network architecture with the same number of layers using solely fully-connected layers. This drastically decreases the required number of training samples in the MLNN method.

Second, it can be shown that $\mathbf{e}^{(i-1)}(\mathbf{z})$ and $\mathbf{e}^{(i)}(\mathbf{z})$ have a similar spatial shape (apart from a scaling factor). This concept is summarised in the following theorem:

Theorem 1. *Assume the solution $\mathbf{u}|_{X^h}$ of the discretised problem can be expanded as follows:*

$$\mathbf{u}^{(h)} = \mathbf{u}_{exc}|_{X^h} + h^d \mathbf{v}|_{X^h} + O(h^{d+1}) \,, \tag{3.13}$$

where h is the grid resolution, d the order of convergence with $d \geq 1$, and $\mathbf{v}$ a function independent of h. Then the errors $\mathbf{e}^{(h/2)} := \mathbf{u}^{(h/2)}|_{X^h} - \mathbf{u}^{(h)}$ and $\mathbf{e}^{(h/4)} := \mathbf{u}^{(h/4)}|_{X^h} - \mathbf{u}^{(h/2)}|_{X^h}$ satisfy

$$c\mathbf{e}^{(h/4)} = \mathbf{e}^{(h/2)} + O(h^k) \,, \tag{3.14}$$

where $k \geq 1$ and where c is a constant independent of h.

Proof. Using the expansion (3.13), we can write

$$\mathbf{e}^{(h/2)} = \left(\left(\frac{h}{2}\right)^d - h^d \right) \mathbf{v}|_{X^h} + O(h^{d+1}) \,, \tag{3.15}$$

$$\mathbf{e}^{(h/4)} = \left(\left(\frac{h}{4}\right)^d - \left(\frac{h}{2}\right)^d \right) \mathbf{v}|_{X^h} + O(h^{d+1}) \,. \tag{3.16}$$

Multiplying the second equation with 2^d results in

$$2^d \mathbf{e}^{(h/4)} = \left(\left(\frac{h}{2}\right)^d - h^d \right) \mathbf{v}|_{X^h} + 2^d O(h^{d+1}) = \mathbf{e}^{(h/2)} + O(h^{d+1}) \,, \tag{3.17}$$

which is of the form displayed in equation (3.14) with $k = d + 1$. □

The existence of expansion (3.13) is a crucial assumption in this theorem. Notice that theorem 1 assumes a restriction to the grid X^h, while the proposed method restricts to the coarsest level $X^{(1)}$. However by assuming the nested property of consecutive grids, theorem 1 also holds for a restriction to the coarsest grid. It has been proven in [23] (theorem 2.1) that this expansion exists for a large class of (discretisations of) linear differential equations under relatively mild conditions. Roughly speaking, these conditions are: existence and uniqueness of the discrete and continuous problem, and sufficient smoothness of the terms appearing in the discretised equations.

Theorem 1 indicates that if an expansion of the form (3.13) holds, then information of a previously trained neural network mapping $P^{(i-1)}$ can be used to more efficiently train $P^{(i)}$. The concept of reusing parts of a previously trained neural network is not new in the field of machine learning, and is known as transfer learning [24]. Transfer learning is used to circumvent the need for sampling many solutions on the higher levels by using weights and biases from the previously trained neural network $P^{(i-1)}$.

The concept of similarity of error profiles, as expressed by theorem 1, holds for linear differential equations, as it relies on assumption (3.13). We numerically show in section 6 that the concept of similarity of error profiles also holds to a large extent for non-linear differential equations, which makes transfer learning applicable to non-linear cases as well.

4 Training the neural networks

In order to train the neural networks $(P^{(2)}, ..., P^{(N_L)})$, we need to specify:

1. ***Architecture***: layer architecture, activation functions, and hyperparameters
2. ***Training/validation data***: data set for training and validating the neural networks
3. ***Hyperparameter tuning***: finding proper hyperparameter values

We will discuss these three components separately in the next subsections.

4.1 Neural Network Architecture

As mentioned before we use a combination of convolutional layers and fully-connected layers, and the architecture is split in two stages.

Convolutional layers

The name "convolutional layer" indicates that the network employs a mathematical operation called convolution. A convolutional layer is simply a fully-connected neural network layer in which the dense matrix multiplication is replaced by a convolution operator (multiplication with a sparse matrix) [25], where filters are convolved over the space of current values. In general, an nD convolutional layer accepts n_c channels of dimension $k_1 \times k_2 \times ... \times k_n$ as input, and convolves a filter of dimension $f_1 \times f_2 \times ... \times f_n \times n_c$ over the input channels, adds a bias, and applies a (non-)linear transformation to produce the output. This procedure is shown schematically in figure 3.2.

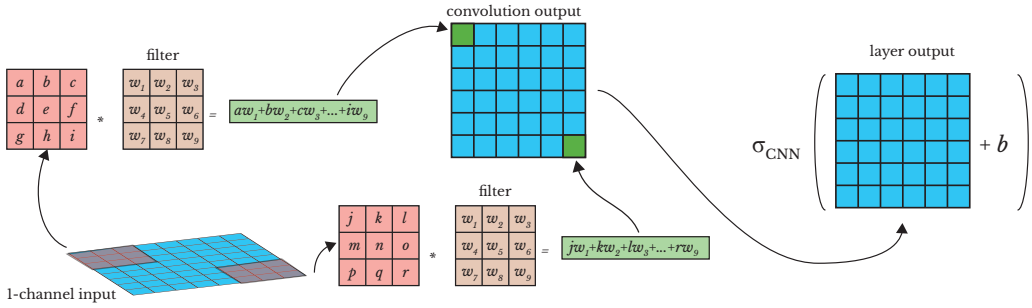


Figure 3.2: *The 2D convolutional layer with a 1-channel input.*

Figure 3.2 shows a 2D convolutional layer for a 1-channel input. The number of channels of the input corresponds to the number of quantities that is solved for in the underlying PDE. E.g., when solving 2D incompressible Navier-Stokes, we solve for the two velocity components and the pressure, which results in a 3-channel input. In this chapter, a single 3×3 filter is used with 9 unknown filter weights w_j, which is a commonly used filter-width that yields the best results on our test-cases (see section 6). In addition, a multi-channel input results in a multi-channel filter with more unknowns. It is common practice to use more than one filter, which results in an output with multiple channels, i.e., one channel associated to the convolution of one filter. Notice that the output of a convolutional layer is in general smaller than the input, as the filter cannot cross the boundaries of the input channel. Therefore, the dimensions of the output depend on the input dimensions and the filter dimensions. As a remedy, we can pad the input channel with zero values at the boundaries in

order to obtain an output which has the same dimensions as the input channel. This is often referred to as zero-padding, and will be used in this chapter to keep the output dimensions of the convolutional layers the same. After convolving the filters, a trainable bias coefficient is added to each output channel and the resulting values are activated with a non-linear activation function σ_{CNN}. The activation function σ for the convolutional neural network (CNN) is chosen to be the Rectified Linear Unit (ReLU):

$$\sigma_{\text{CNN}}(x) = \max(0, x) \ , \tag{3.18}$$

because it does not suffer from the vanishing gradient problem and allows for a sparse representation of the output [25]. The ReLU activation function may suffer from so-called dying neurons [25]. Other variants, e.g., leaky-Relu or ELU, may be used to resolve the dying neuron problem. However, the sparse representation when using ReLU is beneficial and is therefore employed in the remainder of this chapter. The number of convolutional layers N_{CNN} to choose, depends on the intrinsic complexity of the dataset and is treated as a hyperparameter in the MLNN method. Lastly, increasing the number of filters with a factor 2 for consecutive CNN layers is a good starting point to tune the neural network [25], and this strategy is therefore used in this chapter.

Fully-connected layers

After the input has been propagated through the convolutional layers, we flatten the output of the last convolutional layer into a single vector, and the uncertainties $\mathbf{z}$ are concatenated. This vector is the start of the fully-connected part, where we apply a non-linear transformation to the output of the convolutional part. This part of the neural architecture is characterised by the number of fully-connected layers N_{FC}, the number of neurons per layer n, and the activation function σ_{FC}. The parameters N_{FC} and n are treated as hyperparameters and will be tuned during the training process. Additionally, a ReLU activation function is used for the fully-connected layers. Once we have propagated the output of the convolutional part through the N_{FC} fully-connected layers, we output a vector which has the same dimensions as the input of the network. The output layer uses a linear activation function, because it allows for unbounded output values.

Cost function

The goal of training the neural network is to find the weights $\mathbf{w}$ and biases $\mathbf{b}$, of both the convolutional and fully-connected parts, which minimise the following cost functions:

$$c^{(i)}(\mathbf{w}, \mathbf{b}) = \sum_{(\mathbf{u}^{(i)}, \mathbf{u}^{(i-1)}) \in T^{(i)}} \| P^{(i)}(\mathbf{u}^{(i-1)}|_{X^{(1)}}) - (\mathbf{u}^{(i)}|_{X^{(1)}} - \mathbf{u}^{(i-1)}|_{X^{(1)}}) \|_2^2 \ , \quad i = 2, ..., N_L \ , \tag{3.19}$$

where $T^{(i)}$ is the training set. The construction of this training set is discussed in section 2.4 in more detail.

Overfitting may occur due to the large amount of unknowns, i.e., convolutional weights/biases and fully-connected layer weights/biases. In order to prevent overfitting, we use an l_2-regularisation, which introduces a new hyperparameter λ and the altered cost functions:

$$c_\lambda^{(i)}(\mathbf{w}, \mathbf{b}) = \sum_{(\mathbf{u}^{(i)}, \mathbf{u}^{(i-1)}) \in T} \| P^{(i)}(\mathbf{u}^{(i-1)}|_{X^{(1)}}) - (\mathbf{u}^{(i)}|_{X^{(1)}} - \mathbf{u}^{(i-1)}|_{X^{(1)}}) \|_2^2 + \lambda \|\mathbf{w}\|_2^2, \quad i = 2, ..., N_L \ . \tag{3.20}$$

After training the neural network, we validate if the neural network generalises to unseen data by computing the validation error:

$$v^{(i)} = \sum_{(\mathbf{u}^{(i)}, \mathbf{u}^{(i-1)}) \in V^{(i)}} \| P^{(i)}(\mathbf{u}^{(i-1)}) - (\mathbf{u}^{(i)} - \mathbf{u}^{(i-1)}) \|_2^2, \quad i = 2, ..., N_L \ , \tag{3.21}$$

where $V^{(i)}$ is the validation set. The validation error is used to tune the hyperparameters of the network architecture. Additionally, a test set can be used to check if the neural network with proper hyperparameters generalises well outside the training/validation set. However, using a test set results in an increase of required number of samples, and as generalisation is only important within or close to the range of training/validation data. We choose to solely use a training and validation set.

Transfer learning

The amount of training data required for obtaining a proper set of weights and biases may become infeasible when increasing the level, as it requires a training set that is constructed by sampling many high-fidelity solutions. When training $P^{(i)},\ i = 3,...,N_L$, transfer learning is used to circumvent the need for sampling many solutions on the higher levels, by using weights and biases from the previously trained neural network $P^{(i-1)}$. Transfer learning can be performed in multiple ways [24]. We choose to fix the weights and biases of the previously trained neural network $P^{(i-1)}$ and add a small number of fully-connected layers at the end, while keeping the rest of the architecture the same. As the input for $P^{(i)}, i = 2,...,N_L$ is a solution vector, which is defined on the same grid $X^{(1)}$, the learned convolutional filters are still expected to compute a proper set of latent quantities when increasing the level i. Therefore, the transfer learning approach leaves the convolutional part unaltered for all $P^{(i)},\ i = 3,...,N_L$. The transfer learning procedure is schematically shown in figure 3.3.

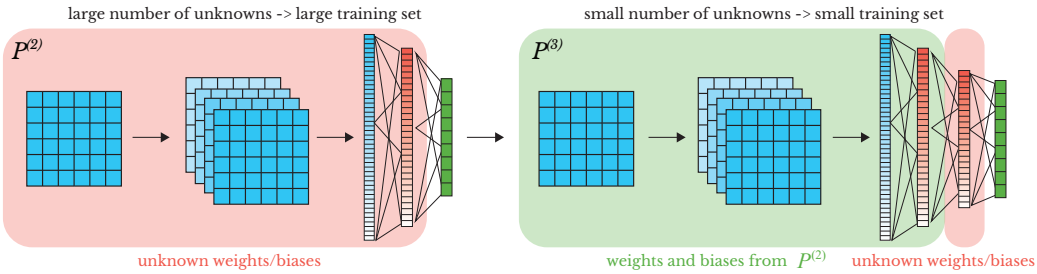


Figure 3.3: *Transfer learning procedure.*

A single fully-connected layer is added with the same activation function as the other fully-connected layers, but with a hyperparameter n representing the number of neurons in the new layer.

4.2 Training/validation data

The neural networks $\{P^{(i)}\}_{i=2}^{N_L}$ are trained to approximate the errors $\mathbf{e}^{(i)}(\mathbf{z})|_{X^{(1)}} = \mathbf{u}^{(i)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}^{(i-1)}(\mathbf{z})|_{X^{(1)}}$. The training set for training $P^{(i)}$ requires solution values for both $\mathbf{u}^{(i-1)}(\mathbf{z})$ and $\mathbf{u}^{(i)}(\mathbf{z})$, sampled at multiple values for $\mathbf{z}$. The training set that is used for training $P^{(i)}$ is denoted as:

$$T^{(i)} := \left\{ \left(\mathbf{u}^{(i)}(\mathbf{z_k})|_{X^{(1)}}, \mathbf{u}^{(i-1)}(\mathbf{z_k})|_{X^{(1)}} \right) \mid k = 1,...,N_{\text{training}} \right\} . \tag{3.22}$$

In order to validate if the neural network generalises to values not in the training set, we validate our neural network on the set:

$$V^{(i)} := \left\{ \left(\mathbf{u}^{(i)}(\mathbf{z_k})|_{X^{(1)}}, \mathbf{u}^{(i-1)}(\mathbf{z_k})|_{X^{(1)}} \right) \mid k = 1,...,N_{\text{validate}} \right\} . \tag{3.23}$$

The amount and quality of data that is used to train and validate the neural network is paramount. How to ensure that we obtain a proper set of weights and biases that generalises well to unseen values, is discussed in this section. The required amount of training data depends on the complexity

of the response to be approximated and on the total number of unknowns in the neural network architecture, i.e., the weights and biases. The numbers of unknown weights and biases in the neural networks $\{P^{(i)}\}_{i=2}^{N_L}$ decrease significantly by using the transfer learning approach described in section 4.1. As a result, we need a large amount of training samples when training $P^{(2)}$, which is feasible when assuming that low-fidelity solutions are computationally cheap to sample. As opposed to this, sampling high-fidelity solutions for training $P^{(N_L)}$ is computationally expensive, but we require significantly less training samples, due to variance decay in $\mathbf{e}^{(i)}$ for increasing i [19] and the small amount of unknown weights and biases induced by transfer learning. The quality of the training data is determined by the sample locations $\mathbf{z}_k$ in (3.23). As determining a proper size of the training/validation set is difficult [25], we employ a sampling strategy which iteratively adds new samples $\mathbf{u}^{(i)}(\mathbf{z}_k)$ when the neural network does not generalise well to the validation set.

The training procedures for the neural networks $\{P^{(i)}\}_{i=2}^{N_L}$ are explained in more detail below.

Training $P^{(2)}$

The first neural network that needs to be trained is $P^{(2)}$. When training $P^{(2)}$, we approximate the error

$$\mathbf{e}^{(2)}(\mathbf{z})|_{X^{(1)}} = \mathbf{u}^{(2)}(\mathbf{z})|_{X^{(1)}} - \mathbf{u}^{(1)}(\mathbf{z}) \,, \tag{3.24}$$

and to construct $T^{(2)}$, the sample locations $\mathbf{z}_k$ need to be specified. As the low-fidelity solutions $\mathbf{u}^{(1)}(\mathbf{z})$ and $\mathbf{u}^{(2)}(\mathbf{z})$ are assumed to be computationally cheap to sample from, we place many samples on the first two levels. In this chapter, the initial sample set comprises a relatively large set of $10^{\dim(I)}$ solutions in the random space I, which shows to give good results for our test-cases in section 6. However, as we employ an iterative sampling strategy, the final size of the sample set is tailored to the underlying problem, which makes a proper size of the initial sample set less significant. The locations are defined using Monte Carlo sampling according to the underlying PDF of the uncertainties $\rho(\mathbf{z})$. As transfer learning can not be used for training $P^{(2)}$, the number of unknown weights and biases is large compared to $P^{(i)}$, $i = 3, \ldots, N_L$. As a result, the initial sample set should be large enough to properly train the neural network, but also small enough to be feasible in high-dimensional random spaces. However, having the dimensional scaling in the sample size of the initial set introduces a curse of dimensionality. One may opt to reduce the size of the initial samples set with the risk of needing to perform more learning cycles which affects the overall computational costs. The sample set is split using the 80/20%-rule in a training set $T^{(2)}$ and a validation set $V^{(2)}$. Splitting the full sample set according to this 80/20% ratio is known as the Pareto Principle [25] and is commonly used in machine learning for determining the size of the training and validation sets. After minimising (3.20), we tune the hyperparameters and pick the network architecture which results in the smallest validation error $v_{\text{min}}^{(i)}$. This is discussed in more detail in section III. Training is stopped if $v_{\text{min}}^{(i)}$ satisfies:

$$v_{\text{min}}^{(i)} < \varepsilon \,, \tag{3.25}$$

where ε is a specified threshold. The threshold ε indicates how well the trained neural network generalises to data that is not contained in the training set. If the stopping criterion is not met, we increase the sample set by sampling another $10^{\dim(I)}$ new PDE solutions. The newly obtained larger sample set is again split randomly in a training/validation set following the 80/20%-rule, which are used to retrain the neural network. This process of training, validating, enlarging the training set is repeated until (3.25) is satisfied. Notice that each time the sample set is increased, the neural network needs to be retrained, which is assumed to be a relatively cheap operation when compared to computing a high-fidelity solution, as the weights and biases of the previous training can be used as a very good initial guess for the retraining.

Training $P^{(i)}$, $i > 2$

Constructing a large training set when increasing i is often not feasible. Therefore, the combination of transfer learning and iterative sample set construction is used to limit the required number of training samples. Training the relatively small neural networks is fast in general. We iteratively increase the size of the training set during the training procedure. We again start with a small sample set that is used for training/validation of the neural network. After training the neural network we check if (3.25) is met. If not, then new samples are added to the sample set, which effectively increases the size of the training/validation set. This process is repeated until (3.25) is satisfied.

Initially, the sample set comprises $2^{\dim(I)}$ randomly placed samples in the space I, which is split in a training set $T^{(i)}$ and validation set $V^{(i)}$ following the 80/20%-rule. The size of the initial sample set is smaller when compared to the initial sample set used for training $P^{(2)}$ to alleviate computational burden when sampling the computationally expensive higher-fidelity solutions. The number of samples in the initial sample set is chosen to be small in order for the MLNN method to still be feasible for high-dimensional random spaces. As mentioned before, the use of an iterative sampling strategy makes the size of the initial sample set insignificant. Furthermore, notice that we only need to sample solutions for $\mathbf{u}^{(i)}$ and $\mathbf{u}^{(1)}$, as accurate approximations of $\mathbf{u}^{(i-1)}$ can be constructed using (3.11) with $N_L = i - 1$.

Nesting of samples on different levels removes the need for sampling additional solutions for $\mathbf{u}^{(i-1)}$, but introduces correlated expectations on the different levels, which is unwanted [18]. Strategies to reuse samples on different levels have recently been proposed [26], and could be used to further enhance the MLNN method, but are not required to show the basic methodology that we propose.

After training the neural network, we check if criterion (3.25) is met. If the stopping criterion is not met, we increase the sample set by sampling $2^{\dim(I)}$ new PDE solutions. The newly obtained larger sample set is again split randomly in a training/validation set following the 80/20%-rule, which is used to retrain the neural network. This process of training, validating, enlarging the training set, is repeated until we satisfy (3.25).

4.3 Hyperparameter tuning

When training the neural networks, values for the hyperparameters need to be specified. A complete list of hyperparameters in this chapter is shown below:

- Hyperparameters when training $P^{(2)}$:
  - λ (regularisation parameter)
  - N_{CNN} (number of convolutional layers)
  - N_{FC} (number of fully-connected layers)
  - n (number of neurons per fully-connected layer)
- Hyperparameters when training $P^{(i)}$, $i > 2$:
  - $\lambda^{(i)}$ (regularisation parameter)
  - $n^{(i)}$ (number of neurons in the added fully-connected layer)

Notice that the hyperparameters directly affect the effective number of degrees of freedom in the neural network. This allows us to find an architecture with a proper number of learnable parameters during training, which is expected to reduce the required number of samples for training each neural network. To tune these hyperparameters, we use a grid search, which specifies pre-defined values for each parameter, and constructs a tensor grid of all possible combinations of hyperparameter

values. A neural network is then trained for each combination of hyperparameter values. To limit the total number of neural networks we have to train, we pick only 3 values for each hyperparameter, which are commonly used in deep-learning approaches:

- $\lambda \in \{0, 10^{-6}, 10^{-3}\}$,
- $N_{\text{CNN}} \in \{2, 4, 6\}$,
- $N_{\text{FC}} \in \{1, 3, 5\}$,
- $n \in \{\lfloor \frac{N^{(1)}}{2} \rfloor, N^{(1)}, 2N^{(1)}\}$.

This range of values for the hyperparameters has been shown to work well for many cases [24, 25, 27, 28, 29, 30] and is therefore employed in this chapter.

5 Complete algorithm

A schematic representation of the MLNN method is shown in figure 3.4.
The number of levels that should be picked, depends on the decay of $\mathbf{e}^{(i)}$ with increasing i. As a result, an intuitive criterion for adding new levels is:

$$\text{if } \exists \mathbf{u}^{(N_L-1)} \in T^{(N_L)} : \|P^{(N_L)}(\mathbf{u}^{(N_L-1)})\|_2 > \varepsilon_{\text{acc}} \rightarrow N_L = N_L + 1 \ , \tag{3.26}$$

where ε_{acc} is the accuracy tolerance value. To clarify, two thresholds need to be specified, i.e., ε in (3.25) and ε_{acc} in (3.26). As mentioned before, the threshold ε indicates how well the trained neural network generalises to data not contained in the training set. The threshold ε_{acc} corresponds to the accuracy of the approximated high-fidelity solution when using (3.11). In general, ε_{acc} is set to the required accuracy of the approximation, and ε is set to a value which is one or two orders of magnitude smaller than ε_{acc} to ensure that the accuracy of the approximate high-fidelity solution is not influenced by possibly poor training of the neural networks.

Performing in-depth analysis on the stability and convergence of the MLNN method is cumbersome due to the inherent non-linearity in the neural network architecture. A wide range of numerical results has shown that MLNN converges to a solution which fits the training/validation set. To clarify, the required number of training/validation samples is often small which results in easy to train neural networks that will almost surely converge, but are sensitive to overfitting. This overfitting is circumvented by introducing the regularisation strategy proposed in section 4. Important to note is that errors from neural networks on the low levels $P^{(2)}, P^{(3)}, \ldots$ do propagate to the higher levels. This can be directly seen in equation (3.11), where the output of a neural network on a specific level enters as input of a neural network on a higher level. The effect of this error propagation can be controlled by setting the threshold ε. Low values for ε result in more accurate predictions of a neural network, hence reducing the magnitude of the errors that will propagate through the neural network chain (3.11).

The MLNN method is flexible and robust and works for a wide variety of problems, some of which are shown in the next section. We note that the assumption that $\mathbf{e}^{(i-1)}(\mathbf{z})$ and $\mathbf{e}^{(i)}(\mathbf{z})$ should possess similar behaviour is a key assumption in the MLNN method. If this assumption does not hold, the transfer learning approach is not optimal and one might still require many samples for approximating $\mathbf{e}^{(i)}(\mathbf{z})$. The extension of transfer learning to problems where the similarity assumption does not hold will require further research in transfer learning; this is an open topic of research in the field of machine learning and it is still difficult to estimate how much $\mathbf{e}^{(i-1)}(\mathbf{z})$ and $\mathbf{e}^{(i)}(\mathbf{z})$ may deviate and which features are transferable [31].

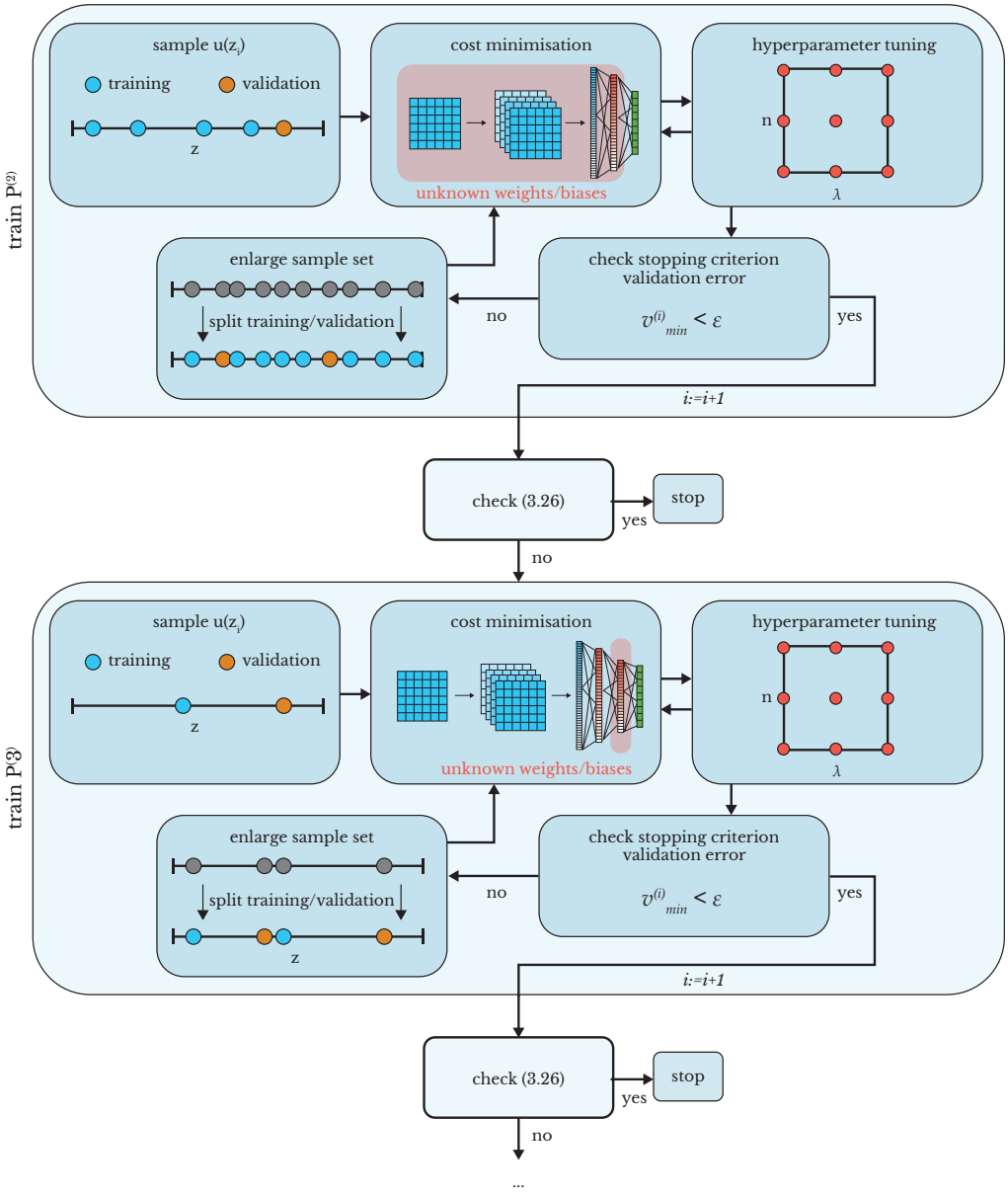


Figure 3.4: *Schematic overview of the MLNN method.*

6 Results

In this section we use the MLNN method to do surrogate construction for test-cases with ranging complexity. These surrogates may be used to extract statistical quantities, e.g., mean and variance. As the purpose of this chapter is to construct accurate surrogates, we assume that all the uncertain parameters are uniformly distributed, which does not put any emphasis on certain parameter configurations. The neural networks are constructed and trained using Tensorflow [32], which is a highly optimised library for machine learning.

6.1 Steady-state advection diffusion equation

In this section we study the following:

- Construction of a parametric solution for a linear 1D differential equation.
- Behaviour of $P^{(i)}$ for a linear differential equation.
- Extrapolation outside the training domain.
- Comparison with MLSC [12].
- Optimality of architecture choice.
- Performance in high-dimensional random space.

Parametric solution

We employ the MLNN method for constructing a parametric solution for the 1D steady-state advection diffusion equation:

$$\frac{\mathrm{d}u}{\mathrm{d}x} - \frac{1}{\mathrm{Re}}\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = 0, \quad u(0) = 0, \quad u(1) = 1, \quad x \in [0,1]\,, \tag{3.27}$$

where Re is the Reynolds number and is uncertain, i.e., $\mathbf{z} = \mathrm{Re}$. The exact solution is given by:

$$u_{\mathrm{exc}}(x, \mathrm{Re}) = \frac{\exp(x\mathrm{Re}) - 1}{\exp(\mathrm{Re}) - 1}\,. \tag{3.28}$$

We aim to construct a parametric solution for u as a function of $\mathrm{Re} \in [1, 100]$.

The equations are discretised using a finite-difference approach on an equidistant grid with a resolution of Δx with $N+1$ grid points. The solution vector on the computational grid $\mathbf{u} = (u_i)_{i=0}^N \approx (u(x_i))_{i=0}^N$, with $x_i = i\Delta x$ where $\Delta x = 1/N$, is obtained by solving the following linear system:

$$L_1\mathbf{u} - \frac{1}{\mathrm{Re}}L_2\mathbf{u} = \mathbf{S}(\mathrm{Re})\,, \tag{3.29}$$

where

$$L_1 = \frac{1}{2\Delta x}\begin{pmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 0 & 1 \\ & & 0 & -1 & 0 \end{pmatrix}, \quad L_2 = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & 0 & 1 & -2 \end{pmatrix}, \tag{3.30a}$$

$$\mathbf{S}(\mathrm{Re}) = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ 0 \\ -\frac{1}{2\Delta x} + \frac{1}{\mathrm{Re}\Delta x^2} \end{pmatrix}, \tag{3.30b}$$

where the boundary conditions enter the discretised equation via the vector $\mathbf{S}$. The discretisation scheme is second order accurate and therefore the error in the solution converges with $O(\Delta x^2)$,

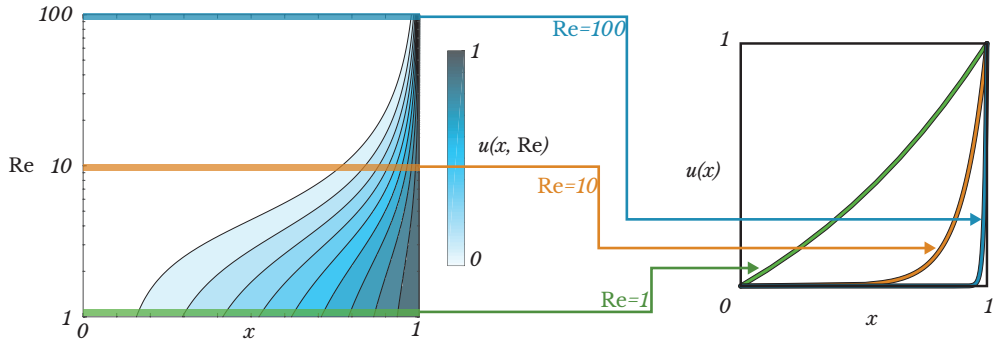


Figure 3.5: *Advection diffusion equation. The exact parametric solution for the advection diffusion equation.*

assuming that, on the basis of the Lax-equivalence theorem, the solution method to be used is numerically stable. The exact parametric solution for $\mathrm{Re} \in [1, 100]$ is shown in figure 3.5. The fidelity of the solution increases when N increases, and we choose to increase the fidelity by increasing the number of grid points with a factor 2 for consecutive levels, i.e., $N^{(i)} = 2N^{(i-1)}$. In order for the discretisation to produce stable results, the cell Reynolds number $\mathrm{Re}_{\Delta x} = \mathrm{Re}\Delta x$ should satisfy $\mathrm{Re}_{\Delta x} < 2$, which is satisfied by picking Δx sufficiently small. The largest value for Re is 100, and therefore picking $N^{(1)} = 100$ satisfies the cell Reynolds condition.

The parametric solution is constructed using an accuracy tolerance $\varepsilon_{\mathrm{acc}} = 10^{-6}$, which corresponds to the desired accuracy of the surrogate. The training tolerance is set to a value which is two orders of magnitude smaller, $\varepsilon = 10^{-8}$, following the rule explained in section 5. As mentioned in section 4.2, we start with 10 samples on the coarsest level (8 training samples, 2 validation samples), and apply the MLNN method from there on. The error between our approximate solution (3.11) and the exact solution (3.28) for a varying number of levels is shown in figure 3.6. The er-

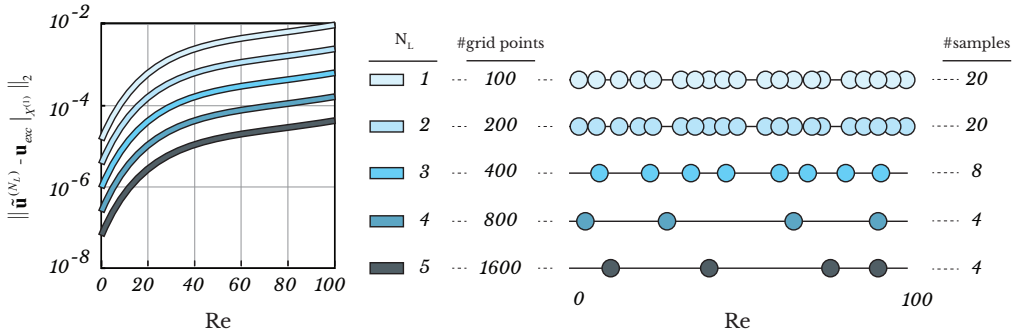


Figure 3.6: *Advection diffusion equation. (left) Error behaviour for different number of total levels. (right) Sample placement on each level.*

ror increases with increasing Reynolds number, due to the more difficult approximation of the thin boundary layer at high Reynolds numbers. The spacing between the errors for different levels at a specific Reynolds number (left figure) indicates that the method is indeed second order accurate. As expected, we see a clear decrease in required number of samples with increasing level, which indicates that the transfer learning is positively affecting the required number of samples on each level. Furthermore, the difference in training time for each consecutive neural network is negligible when compared to the cost of a single high-fidelity solve.

Neural network mappings $P^{(i)}$

The neural network mappings $P^{(i)},\ i = 2,3,4,5$ are shown in figure 3.7. Important to note is that

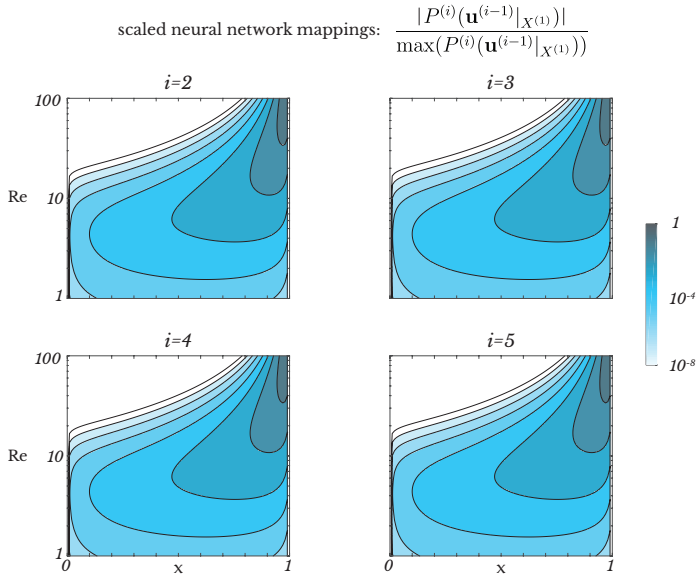


Figure 3.7: *Advection diffusion equation. The neural network mappings for the advection-diffusion test-case in (x, Re)-space.*

$P^{(i)},\ i = 2,3,4,5$ are approximately equal, apart from a scaling factor, which is the key insight utilised by our transfer-learning approach. This is as expected, because it can be shown that discrete solutions of this particular steady-state advection-diffusion boundary value problem satisfy the conditions outlined in [23] (Theorem 2.1) so that expansion (18) holds [33, 34], and therefore theorem 1 holds for this case. As mentioned before, when the level increases, the error between consecutive levels decreases. As a result, less samples are required to approximate these high-level mappings, which decreases the required number of high-fidelity samples.

Extrapolation capabilities

Figure 3.6 shows that the MLNN method is suitable for efficiently constructing parametric solutions inside the training domain $\mathrm{Re} \in [1, 100]$. However, extrapolation outside the domain where the neural networks are trained, is difficult in general. Figure 3.8 shows the errors between our approximate solution, based on $P^{(i)},\ i = 2,3,4,5$ and computed with (3.11), and the exact solution (3.28) for $\mathrm{Re} > 100$. The errors in the approximate solution increase rather drastically when using the trained neural networks outside the training domain. As a result, the trained neural networks are not suitable for constructing accurate solutions outside the training domain, when deviating too far from the domain boundaries [25]. This is a common problem in machine learning. Fixing this problem is not the scope of this manuscript.

Comparison with MLSC

We compare the computational cost of MLNN and MLSC (as proposed in [12]) for the case of constructing a surrogate with a specified accuracy. The computational cost comprises CPU time for

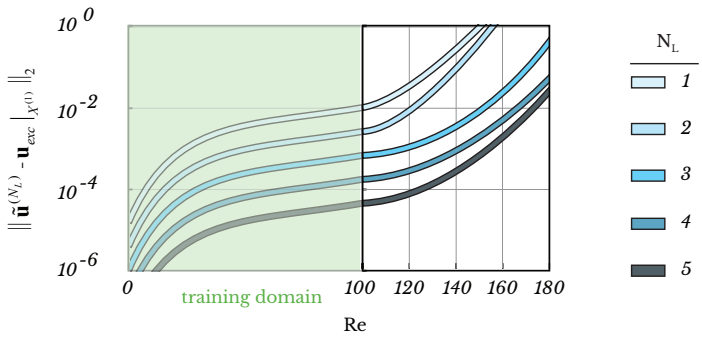


Figure 3.8: *Advection diffusion equation. Extrapolation errors outside the training domain* $Re \in [1, 100]$.

both solving the underlying differential equation and sampling procedure, i.e., training the neural networks (MLNN) or constructing the Clenshaw-Curtis grids (MLSC). The dependence of the computational cost on the implementation is negligible, because MLSC requires a minimum amount of implementation whereas training and constructing the neural networks for the MLNN method are performed using Tensorflow, which is a highly optimised library for machine learning. Both approaches use a sampling threshold on each level of $\varepsilon = 10^{-10}$. The levels are determined by the grid resolution which is refined with a factor 2 for subsequent levels. As both approaches use the same solver with the same grid resolution on each level, we are able to compare both approaches directly. The computational costs are scaled with respect to the maximum computational cost of the MLSC approach and the results are shown in figure 3.9. Notice that the error on the horizontal

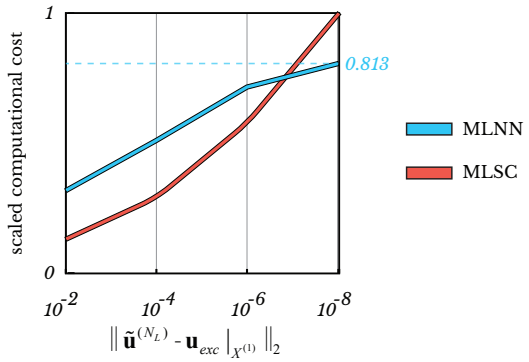


Figure 3.9: *Advection diffusion equation. The scaled computational cost when constructing a surrogate for a range of accuracies.*

axis is directly related to the number of levels that are used in both approaches. The MLNN method requires more computational time for the approximation on the first level when compared to the MLSC. However, the transfer learning approach significantly reduces the required number of samples on subsequent levels. As a result, MLNN appears to be more computationally efficient with increasing surrogate accuracy.

Performance in high-dimensional random space

In this section we show how the MLNN performs in a high-dimensional random space. We again consider the steady state advection diffusion equation, but now with a spatially varying Reynolds number:

$$\frac{\mathrm{d}u}{\mathrm{d}x} - \frac{1}{\mathrm{Re(x)}}\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = 0, \quad u(0) = 0, \quad u(1) = 1, \quad x \in [0,1]\,, \tag{3.31}$$

where the Reynolds number is represented by:

$$\mathrm{Re}(x) = \sum_{i=0}^{N_z-1} z_i x^i\,, \tag{3.32}$$

where the uncertainties are $z_i \in [1, 100/N_z]$, such that $\mathrm{Re}(x) \in [1, 100]$. We will study how the required computational cost, i.e., constructing training/validation set and training the neural networks, for constructing a surrogate $\tilde{\mathbf{u}}^{(5)}$. To get an impression of how the method scales to multiple dimensions, we show the computational cost per dimension and scale it with respect to the computational cost required for $N_z = 1$. The results are shown in figure 3.10. We see a clear decrease in the

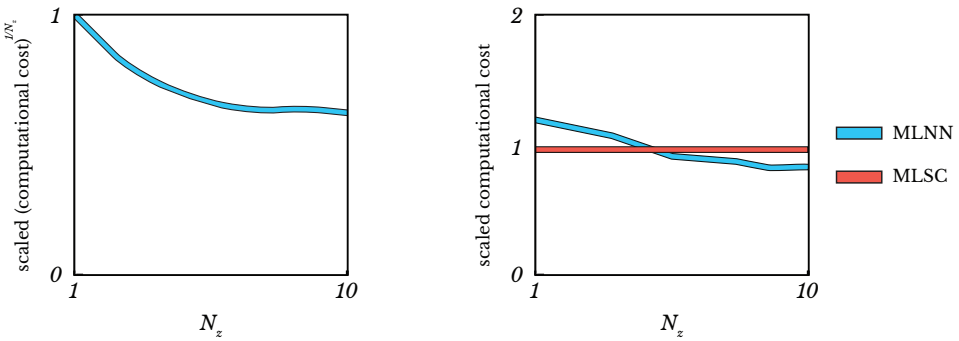


Figure 3.10: *Advection diffusion equation. (left) Scaled computational cost per dimension for the MLNN when constructing* $\tilde{\mathbf{u}}^{(5)}$*. The costs per dimension are scaled with respect to the cost required when* $N_z = 1$*. (right) comparison with the MLSC method for constructing a surrogate model for* $\mathbf{u}^{(5)}$*. The computational costs are scaled with respect to the computational costs required when using the MLSC method.*

required computational cost per dimension, which stagnates around $N_z = 7$, which indicates that the curse of dimensionality still exists. Furthermore, we see that MLSC outperforms MLNN up to $N_z = 5$, after which the MLNN method is superior.

Fully connected vs. Convolutional

This subsection shows that convolutional neural networks outperform fully-connected neural networks when applied to the steady-state advection diffusion equation. In order to fairly compare both the fully-connected and convolutional approach, we keep the number of learnable parameters in both approaches the same and also apply hyperparameter tuning for the number of layers and neurons per layer in the fully-connected neural network approach. The maximum allowed number of fully-connected layers/neurons for $P^{(i)}$ is chosen such that the total number of learnable parameters is not higher than the number of learnable parameters used in the convolutional approach. The scaled computational cost to achieve a certain accuracy is shown in figure 3.11. We see that the CNN architecture has a slight advantage over using a fully connected neural network. This advantage increases with the complexity of the underlying problem, but is not shown in the remainder of this paper for repetitiveness.

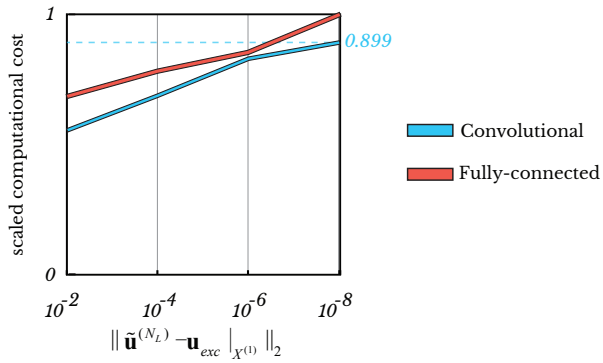


Figure 3.11: *Advection diffusion equation. Comparison of computational cost between a fully-connected neural network architecture and convolutional neural network architecture.*

6.2 Steady-state Burgers equation

In this section we increase the complexity of the previous test-case by introducing a non-linearity in the underlying differential equation. We study the following:

- Construction of a parametric solution for a non-linear 1D differential equation.
- Behaviour of $P^{(i)}$ for a non-linear differential equation.
- Comparison with MLSC [12].
- Performance in high-dimensional random space.

Parametric solution

In order to study how the MLNN method performs in the presence of non-linearities in the underlying equation, we consider the non-linear steady-state Burgers equation:

$$\frac{1}{2}\frac{\mathrm{d}u^2}{\mathrm{d}x} - \frac{1}{\mathrm{Re}}\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = 0, \quad u(0) = 0, \quad u(1) = 1, \quad x \in [0,1]\,, \tag{3.33}$$

where Re is again the Reynolds number and is assumed to be uncertain, i.e., $\mathbf{z} = \mathrm{Re}$. The effect of the non-linearity increases for increasing Reynolds number. In the linear advection-diffusion case we considered $\mathrm{Re} \in [1,100]$, but this range is increased to have a more pronounced non-linear effect in the solution at higher Reynolds numbers. As a result, we aim to construct a parametric solution for u as a function of $\mathrm{Re} \in [1,1000]$.

The equations are discretised using a finite-difference approach on an equidistant grid with a resolution of Δx with $N+1$ grid-points. The solution vector on the computational grid $\mathbf{u} = (u_i)_{i=0}^N \approx (u(x_i))_{i=0}^N$, with $x_i = i\Delta x$ where $\Delta x = 1/N$, is obtained by solving the following non-linear system:

$$\mathbf{F}(\mathbf{u}) = 0\,, \tag{3.34}$$

where

$$\mathbf{F}(\mathbf{u}) = \begin{pmatrix} u_0 \\ \frac{1}{4\Delta x}(u_2^2 - u_0^2) - \frac{1}{\mathrm{Re}\Delta x^2}(u_2 - 2u_1 + u_0) \\ \vdots \\ \frac{1}{4\Delta x}(u_N^2 - u_{N-2}^2) - \frac{1}{\mathrm{Re}\Delta x^2}(u_N - 2u_{N-1} + u_{N-2}) \\ u_N - 1 \end{pmatrix}. \tag{3.35}$$

This non-linear system is solved using Newton iteration and the complete discretisation scheme is second order accurate. We choose to increase the fidelity by increasing the number of grid-points with a factor 2 for consecutive levels, i.e., $N^{(i)} = 2N^{(i-1)}$, and picking $N^{(1)} = 300$. The grid-resolution of the first level is picked such that we produce stable results for $\mathrm{Re} \in [1, 1000]$. The parametric solution for $\mathrm{Re} \in [1, 1000]$ is shown alongside the solution error convergence for $\mathrm{Re} = 1000$ in figure 3.12. As for the the linear case, the tolerance for the training procedure is set to

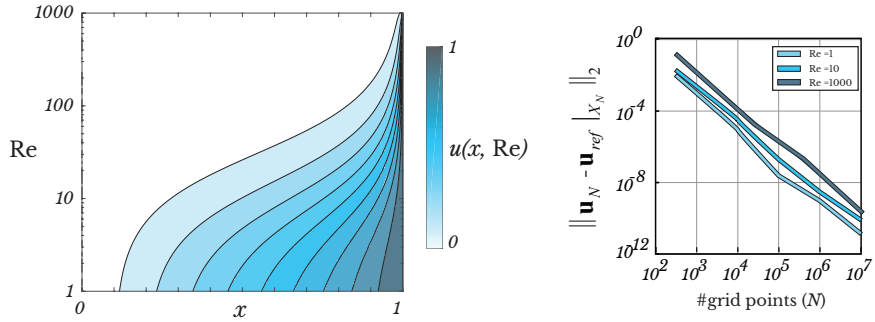


Figure 3.12: *Burgers equation. (left) The parametric solution for the Burgers equation computed with $N = 10^5$ grid points. (right) The convergence of the error in the solution for $Re = 1, 10, 1000$ as a function of the number of grid points. The reference solution is computed with $N = 10^8$.*

$\varepsilon = 10^{-8}$ and the accuracy tolerance is set to $\varepsilon_{\mathrm{acc}} = 10^{-6}$. As mentioned in section 4.2, we start with 10 samples on the coarsest level (8 training samples, 2 validation samples), and apply the MLNN method from there on. The error between our approximate solution (3.11) and the converged solution, with a varying number of levels, is shown in figure 3.13. Again, we see a decrease in required number of samples with increasing level. The error increases with increasing Reynolds number,

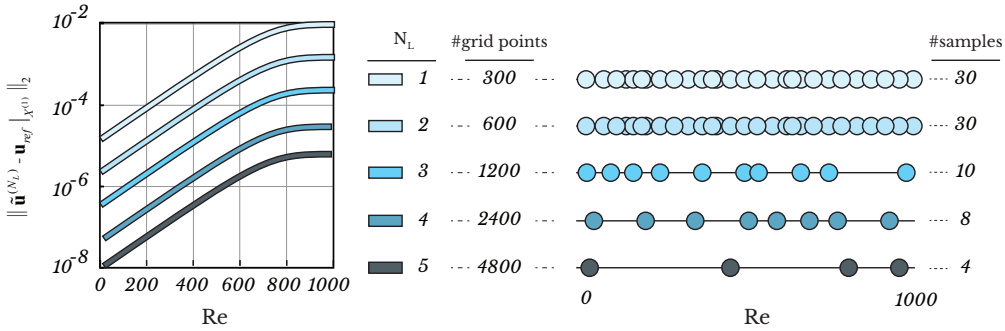


Figure 3.13: *Burgers equation. (left) Error convergence for different number of total levels. (right) Sample placement on each level.*

due to the more difficult approximation of the thin boundary layer at high Reynolds numbers and possibly the increasing effect of the non-linearity in the underlying equation.

Neural network mappings $P^{(i)}$

The neural network mappings $P^{(i)}$, $i = 2, 3, 4, 5$ are shown in figure 3.14. The approximated mappings $P^{(i)}$ differ slightly for consecutive levels, which is caused by the increasing effect of the non-linearity in the underlying PDE for higher Reynolds numbers. Note that for $\mathrm{Re} \in [1, 100]$ the

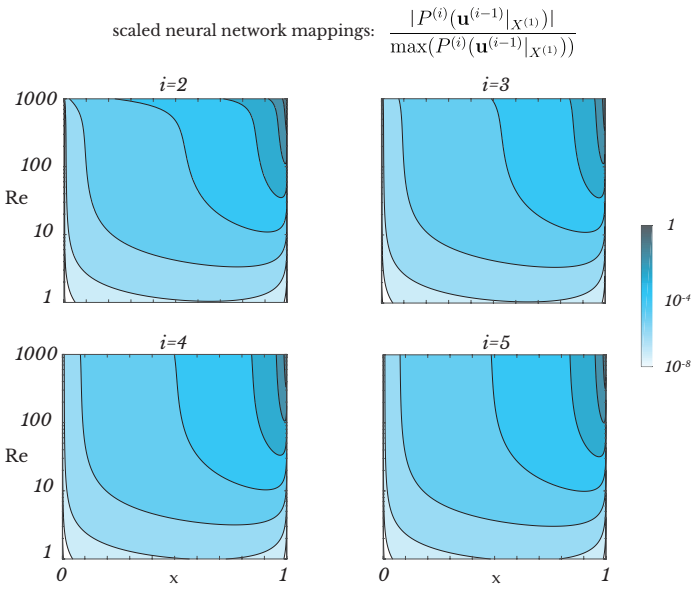


Figure 3.14: *Burgers equation. The neural networks mappings for the Burgers test-case in (x, Re)-space.*

mappings for the consecutive levels are very similar, just as with the linear advection-diffusion equation.

To summarise, the MLNN method shows similar result when comparing it to the linear advection-diffusion test-case. The number of samples required on each level increased, due to the more complex error responses $\mathbf{e}^{(i)}(\mathbf{z})$, which is caused by the increasing non-linearity in the underlying PDE with increasing Reynolds number. However, as the error responses are still similar, our proposed transfer learning approach is a very efficient way to learn the neural network mappings with increasing i.

Comparison with MLSC

We compare the computational cost of MLNN and MLSC for the case of constructing a surrogate with a certain accuracy. The computational cost is computed in the same way as described in section 6.I.4 and the results are shown in figure 3.15. The MLNN method requires significantly more computational time for the approximation on the first level when compared to the MLSC. Again we benefit by using our approach for higher accuracy levels.

Performance in high-dimensional random space

In this section we show how the MLNN performs in a high-dimensional random space for an underlying non-linear PDE. We again consider the steady-state Burgers equation, but with a spatially varying Reynolds number:

$$\frac{1}{2}\frac{\mathrm{d}u^2}{\mathrm{d}x} - \frac{1}{\mathrm{Re(x)}}\frac{\mathrm{d}^2u}{\mathrm{d}x^2} = 0, \quad u(0) = 0, \quad u(1) = 1, \quad x \in [0,1]\,, \tag{3.36}$$

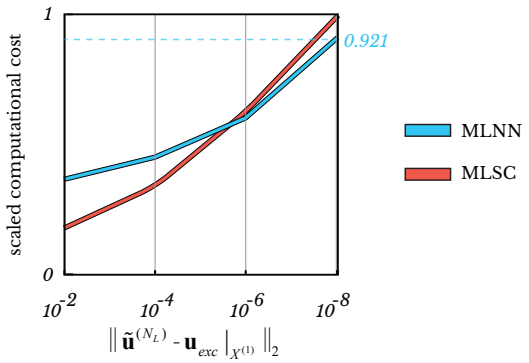


Figure 3.15: *Burgers equation. The scaled computational cost when constructing a surrogate for a range of accuracies.*

where the Reynolds number is represented by:

$$\mathrm{Re}(x) = \sum_{i=0}^{N_z-1} z_i x^i \,, \tag{3.37}$$

where the uncertainties are $z_i \in [1, 100/N_z]$, such that $\mathrm{Re}(x) \in [1, 100]$. The setup of this test-case is similar to the one used in section 6.I.5. We will study how the required computational cost, i.e., the cost of constructing training/validation set and training the neural networks, and the cost for constructing a surrogate $\tilde{\mathbf{u}}^{(5)}$, scales. To get an impression of how the method scales to multiple dimensions, we show the computational cost per dimension and scale it with respect to the computational cost required for $N_z = 1$. The results are shown in figure 3.16. We see a clear

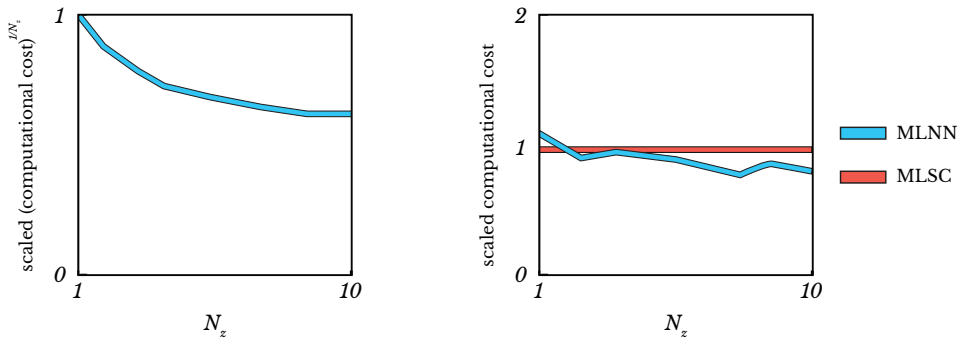


Figure 3.16: *Burgers equation. (left) Scaled computational cost per dimension for the MLNN when constructing $\tilde{\mathbf{u}}^{(5)}$. The costs per dimension are scaled with respect to the cost required when $N_z = 1$. (right) comparison with the MLSC method for constructing a surrogate model for $\mathbf{u}^{(5)}$. The computational costs are scaled with respect to the computational cost required when using the MLSC method.*

decrease in the required computational cost per dimension, which stagnates around $N_z = 8$, which indicates that the curse of dimensionality still exists. Furthermore, we see that MLSC outperforms MLNN up to $N_z = 3$, after which the MLNN method is superior.

6.3 Steady-state incompressible Navier-Stokes equations

In this section we study the following:

- Construction of a parametric solution for the 2D steady-state incompressible Navier-Stokes equations.
- Behaviour of $P^{(i)}$ for the steady-state incompressible Navier-Stokes equations.
- Comparison with MLSC [12].

Parametric solution

This test-case discusses the steady-state flow over a backward-facing step, which is a common fluid mechanics problem. The governing equations are the steady-state incompressible Navier-Stokes equations in dimensionless form:

$$\nabla \cdot \mathbf{u} = 0 \,, \tag{3.38a}$$

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}} \nabla^2 \mathbf{u} \,, \tag{3.38b}$$

where $\mathbf{u} = (u, v)$ is the velocity field, p the modified pressure, and Re the Reynolds number which is assumed to be uncertain, i.e., $\mathbf{z} = \mathrm{Re}$. The incompressible Navier-Stokes equations are hard to solve due to the non-linearity and the implicit coupling between mass conservation (3.38a) and momentum conservation (3.38b) by means of the pressure.

The set of steady state Navier-Stokes equations (3.38a)-(3.38b) are accompanied with a proper set of boundary conditions on a specified domain. In this section we consider the boundary conditions and domain that correspond to the backward-facing step problem, of which a schematic representation is shown in figure 3.17. The domain comprises a rectangle with length 15 and height

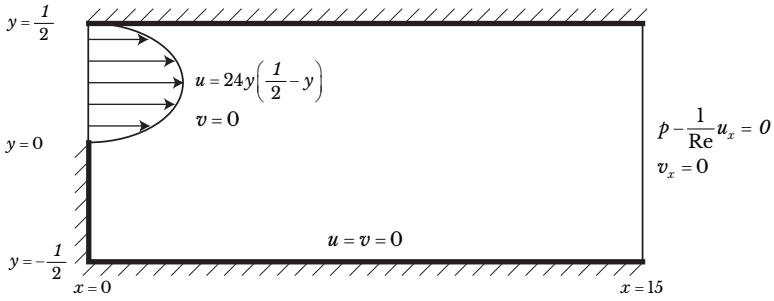


Figure 3.17: *Schematic of the backward-facing step problem.*

1 with solid boundaries, except for the upper part of the inlet ($x = 0$) and the full outlet ($x = 15$). A Poiseuille flow is imposed at the upper part of the inlet, while a pressure outlet condition is enforced at the outlet. The solver [35] is verified by comparing solutions for different mesh-sizes with a reference solution, which is believed to be an accurate benchmark for $\mathrm{Re} = 800$ [36]. The flow is highly dependent on the Reynolds number, and the solutions for $\mathrm{Re} = 100$ and 800 are shown in figure 3.18. The Reynolds number determines the size and location of the recirculating flow region right after the step and near the top boundary. The flow develops to a steady Poiseuille flow after a distance which is determined by the Reynolds number. However, in this case the flow is not yet fully developed and that is why we impose no Poisseuille boundary conditions at the outlet. The u-velocity profiles at $x = 7$ and $x = 15$ are shown in figure 3.19.

The solutions on the first level are computed on a 240×16 grid, and consist of the three quantities $(\mathbf{u}, \mathbf{v}, \mathbf{p})$. We choose to increase the fidelity by increasing the number of grid points with a

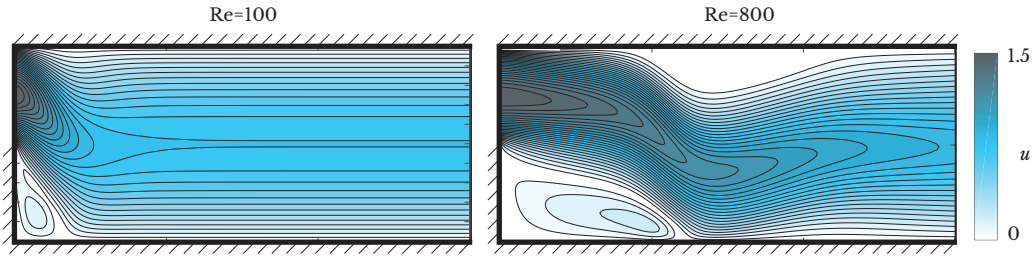


Figure 3.18: *Examples of flows over a backward facing step at different Reynolds numbers.*

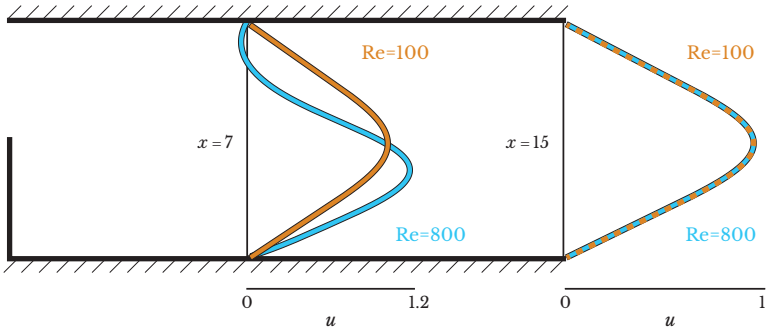


Figure 3.19: *The u-velocity profiles for backward facing step at $x = 7$ and $x = 15$ for two different Reynolds numbers.*

factor 2 for consecutive levels, i.e., $(N_x^{(i)}, N_y^{(i)}) = 2(N_x^{(i-1)}, N_y^{(i-1)})$. All three quantities $(\mathbf{u}, \mathbf{v}, \mathbf{p})$ are given as input to the neural network as a 2D 3-channel convolutional input. The tolerance for the training procedure is set to $\varepsilon = 10^{-6}$ and an accuracy tolerance $\varepsilon_{\text{acc}} = 10^{-4}$ is used. We start with 10 samples on the coarsest level (8 training samples, 2 validation samples), and apply the MLNN method from there on. The normed error difference for different numbers of levels, between our approximate solution and the solution computed on a fine 3840×256 grid (corresponding to level 5), is shown in figure 3.20. The number of samples required for approximating the mappings ac-

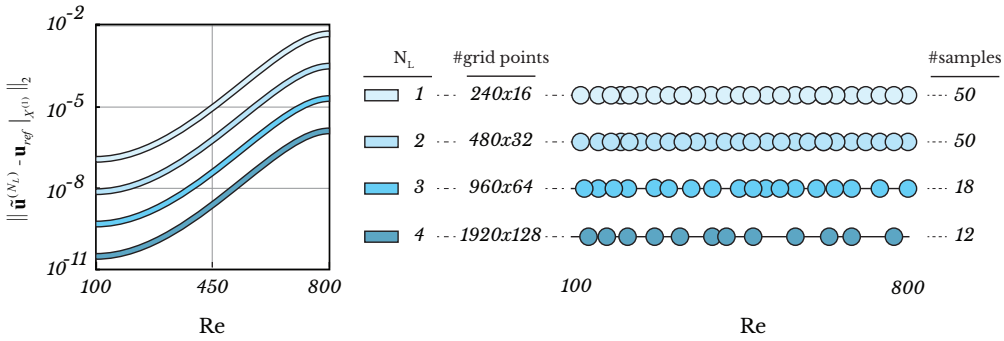


Figure 3.20: *Navier-Stokes equations. (left) Error convergence for different number of total levels. (right) Sample placement on each level.*

curately, increases when compared to the previous test-cases. This is due to the increase in degrees

of freedom in the neural networks, as the 2D multi-channel inputs require 2D convolutional layers, instead of the 1D convolutional layers that were used in the previous test-cases. Furthermore, the mappings are more complex when compared to the previous two test-cases.

Neural network mappings $P^{(i)}$

The neural network mappings $P^{(i)}$, $i = 2,3,4$ are shown in figure 3.21. It is striking that, even

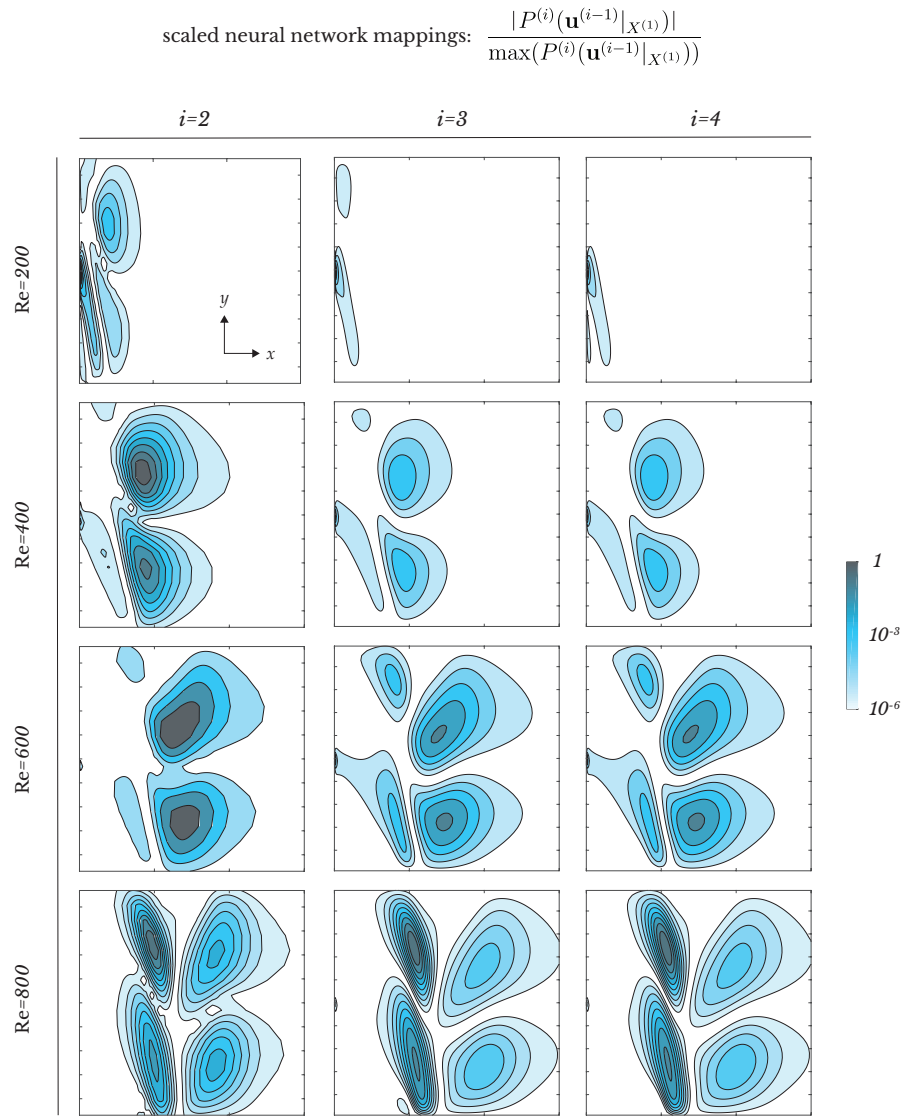


Figure 3.21: *Navier-Stokes equations. The neural networks mappings for the Navier-Stokes test-case in* (x, y, Re)*-space.*

for this complex non-linear test-case, the neural network approximations still show very similar

behaviour for consecutive levels. We stress again that this is the property that is utilised by our proposed transfer-learning approach to reduce the total number of samples required.

Comparison with MLSC

We compare the computational cost for MLNN and MLSC when constructing a surrogate with a certain accuracy. The computational cost is computed in the same way as described in section 6.I.4 and the results are shown in figure 3.22. Again, the MLNN method requires more computational

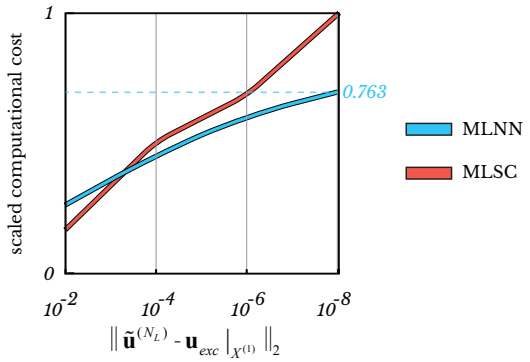


Figure 3.22: *Navier-Stokes equations. The scaled computational cost when constructing a surrogate for a range of accuracies.*

time for the approximation on the first level when compared to MLSC. However, for this non-linear test-case, the MLNN method outperforms the MLSC approach in terms of computational cost when constructing a surrogate of accuracy less than 10^{-4}.

6.4 Surrogate modelling for 3D fluid sloshing

As mentionded before, the focus of this part of the thesis is to enhance PIC/FLIP solutions. In this section we investigate the MLNN method to construct a surrogate for unsteady fluid sloshing in a 3D rectangular tank with 2 uncertainties. These surrogates are perfectly suited for accurate uncertainty propagation. The solver is discussed in more detail in section P of Part II.

Uncertainties

We prescribe a rotational tank motion by altering the gravity vector as is described in section P. 3. The gravity vector is defined as

$$\mathbf{g} = 9.81(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi)) \,, \tag{3.39}$$

where the two angles ψ and ϕ represent rotation in the x,y-plane and y,z-plane, respectively. These angles are assumed to change during the simulation as a function of time, given by

$$\psi(t) = \begin{cases} A\sin(2\lambda t), & t < \frac{2\pi}{\lambda} \,, \\ 0, & t \geq \frac{2\pi}{\lambda} \,, \end{cases} \quad \phi(t) = \begin{cases} A\sin(\lambda t), & t < \frac{2\pi}{\lambda} \,, \\ 0, & t \geq \frac{2\pi}{\lambda} \,, \end{cases} \tag{3.40}$$

where the amplitude A and period λ of the oscillation are assumed to be uncertain, i.e., $\mathbf{z} = (A, \lambda)$ and are assumed to be uniformly distributed on the intervals $I_A = [\frac{\pi}{8}, \frac{3\pi}{8}]$ and $I_\lambda = [\frac{1}{2}, \frac{3}{2}]$, respectively. This particular motion leads to heavy sloshing inside the tank and is suitable for testing the applicability of the MLNN method to a highly irregular fluid motion.

The QoI

When performing PIC/FLIP simulations, the shape of the fluid surface is our main interest, which follows directly from the particle positions. As a result, the QoI is defined as the number of particles contained within each grid cell at a given time level, which is chosen to be $t = 3\pi$. At this time instant, the tank rotation is back to its initial position, which allows us to compare the QoI.

Fidelities

As most of the computational expense in a PIC/FLIP simulation comes from the grid-based computations, we define the fidelity as the grid resolution. The time-step is tuned accordingly to satisfy the stability condition with safety factor 0.8 [1]. The solutions on the first level are computed on a fixed grid of $32 \times 16 \times 16$ cells. We choose to increase the fidelity by increasing the number of grid cells in every spatial direction with a factor 2 for consecutive levels, i.e., $(N_x^{(i)}, N_y^{(i)}, N_z^{(i)}) = 2(N_x^{(i-1)}, N_y^{(i-1)}, N_z^{(i-1)})$. Obtaining a fully converged solution for this test-case is difficult and we therefore consider as reference solution a solution that is computed on a fine $256 \times 128 \times 128$ grid, which corresponds to the 4th level. The QoI defined on the low-fidelity grid is given as input to the neural network as a 3D 1-channel convolutional input. An example of a low and a high-fidelity simulation result for $A = \frac{\pi}{4}$ and $\lambda = 1$ is shown in figure 3.23.

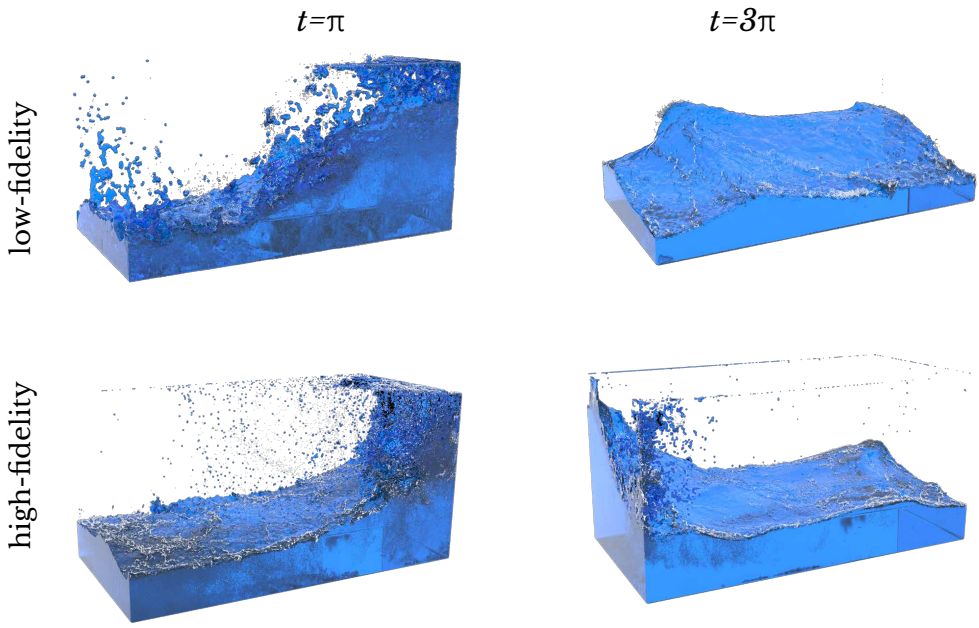


Figure 3.23: *3D Sloshing. Example simulation for $A = \frac{\pi}{4}$ and $\lambda = 1$. Isosurface generation is used to convert the particle positions to a fluid surface, which can then be rendered [37].*

Parametric solution

The number of particles in each of the low-fidelity grid cells, i.e., the QoI, is a function of the uncertain parameters A and λ. The MLNN method is used to construct a surrogate model of the

QoI in the random space spanned by the uncertain parameters. The goal is to approximate the solution that is computed on the high-fidelity $256 \times 128 \times 128$ reference grid, which corresponds to the 4th level in the MLNN approach. Minimising the samples on the 4th level required for accurate surrogate construction is paramount for the feasibility of the MLNN approach. As before, the tolerance for the training procedure is set to $\varepsilon = 10^{-6}$ and an accuracy tolerance $\varepsilon_{\text{acc}} = 10^{-4}$ is used. Errors are computed using a reference surrogate that is constructed using high-fidelity simulations (on level 4) with a 25×25 Gauss-Legendre grid [38] and the L_2-norm is taken over the solutions in the entire parameter space. The resulting error convergence for the QoI is shown in figure 3.24.

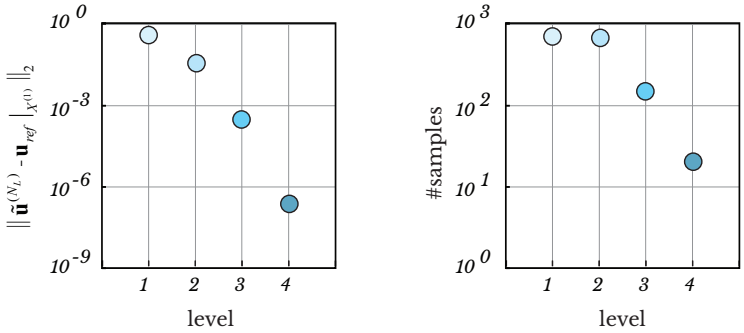


Figure 3.24: *3D Sloshing. Error convergence of the surrogate for the QoI constructed with the MLNN method.*

For these non-linear fluid motions, the surrogate construction is challenging, which can be noticed in the required number of samples on each level. The figure shows that constructing the surrogate requires significantly more samples on each level when compared to previous test-cases, which is caused by having two uncertainties and by the complexity of the test-case. However, there is still a significant decrease in the required number of samples when increasing the level, which shows the good applicability of our transfer learning approach also for this complex test-case.

Comparison with MLSC

The MLNN method is again compared with the MLSC method and the results are summarised in figure 3.25.
Our approach requires slightly more computational time for the approximation on the first level when compared to MLSC. On second and higher levels, our method shows a significant gain in computational efficiency when compared to the MLSC method, which is more pronounced than in previous test-cases. In this case, the MLNN method has a clear advantage when used for surrogate construction with medium to high level of accuracy.

7 Conclusion

In this chapter we have presented a novel method for surrogate construction. The method is based on a multi-level expansion of the solution and constructs approximations of the relative global discretisation error on different levels to enhance low-fidelity solutions. Inspired by the idea that these errors can be expressed in terms of the solution, the approximations of the error on each level are constructed using convolutional neural networks that apply a non-linear mapping of the

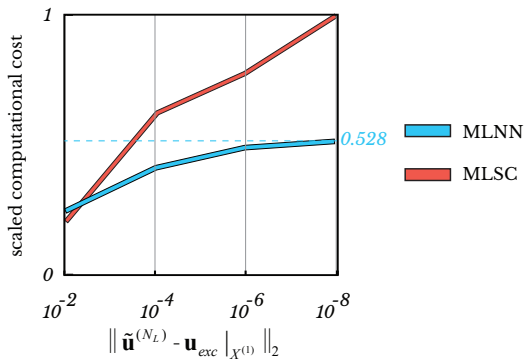


Figure 3.25: *3D Sloshing. Comparison with MLSC.*

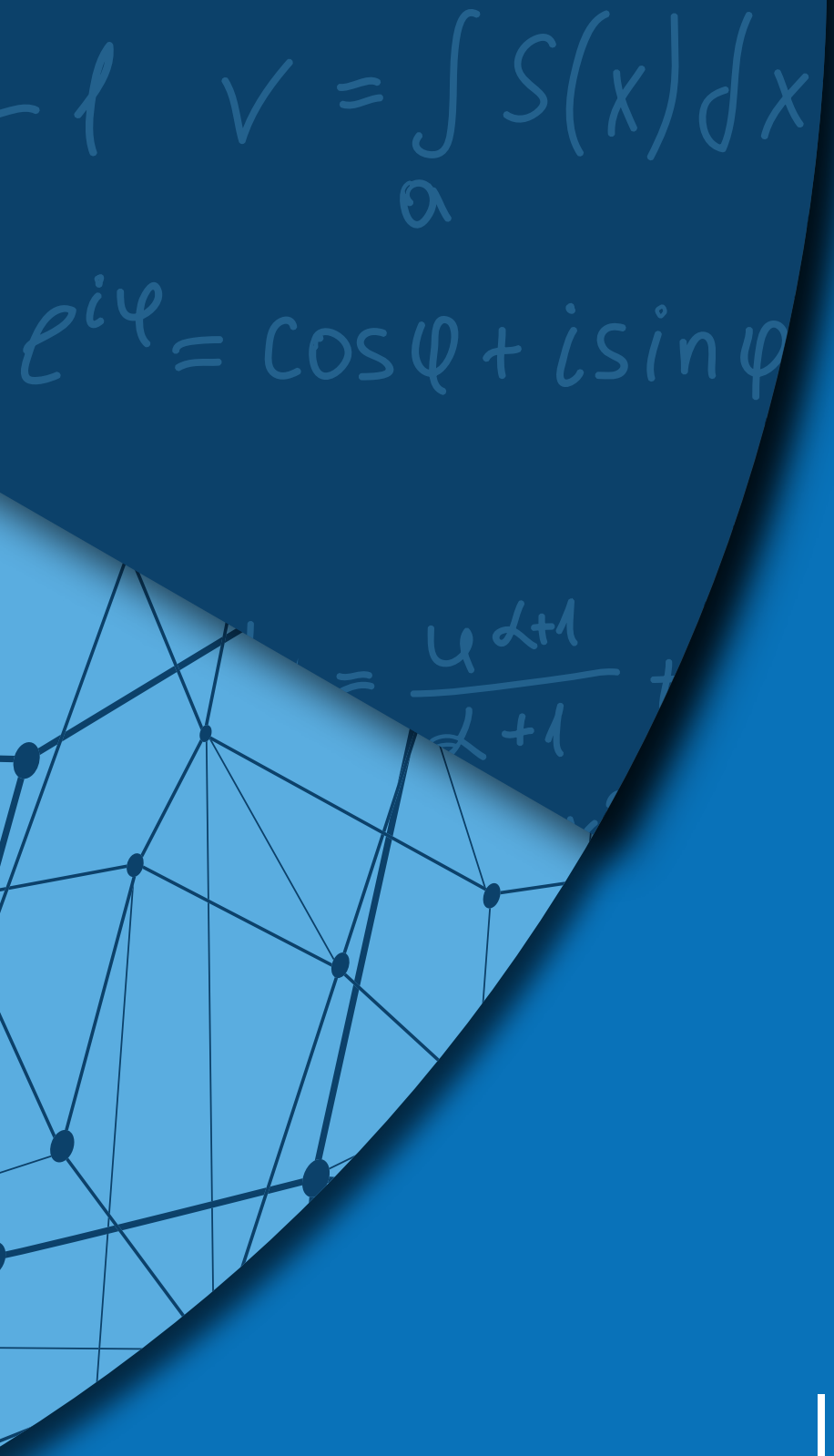
solution values to the difference between the solutions computed on the corresponding level and the subsequent level. Transfer learning reduces the amount of training samples that are required to properly train the neural networks.

The MLNN method has been employed for surrogate construction for several parametric partial differential equations: steady 1D advection diffusion equation, steady 1D Burgers equation, steady 2D incompressible Navier-Stokes equations, and unsteady 3D incompressible Navier-Stokes. We justified the use of transfer learning for these specific test-cases by studying the neural network mappings. We expect the applicability of transfer learning to generalise to other types of differential equations, provided that they possess similar smoothness properties, which is necessary for the error behaviour to be similar at different levels. In all test-cases, fast convergence is obtained, leading to an accurate surrogate model already at a relatively low number of model runs. We compared our approach to MLSC. The transfer learning reduces the number of samples on the higher levels, and therefore significantly decreases the computational cost when medium/high fidelity samples are expensive to sample or when a surrogate is wanted with a high accuracy, which effectively increases the number of required levels. The conclusion is that our method outperforms MLSC when either the PDEs are sufficiently complex, or when a highly accurate surrogate model is required. For example, for the complex test-case of a liquid sloshing in a tank, our approach leads to a computational cost saving of a factor 2 compared to MLSC, when requiring medium to high accuracy. The resulting surrogate model can be directly used as a computationally inexpensive tool for uncertainty quantification.

Furthermore, the method can be applied to unsteady problems, but is not optimal for it in its current form. Either the neural networks have to be retrained for different time instances, or the temporal component should be added as an extra dimension to the convolutional layers and the output of the neural networks.

What did we achieve?

This chapter discussed how to non-intrusively enhance the solutions coming from a low-fidelity solver. The non-intrusive approach is easy to implement as it does not require the underlying solver to be altered, but it does not allow for efficient solution enhancement involving a time dependency. How to effectively deal with this issue is discussed in chapter 4.



Intrusive solver enhancement

4 Intrusive solver enhancement

Multi-fidelity neural networks for real-time fluid flow predictions

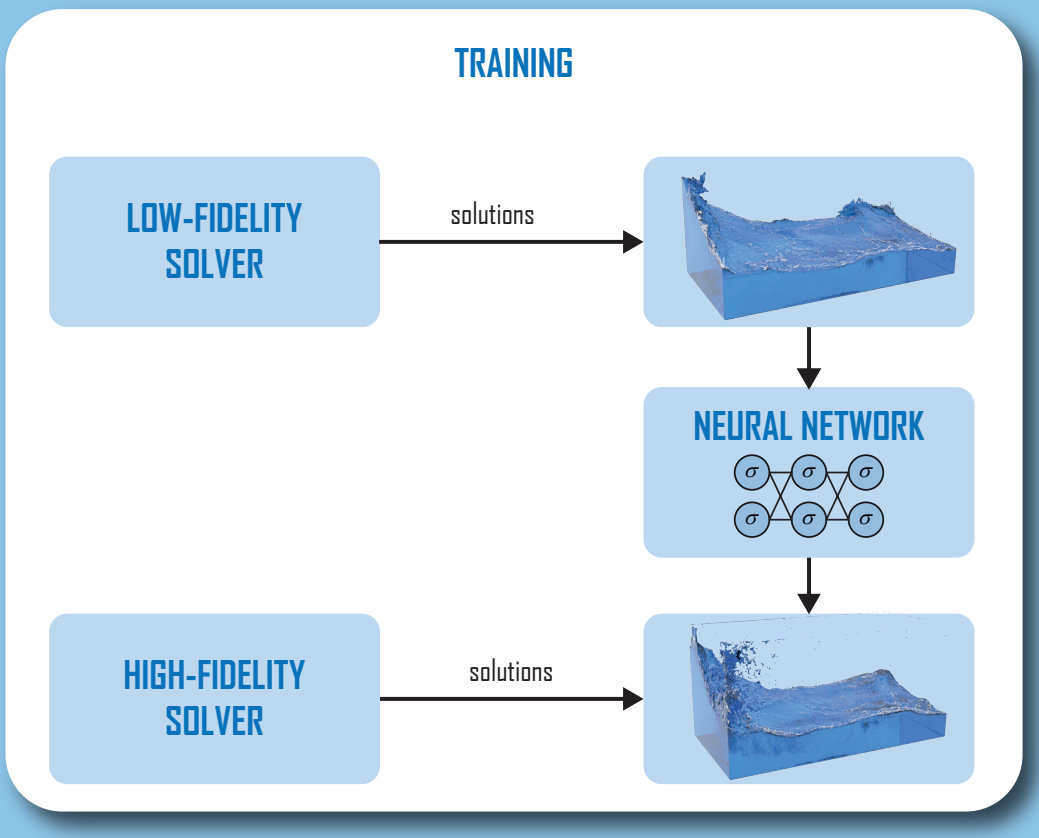
Traditional fluid flow predictions require large computational resources. Despite recent progress in parallel and GPU computing, the ability to run fluid flow predictions in real-time is often infeasible. Recently developed machine learning approaches, which are trained on high-fidelity data, perform unsatisfactorily outside the training set and remove the ability of utilising legacy codes after training. We propose a novel methodology that uses a deep learning approach that can be used within a low-fidelity fluid flow solver to significantly increase the accuracy of the low-fidelity simulations. The resulting solver enables accurate while reducing computational times up to 100 times. The deep neural network is trained on a combination of low- and high-fidelity data, and the resulting solver is referred to as a multi-fidelity solver. The proposed methodology is demonstrated by means of enhancing a fluid flow simulator, known as PIC/FLIP, which is a popular fluid flow simulator in the field of computer generated imagery.

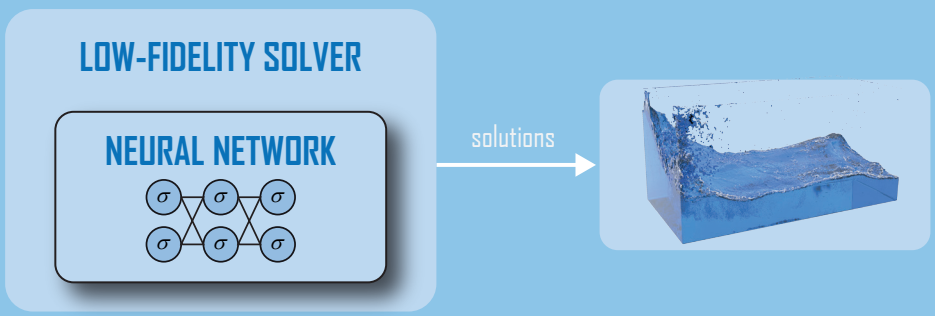
Background

As stated in the introduction, the goal of part II is to increase the accuracy of fast, but inaccurate solutions. The resulting accurate low-fidelity solver can then be used to efficiently sample a QoI, even in high-dimensional random spaces. This chapter discusses an intrusive way to enhance such a low-fidelity solver. In fluid dynamics, a distinction can be made between low-fidelity and high-fidelity simulations. Low-fidelity simulations are computationally cheap to perform, but have limited accuracy, as they do not capture all the underlying physics or do not resolve all scales. On the other hand, high-fidelity simulations incorporate all the relevant physical phenomena and corresponding scales, but may require a tremendous amount of computational resources. This makes it difficult to this date to perform accurate fluid simulations in real-time.

In order to address the problem of computational expense, various types of methods have been introduced. A commonly used method is Principal Component Analysis [39, 40], which provides the desired speed-up by transforming the dynamics of the fluid simulation to operations on linear combinations of snapshots, therefore restricting the richness of the dynamics. Recently developed data-driven methods use machine learning algorithms [29, 41, 42], such as regression forests and neural networks, that are trained on a large set of high-fidelity fluid flow simulation data. After training, these approaches are able to simulate fluid flows in real-time. However, they often perform unsatisfactorily outside the training set resulting in unrealistic fluid flow. This extrapolation problem is a weakness of almost all machine learning approaches but can be partially overcome by supplying the machine learning algorithm with more information about the underlying physics problem during training [41]. Furthermore, these recent deep learning methods have not yet been combined with existing fluid flow solvers, which have been developed and validated over many years.

Our Approach





The highlights of the new approach are:

- *Deconvolutional neural network for mapping quantities that are defined on a low-resolution grid to their high-resolution counterparts;*
- *Intrusive use of the neural network in a PIC/FLIP solver;*
- *Reasonable generalisation properties outside the training set.*

1 Our approach in short

We propose a novel method that enhances a low-fidelity fluid flow solver by training a deep neural network that maps values from a low-resolution computational grid to a high-resolution computational grid. This neural network is then used intrusively to enhance low-fidelity simulations at each time step. The input and output of the neural network are the current state of the low-fidelity simulation and the approximate high-fidelity current state, respectively. Compared to data-driven approaches that are purely trained on high-fidelity data, the crucial advantage of our model+data-driven approach is that the low-fidelity simulation is acting as a preconditioner, as it already comprises parts of the physics involved in the problem. Our proposed multi-fidelity approach is able to significantly enhance the accuracy of low-fidelity fluid flow predictions, even outside the training set.

2 Deep learning for enhancing low-fidelity fluid flow predictions

Our approach is to incorporate a deep neural network in the PIC/FLIP solver (discussed in chapter P of Part II, page 82) that effectively increases the resolution of the computational grid by mapping a coarse-grid velocity field to its corresponding fine-grid counterpart, reducing the dominant errors that are caused by computing the incompressible velocity field and grid to particle transfer on a coarse grid. The resulting PIC/FLIP solver will be subsequently called the multi-fidelity PIC/FLIP, as it utilises both low and high-fidelity data to train the deep neural network. A schematic overview of the steps in a low-fidelity and multi-fidelity PIC/FLIP solver is shown in figure 4.1.

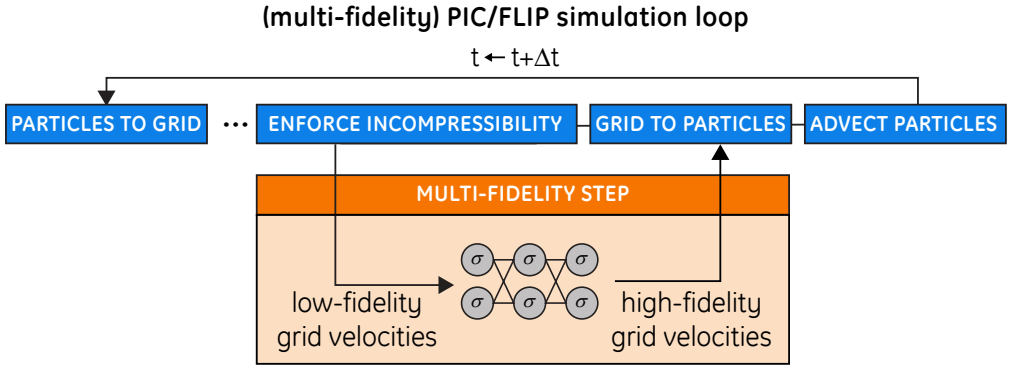


Figure 4.1: *Steps in the simulation loop for a single time step in the (multi-fidelity) PIC/FLIP solver.*

In order to train a neural network, we need to specify [24, 25]:

1. Why this approach?
2. Define low-fidelity and high-fidelity,
3. Input/output quantities for the neural network,
4. Training data,
5. Neural network architecture,
6. Training procedure.

The steps are discussed individually in the next subsections.

2.1 Why this approach?

The core idea is to learn low-level features of the low-fidelity simulations and map them to the corresponding fine-grid counterpart at every time step. Recent work shows that neural networks are perfectly suited for predicting fluid flows [43, 44, 45]. However, they are often used non-intrusively as a post-processing tool or as a standalone solver. Strictly enforcing physical laws inside the neural network is challenging and often not possible. Using neural networks intrusively in a solver allows the neural network to optimally utilise these physical laws provided by the solver optimally. Enhancing coarse-grid solutions by using low-level features is not new. It is done in for instance turbulence simulations using Large-Eddy Simulation (LES) [46], where sub-grid features on a grid are modelled using quantities on that very grid only.

2.2 Defining low and high-fidelity PIC/FLIP

In order to apply our method for enhancing a low-fidelity PIC/FLIP solver, we need to define low and high-fidelity. As the computational bottleneck occurs in the computations on the grid, both the low and high-fidelity computations use the same number of particles, but differ in grid resolution. To clarify, a high-fidelity simulation corresponds to simulations on a fine-resolution computational grid with resolution Δs_{HF} where we place $n_{p,\text{init}} = 8$ particles in a wet cell during the initialisation, while a low-fidelity simulation corresponds to computations on a coarse-resolution computational grid with resolution Δs_{LF} and where we place $8\left(\frac{\Delta s_{LF}}{\Delta s_{HF}}\right)^3$ particles in a wet cell during initialisation. Hence, the low-fidelity simulations have the same number of particles as the high-fidelity simulations and the combination of upscaling the resolution of the low-fidelity velocity grid and a large number of particles may result in an increase in accuracy. The grid resolutions Δs_{HF} and Δs_{LF} are specified in section 4. Lastly, the time step of the enhanced low-fidelity solver needs to be such that the simulation remains stable when upscaling the velocity field. Therefore, we take the low-fidelity time-step the same as the high-fidelity time-step, which latter is chosen such that the high-fidelity simulation is stable. Even though a stable high-fidelity time-step is often orders of magnitude smaller than its low-fidelity counterpart, an upscaled low-fidelity solution is still beneficial as most of the computations are performed on a coarse grid which significantly reduces the time it takes to perform a single time-step when compared to high-fidelity computations.

2.3 Input/output quantities for the neural network

Our approach uses a deep neural network to enhance low-fidelity simulations at each time step. The inputs for the neural network have to be quantities that can be computed directly from the low-fidelity simulations at the current and/or previous time levels.

The choice of input and output quantities is motivated by the fact that the main error is caused by performing velocity calculations on a coarse grid. The input for the neural network is only the current state of the simulation as seen by the computational grid:

- Scaled number of particles $P^s_{i,j,k}$ in each grid cell:

$$P^s_{i,j,k} := \min\left(1, \frac{P_{i,j,k}}{n_{p,\text{init}}\left(\frac{\Delta s_{LF}}{\Delta s_{HF}}\right)^3}\right), \tag{4.1}$$

  where $P_{i,j,k}$ is the number of particles in grid cell i, j, k.

- The face velocities defined on the low-fidelity grid $\mathbf{u}_{LF}$.

The scaled number of particles indicates if the cell is dry ($P^s_{i,j,k} = 0$), fully wet ($P^s_{i,j,k} = 1$), or partially wet ($0 < P^s_{i,j,k} < 1$). The $n_{p,\text{init}}$ in the definition of the scaled number of particles comes from the number of particles that are placed in a wet cell in the initialisation of a high-fidelity simulation (see section P. 1), and the ratio $\frac{\Delta s_{LF}}{\Delta s_{HF}}$ accounts for the difference in the number of particles placed in a wet cell in the initialisation. The scaled number of particles is also bounded to be within the interval $[0,1]$. One can use the positions of all particles directly as input for the neural network, but we opt for using the scaled number of particles per cell as input, which significantly reduces the dimensionality of the input when many particles are simulated, at a slight reduction in the information content.

The outputs of the neural network are the new face velocities $\mathbf{u}_{HF}$, which are defined on the high-fidelity grid, hence, effectively increasing the resolution of the grid. This new and improved velocity field is then fed back into the solver and used to calculate the new particle velocities in the ***grid to particles*** transfer at the current time level with a smaller interpolation error (P.17).

2.4 Training data

The neural network is trained on a set of data that comprises both low- and high-fidelity data. The low-fidelity data is used as input for the neural network, while the high-fidelity data acts as a reference in the cost function, which is discussed in section 2.6. The amount of data required in the training set may vary depending on the application and is often found heuristically [25]. Furthermore, a validation set is constructed, which is used to tune the neural network architecture and hyperparameters, and a test set is used to test how well the network generalises.

We focus on performing sloshing simulations [47]. Sloshing is the movement of a liquid contained inside a (possibly moving) object. These types of fluid flow simulations are relevant in many industry and computer generated imagery applications. For our application we construct the data set that is used for training, validation and testing, as is shown in figure 4.2.
The procedure in figure 4.2 performs a specified number of time steps and the motion of the rectangular tank is imposed by rotating the gravity vector in the simulation, i.e., $\psi(t=0) = \phi(t=0) = 0$ and $\mathbf{g}(t) = 10(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$, where the angles change randomly over time. After the data set has been constructed, we split the entire set in 70% training, 20% validation and 10% testing samples [25].

The number of time steps, angle increments $\Delta\psi$, $\Delta\phi$, and the filling height determine how well the neural network performs and this will be studied in detail in section 3.

2.5 Neural network architecture

Various options for the neural network type are available, e.g., fully-connected multilayer perceptrons (MLPs), convolutional neural networks (CNNs) or recurrent neural networks (RNNs), and determining which type to choose is often difficult. In general, the optimal neural network architecture depends on the application and a proper network type and architecture is often found by using expert knowledge.

Fully-connected, convolutional, recurrent, or ...?

In this study, there is a clear spatial component in both the input and the output of the neural network while there is no temporal component in the randomly picked training data. As a result, we opt for using a CNN architecture due to its inherent spatial nature. Furthermore, it reduces the total number of degrees of freedom significantly as compared to using an MLP architecture. The spatial nature of CNNs is caused by using local filters that take a weighted sum of neighbouring velocity values to compute a new velocity value. If we increase the number of layers, then the amount of neighbouring velocity values, that are used to calculate a new velocity value, increases. We do not know

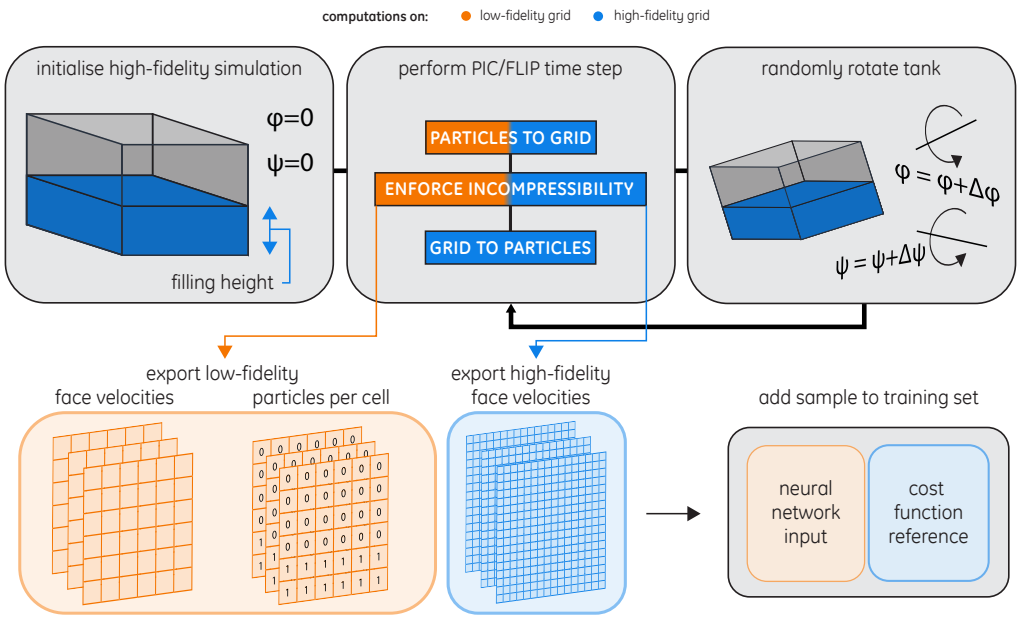


Figure 4.2: *Schematic of how the data set is constructed that is used for training, validation, and testing. The magnitudes of the changes in angles $\Delta\psi$ and $\Delta\phi$ depend on the dimensions of the tank.*

beforehand how many neighbouring velocity values should be used for accurately approximating the new velocity values and the number of CNN layers is therefore considered a hyperparameter in our approach.

How to increase the dimensions of the low-fidelity input to match the dimensions of the high-fidelity output?

When only using convolutional layers, the dimensionality of the input cannot be increased. This is problematic, because the goal of the neural network is to increase the effective grid resolution of the low-fidelity grid. To clarify, the low-fidelity input is of lower dimension than the high-fidelity output. This can be fixed by adding fully-connected layers after the convolutional part of the neural network. However these fully-connected layers significantly increase the number of degrees of freedom in the neural network and this leads to an increase of required number of training data. Constructing a larger training set requires more or longer high-fidelity simulations and is unwanted due to computational expense. The transposed convolutional layer, also known as deconvolutional layer [48, 49], solves this issue by performing a transposed matrix multiplication of the convolution matrix, which effectively increases the dimensions of the input and removes the need for adding fully-connected layers at the end of the network architecture. A schematic of the difference between a standard 2D convolutional layer and a 2D transposed convolutional layer is shown in figure 4.3.

The transpose convolutional layer is characterised by the number of filters, the filter size and the stride [25].

How to build the complete network architecture?

The complete neural network consists of a combination of both convolutional and transpose convolutional layers. The deconvolutional layers are used to increase the dimensionality of the neural network output, while the convolutional layers are used to effectively map the low-fidelity velocities

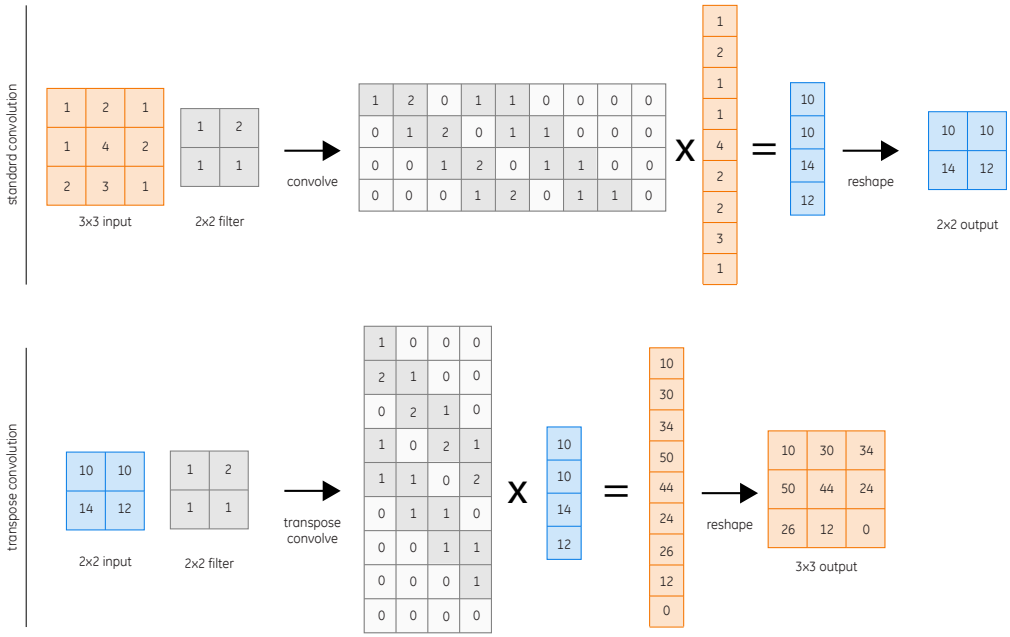


Figure 4.3: *A standard 2D convolutional layer and a 2D transposed convolutional layer of a single kernel for a 1-channel input.*

to the high-fidelity velocities. A schematic overview of the complete network architecture is shown in figure 4.4.
The architecture is defined as follows:

- The network starts with a number of 3D standard convolutional layers with a specific number of filters (discussed later).
- After the first layer the network consists of groups of a single 3D transpose convolution, and N_{CNN} 3D standard convolutional layers, where N_{CNN} is a hyperparameter.
  - The transpose convolutional layers increase the dimensions of their input with a factor 2 in every dimension, by using a 2×2-filter and a stride 2.
  - The standard convolutional layers use a 3×3-filter, which is a commonly used filter size in deep convolutional neural networks [25].
  - Zero padding is used at the standard convolutional layers to keep the input and the output dimensions of these layers unaltered.

The number of pairs is determined by the input (low-fidelity grid) and the output (high-fidelity grid) and the number of filters for each pair of deconvolution and convolutional layers decreases with a factor 4, ending with a single filter for the output layer representing the high-fidelity velocity field. For example, if the input corresponds to a low-fidelity grid of $16 \times 16 \times 16$ cells and the output corresponds to a high-fidelity grid of $256 \times 256 \times 256$ cells, 4 pairs of transpose and standard convolutional layers are needed to match the dimensions of the output, where the first convolutional layer uses 64 filters. To ensure that this makes sense, we require that the number of cells in each dimension in the high-fidelity grid is a multiple of 2 of the number of cells in the same dimension in the low-fidelity grid.

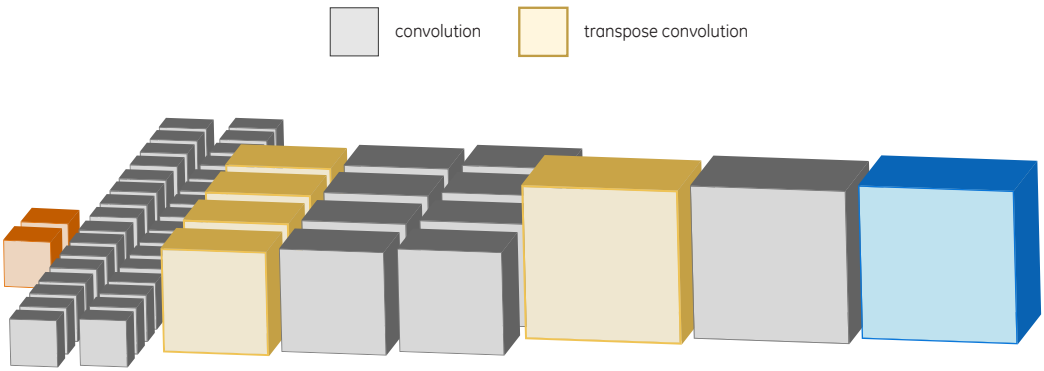


Figure 4.4: *The complete neural network architecture with hyperparameter $N_{CNN} = 2$. The architecture consists of a combination of convolutional and transpose convolutional layers.*

The network architecture is further characterised by the activation function used for each layer. In this chapter, we use a linear output activation to allow for unbounded values, and an exponential linear unit (ELU) [50] activation function for the remaining layers:

$$\sigma(z) = \begin{cases} z & , z > 0 \, , \\ \gamma(e^z - 1) & , z \leq 0 \, , \end{cases} \tag{4.2}$$

where γ is a hyperparameter. We prefer the ELU activation function over conventional activation functions, such as (leaky-)ReLU and hyperbolic tangent, as it can cope better with the dying neuron problem, is easy to evaluate, suffers less from the vanishing gradient problem when compared to hyperbolic tangent activation [51], and because there is evidence that it speeds up training when compared to ReLU [50].

2.6 Training the neural network

The cost function c is a function of the weights and biases and indirectly depends on the hyperparameters. To prevent overfitting, regularisation will be used [25].

The randomly generated training data in section 2.4 is used to train neural networks with different choices for the hyperparameters discussed in section 2.5. The cost function that is minimised is given by:

$$c_\lambda(W) = \sum_{i=1}^{N_t} \| (NN(\mathbf{u}_i^{LF}, \mathbf{p}_{\text{cell},i}) - \mathbf{u}_i^{HF} \|_2^2 + \lambda(\|W\|_2^2) \, , \tag{4.3}$$

where W are the trainable parameters (filter coefficients, biases) of the neural network NN, and where λ is the regularisation parameter, an additional hyperparameter for which a proper value needs to be determined during training. The cost function is minimised using the Adam optimizer with default parameter values $\beta_1 = 0.9$, $\beta_2 = 0.999$ and a step size α that is treated as a hyperparameter. Additionally, batch minimisation with batch size 256 is used to resolve out-of-memory errors, which may occur when passing a large dataset entirely for minimising the cost function.

We find proper values for the hyperparameters $(N_{CNN}, \gamma, \lambda, \alpha)$ by performing a grid search on the tensor grid H with 4 values for each hyperparameter. The 4 values for each hyperparameter are:

- ***Number of CNN layers after a deconvolution layer***: $N_{CNN} \in \{1,2,3,4\}$,
- ***ELU shape parameter***: $\gamma \in \{0.0001, 0.001, 0.01, 0.1\}$,

- ***Regularisation constant***: $\lambda \in \{10^{-8}, 10^{-6}, 10^{-4}, 10^{-2}\}$,
- ***Optimiser step size***: $\alpha \in \{0.0001, 0.001, 0.01, 0.1\}$.

As a result, we need to train $4^4 = 256$ neural networks, one for each different combination of the hyperparameters. From this set of trained neural networks, the optimal network is defined as the one that has the smallest validation error

$$\arg \min_{(N_{CNN},\gamma,\lambda,\alpha)\in H} \left[\sum_{i=1}^{N_v} \| (NN(\mathbf{u}_i^{LF}, \mathbf{p}_{\text{cell},i}) - \mathbf{u}_i^{HF} \|_2^2 \right] . \tag{4.4}$$

The test set can be used to determine if the found optimal neural network generalises well to unseen data. The resulting neural network is used to enhance the low-fidelity PIC/FLIP simulations.

3 Results

This section discusses the results of training and applying a multi-fidelity PIC/FLIP solver for fluid sloshing. The Tensorflow library for Python3 [52] is used for constructing and training the neural networks, and it runs on 4 NVIDIA GTX 1080Ti GPUs. The PIC/FLIP solver is also implemented to run on 4 NVIDIA GTX 1080Ti GPUs. The section is divided into four parts:

- Enhancing low-fidelity PIC/FLIP sloshing simulations.
- Generalisation capabilities for solver parameters.
- Generalisation to a 3D dambreak problem.
- A discussion of weaknesses of the multi-fidelity PIC/FLIP solver.

For all simulations we take a computational domain $(x,y,z) \in [0,10] \times [0,5] \times [0,5]$, unless stated otherwise. The high-fidelity PIC/FLIP solutions are computed with a cell width of $\Delta s = 1/50$ ($500 \times 250 \times 250$ cells) and simulate up to 250 million particles. The low-fidelity PIC/FLIP solutions are computed with $\Delta s = 1/10$ ($100 \times 50 \times 50$ cells) and simulate up to 2 million particles. The grid resolution of the low-fidelity simulations is chosen such that it is able to run in real-time, while the high-fidelity simulation grid contains 125 times more grid cells and can not be performed in real-time. The time step needs to be the same for both fidelities and is set to $\Delta t = 1/350$ which results in stable low- and high-fidelity simulations. Lastly, the picness f is set to 0.99 for both fidelities, which is a commonly used value in literature for simulating free surface water flow [2].

3.1 Enhancing low-fidelity PIC/FLIP sloshing simulations

We show that the proposed multi-fidelity approach can indeed enhance low-fidelity PIC/FLIP sloshing simulations.

The type of solutions that are enhanced

In order to study the effectiveness of a multi-fidelity PIC/FLIP solver, we prescribe a rotational tank motion by altering the gravity vector:

$$\mathbf{g} = 9.81(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi)) , \tag{4.5}$$

where ψ and ϕ are given by:

$$\psi(t) = \begin{cases} \sin(2t) & , t < 2\pi , \\ 0 & , t \geq 2\pi , \end{cases} \quad \phi(t) = \begin{cases} \sin(t) & , t < 2\pi . \\ 0 & , t \geq 2\pi . \end{cases} \tag{4.6}$$

This particular motion induces heavy liquid sloshing in the tank and is perfectly suited for testing our method. The filling height is varied to study generalisation capabilities of the method, and ranges between $(0,100)\%$.

Construction of the training set

The procedure for constructing the training data is shown in figure 4.2. A summary of all training parameters is given by:

- Data set: 10^6 generated training samples (see figure 4.2) with a filling height of 40%, $\Delta\psi, \Delta\phi$ are picked randomly every time step in the ranges $[-2\Delta t, 2\Delta t]$, $[-\Delta t, \Delta t]$, respectively.
  - 70% training,
  - 20% validation,
  - 10% testing,
- Training parameters: batch-size=256 , number of epochs=1000.

The training set only comprises samples with a single filling height of 40%, which corresponds to a filling height that allows for heavy sloshing motions and therefore a wide range of fluid configurations. This allows us to study if the neural network generalises well to cases with a different filling height. The range for $(\Delta\psi, \Delta\phi)$ corresponds to the same oscillation frequency as the one in (4.6). However, the probability that the tank motion (4.6) is comprised in the training data is negligible.

Training the neural networks

A total of 256 neural networks need to be trained (1 for each set of hyperparameters), and the resulting training and validation errors are shown in figure 4.5.

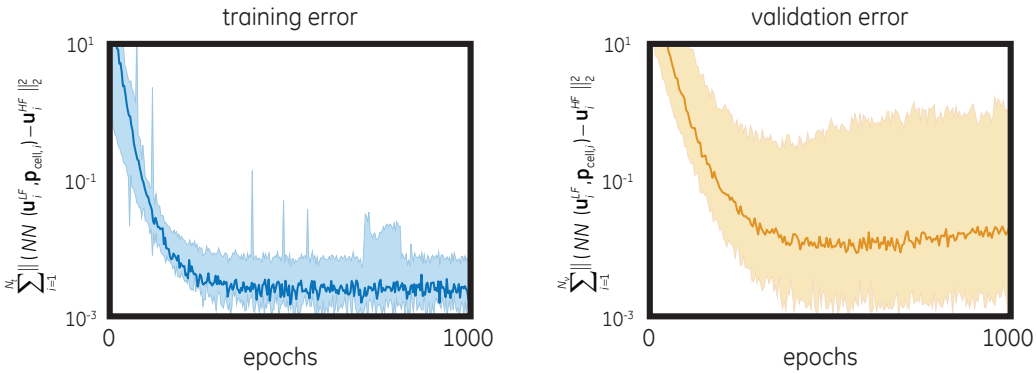


Figure 4.5: *The mean and envelope of both the training and validation error. The envelope shows the minimum and maximum values over all 256 trained neural networks.*

We clearly see that all 256 trained neural networks converge to a similar error in the training set, while the variation in the validation error is more pronounced. This indicates under/overfitting and a lack of generalisation capabilities for specific hyperparameter values. Remarkable is that the 103 networks with the largest validation errors are the networks where $N_{CNN} = 1,2$, which indicates that these relatively local multi-fidelity velocity approximations are not capable of generalisation outside the training set. This indicates that to approximate a local multi-fidelity velocity, we may need a large area of surrounding low-fidelity velocities. This global character of the required approximation may be caused by the incompressibility of the fluid, which leads to a coupling of velocities

by means of the pressure Poisson equation (P.12), which is also felt globally. The hyperparameters $(N_{CNN}, \gamma, \lambda, \alpha) = (3, 0.01, 10^{-4}, 0.001)$ satisfy (4.4), and the trained neural network with these hyperparameter values is used in the remainder of this chapter.

Enhancing solution with the same filling height as used during training

First, we start by using the trained neural network to enhance a sloshing simulation with the motion given by (4.6), given a filling height of 40%. Notice that this is the same filling height that is used for constructing the training set and is therefore considered a consistency test for the trained neural network. The low-fidelity, multi-fidelity, and high-fidelity PIC/FLIP simulations are compared at $t = \pi, 3\pi$. The rendered comparison is shown in figure 4.6.

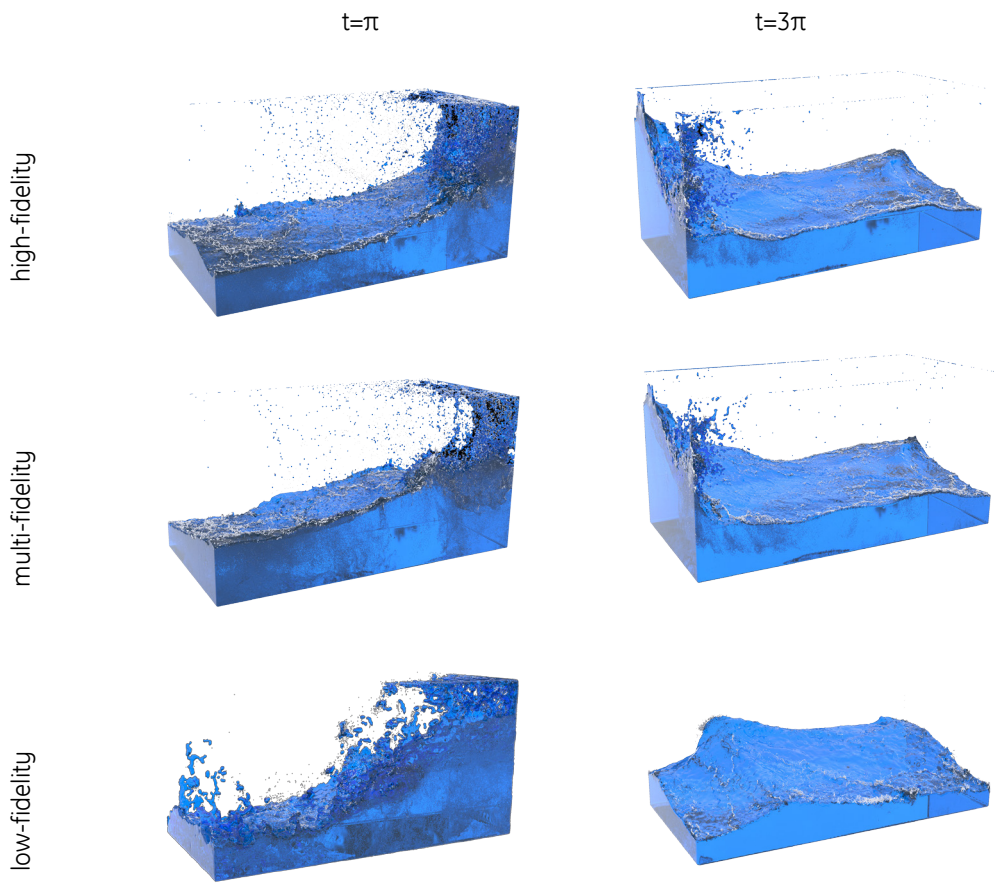


Figure 4.6: *Rendered fluid surface for the three different fidelities at $t = \pi, 3\pi$ with a filling height of 40%. The white particles are passive tracers that represent foam and spray, which are added in a post-processing stage based on physics phenomena described in [53]*

We clearly see a qualitative improvement in the shape of the fluid surface. Notice that the multi-fidelity solution at $t = \pi$ shows smaller scale droplets when compared to the low-fidelity solution, which is caused by effectively increasing the resolution of the grid. Furthermore, the multi-fidelity and the high-fidelity solutions at $t = 3\pi$ show a similar fluid surface, while the low-fidelity fluid surface has a completely different shape.

To study the difference between the three fidelities quantitatively, we perform a simulation until

$t = 4\pi$ for each of the three fidelities, and compare the so-called fluid-match. The fluid-match is defined as the percentage of the total volume of fluid that coincides with the fluid stemming from a high-fidelity solution. This is used as a measure to quantify how close a solution is to the reference high-fidelity solution. To clarify, if the fluid-match is 100%, then the fluid surface in the performed simulation is the same as the high-fidelity fluid surface. Notice that this does not necessarily mean that the neural network velocities are the same as the high-fidelity velocities. A quantitative comparison between the different fidelities is shown in figure 4.7.

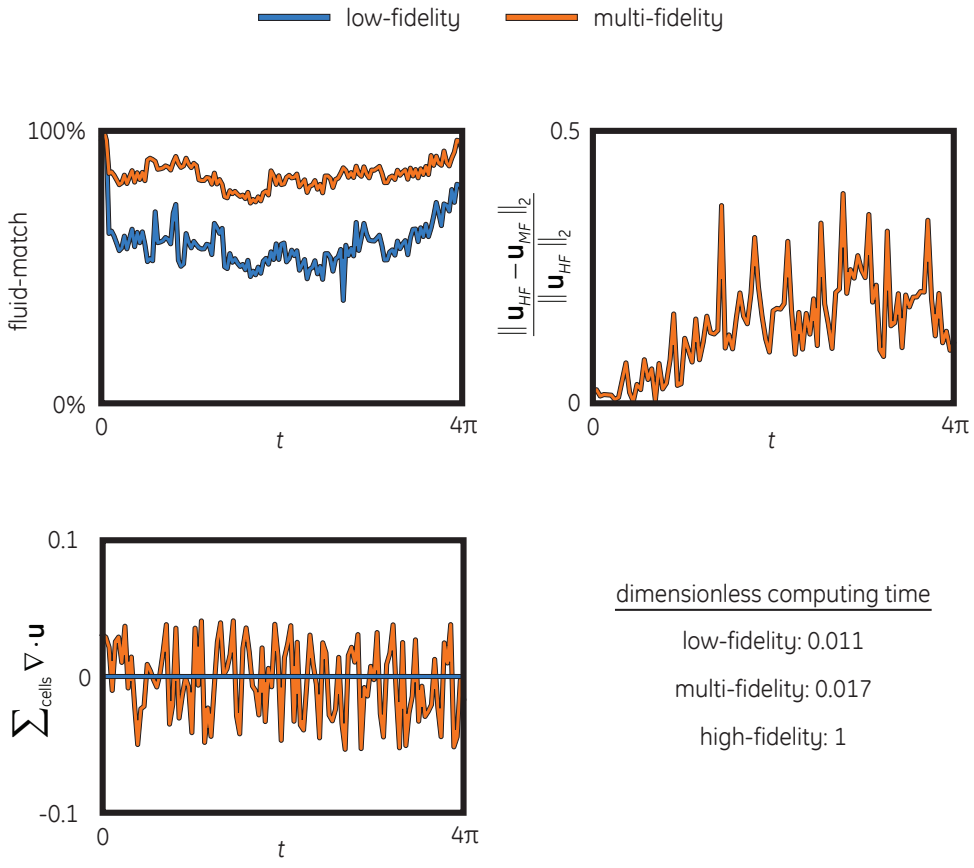


Figure 4.7: *Fluid-match, multi-fidelity (MF) incompressibility computed with* (P.9)*, and multi-fidelity relative velocity error*

A smooth initial part of the fluid-match resembles the period before the fluid impacts the boundaries of the domain. After the first impact, the fluid-match oscillates. When simulating for a longer period, the fluid-match converges back to almost 100%, due to the fluid coming to rest again. Notice that the fluid-match is not exactly 100% after the fluid being almost at rest, which is due to the fact that the PIC/FLIP solver does not preserve fluid volume in the presence of large particle velocities. These large velocities may result in particles being more densely clustered which causes the fluid volume to shrink. We clearly see a significant improvement in fluid-match when using the multi-fidelity PIC/FLIP solver, while not significantly increasing computational time. The neural network produces slightly compressible face-velocities, which may also cause the shrinking and growing of the total fluid volume. Lastly, the multi-fidelity and high-fidelity velocity fields show a gradual increase of mismatch in velocity values. The oscillating behaviour with large outliers of

the velocity mismatch is caused by the difference in droplet distributions, as shown in figure 4.6 at $t = \pi$. These droplets often have high velocities and a slight difference in droplet positions may cause a large mismatch in the face velocities. However, because these large outliers in velocity mismatch are caused by droplets, this large mismatch is not seen in the fluid-match, as they only affect the flow locally in a small part of the computational domain. *To summarise, the multi-fidelity solver significantly increases accuracy in terms of fluid-match for a sloshing simulation with the same filling height that was used during training, but generates slightly compressible velocity fields.*

Enhancing solutions with different filling heights

Enhancing solutions where the filling height differs from the filling height that was used during training is more difficult, as the trained neural network has never seen inputs that are associated with different filling heights. Figure 4.8 shows how well the neural network generalises for filling heights outside the training set.

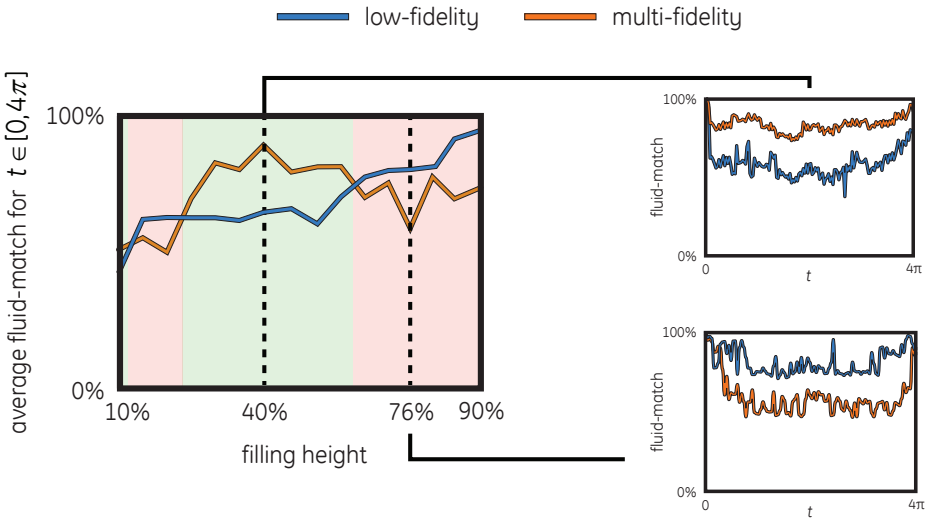


Figure 4.8: *Error in enhanced sloshing simulations for filling heights that were not used during training. The errors are computed over the interval $t \in [0,4\pi]$.*

The multi-fidelity solver outperforms the low-fidelity solver for a significant portion of filling heights (indicated by the green area in figure 4.8). The low-fidelity fluid-match shows improvement with increasing filling height, eventually leading to out-performance of the multi-fidelity solver. This increasing trend is caused by the fact that fluid sloshing does not affect the fluid-match significantly when the filling height reaches close to 100% (completely filled tank without fluid sloshing for which low-fidelity is already close to high-fidelity simulation). *To summarise, the multi-fidelity solver is able to the generalise to other filling heights, but leads to inaccurate results when deviating too much from the training set. Furthermore, without a high-fidelity reference, which is not available after training, it is difficult to predict the limits of the generalisation capabilities of the trained neural network. This is a common problem in machine learning and is an active field of research.*

3.2 Generalisation capabilities for solver parameter changes

The filling height is only one out of several parameters that were used in the sloshing test-case. Studying how the multi-fidelity solver generalises when changing solver parameters provides information on how sensitive the neural network is to the training data. In this section we show how the multi-fidelity solver generalises when changing the following parameters:

- ***Time step***: $\Delta t \in [\frac{1}{275}, \frac{1}{500}]$.
  - The time step is picked such that the high-fidelity simulations are stable, which makes it easy to compare all three fidelities.
- ***Computational domain size scaling***: $c_{\Delta s} \in [\frac{1}{2}, \frac{3}{2}]$.
  - While keeping the number of cells in the low- and high-fidelity simulations the same, we can change $\Delta s \rightarrow c_{\Delta s}\Delta s$ to effectively increase the size of the computational domain.
- ***Number of particles in wet cell during initialisation***: $n_{p,\text{init}} \in [4, 12]$.
  - The wet cells in the multi-fidelity simulation are initialised with $n_{p,\text{init}} \left(\frac{\Delta s_{LF}}{\Delta s_{HF}}\right)^3$ particles ($n_{p,\text{init}} = 8$ was used for training).
- ***Picness parameter***: $f \in [0, 1]$.
  - Determines how diffusive the fluid motions are. Changing this value significantly changes the behaviour of the fluid ($f = 0.99$ was used for training).
- ***Gravity vector scaling***: $c_{\mathbf{g}} \in [5, 15]$.
  - The gravity vector will be defined as $\mathbf{g} = c_{\mathbf{g}}(\sin(\psi)\cos(\phi), \sin(\phi), -\cos(\psi)\cos(\phi))$ ($c_{\mathbf{g}} = 9.81$ was used for training). The value for $c_{\mathbf{g}}$ determines the strength of the gravitational force and changing this value significantly changes fluid behaviour.

The fluid-matches for the low- and multi-fidelity simulations are compared for the sloshing simulation characterised by the sloshing motion (4.6) for $t \in [0, 4\pi]$. The results are shown in figure 4.9. We clearly see that the multi-fidelity solver generalises well for all solver parameters apart from the picness parameter f and gravity scaling constant $c_{\mathbf{g}}$. The parameters Δt, Δs and $n_{p,\text{init}}$ do not significantly change the fluid motions and the neural network will receive inputs that are similar to the training data. As a result, the multi-fidelity solver attains a significant increase in accuracy when compared to the low-fidelity solver. The parameters f and $c_{\mathbf{g}}$ do significantly change the fluid behaviour though, which results in neural network inputs that significantly deviate from the training data. *To summarise, for a large portion of the parameter value ranges the multi-fidelity solutions show an increase in accuracy when compared to the low-fidelity solutions. However, it is not possible to generalise to cases where c_g and f deviate significantly from the training data.*

3.3 Generalisation to a 3D dambreak problem

It was shown in sections 3.1 that the trained neural network, that was used in the multi-fidelity solver, was able to generalise to different filling heights and solver parameters values. However, the simulations were still similar in the sense that the prescribed motion (4.6) was the same and the initial fluid configuration was similar. In this section we show that the trained neural network is able to generalise to a different test-case as well.

We consider a wet dambreak problem of which the initial fluid configuration is shown in figure 4.10.

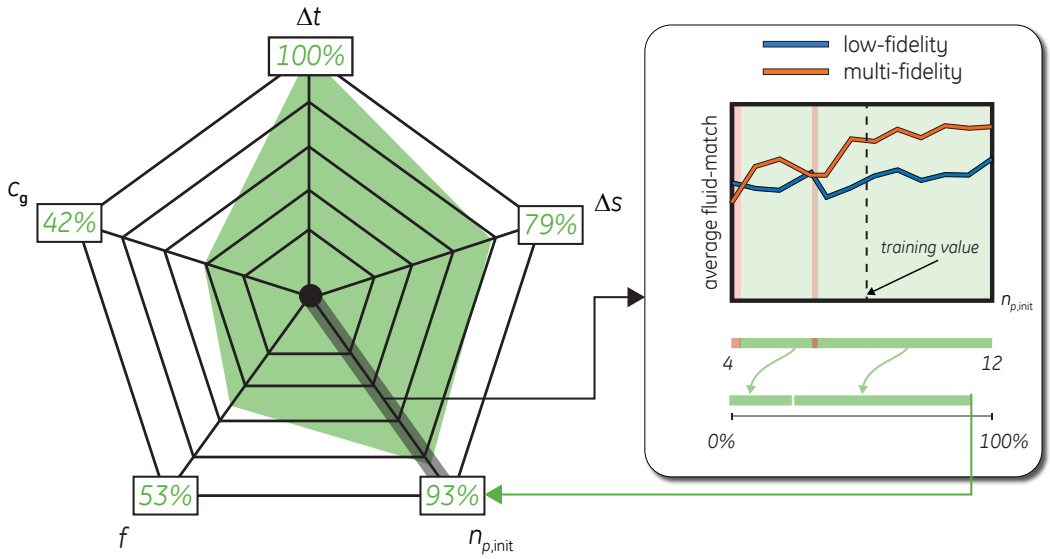


Figure 4.9: *Generalisation capabilities for different solver parameters. The green regions indicate the fraction of parameter values for which the multi-fidelity fluid-match is higher than the low-fidelity fluid-match for the sloshing simulation for* $t \in [0, 4\pi]$.

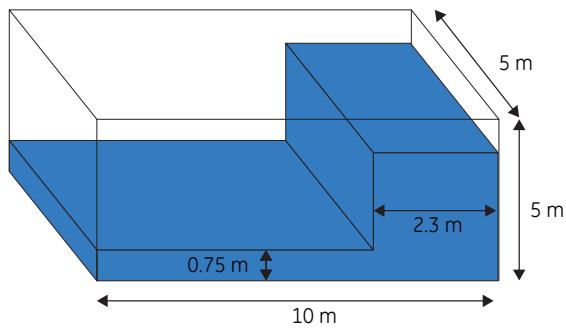


Figure 4.10: *The initial fluid configuration of the wet dambreak problem.*

This particular configuration is chosen such that the total volume corresponds to a 30% filling height. This allows us to study if the neural network generalises to a case with a different filling height than the one that was used for training, and was also initialised in a different configuration. This dambreak problem leads to a breaking wave that impacts the left domain boundary. The low and multi-fidelity solutions are compared for $t \in [0, 15]$, i.e., the period before, during, and after the wave impact. A qualitative and quantitative comparison of the three fidelities is shown in figure 4.11.

Both the low- and multi-fidelity simulations show good agreement with the high-fidelity solution. The effectiveness of the neural network approach is again investigated through the fluid-match, which shows an increase of accuracy for the multi-fidelity solution, even for this dambreak problem for which the training set was not tailored. However, the multi-fidelity solution shows tiny oscillations on the fluid surface, which is caused by oscillatory velocity predictions. The frequency of the oscillation corresponds to the largest frequency that can be resolved on the high-fidelity com-

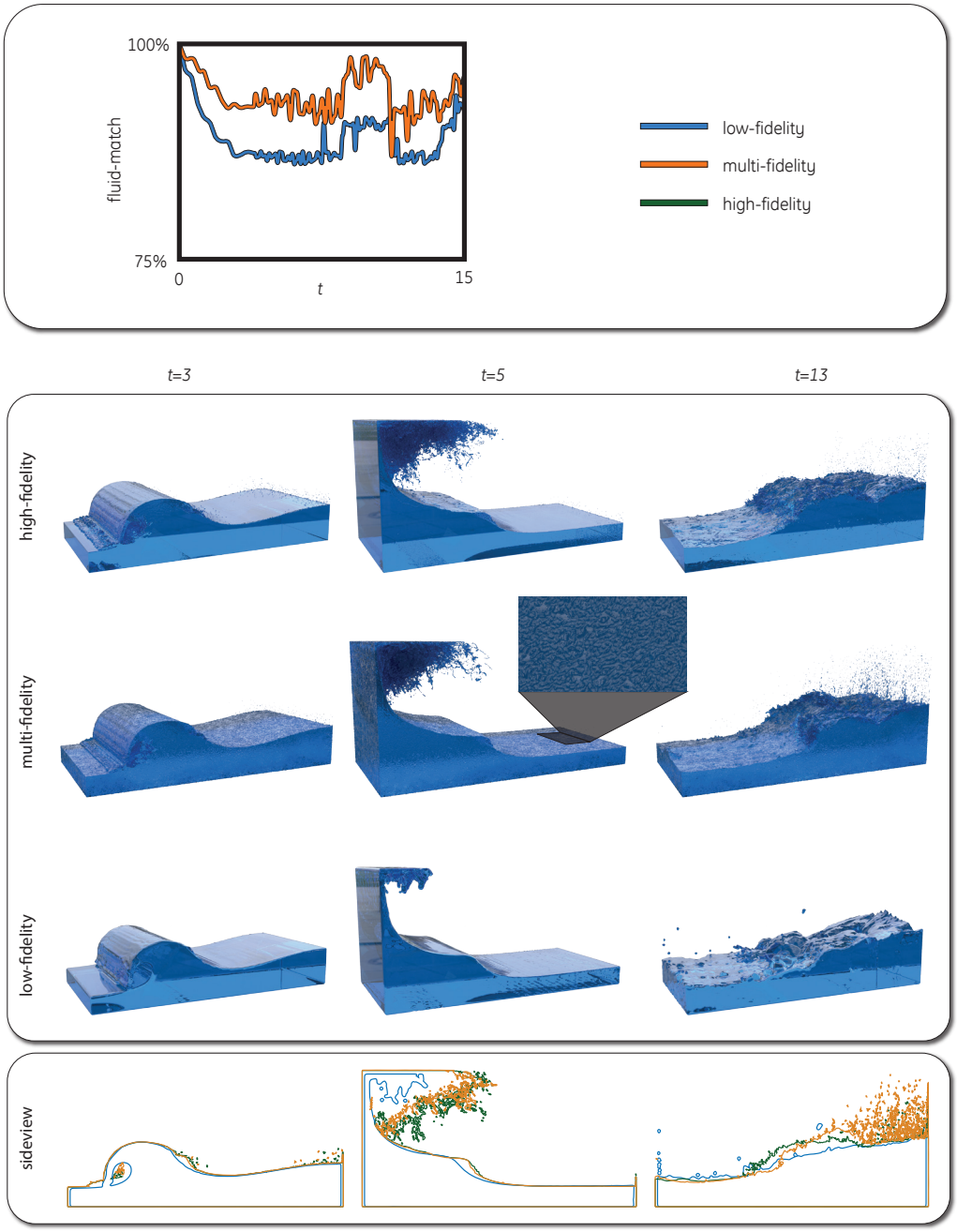


Figure 4.11: *Low-, multi-, and high-fidelity solutions for the dambreak problem.*

putational grid. The oscillations immediately appear in the first few time steps of the multi-fidelity simulation and grow slightly until the wave impacts the domain boundary. This phenomenon is most likely caused by the configuration of the initial condition which is not comprised in the training set.

Even though these oscillations are unwanted, they do not affect the accuracy of the multi-fidelity solution severely. In order to possibly remove the unwanted oscillations, one may enlarge the training set to encapsulate more test-cases. *To summarise, these results indicate that the generalisation to other test-cases is possible, but a carefully constructed training set may be needed to remove unwanted artefacts in the enhanced velocity field.*

3.4 Weaknesses of the multi-fidelity PIC/FLIP solver

It was shown that our approach is able to generalise to solver parameters and test-cases slightly outside the training set. However, the problem of generalisation to cases further away from the training data still exists. In this section we give an overview of the problems we encountered during training of the neural network and of the limitations of the proposed approach. In short the limitations of our multi-fidelity approach are:

- Sensitivity to solver parameters.
- Does not generalise to other test-cases.
- Difficult to obtain physical interpretation of the multi-fidelity approach.

Sensitivity to solver parameters

As is shown in figure 4.9, the neural network is able to simulate cases with slightly different solver parameters. It is possible to construct a training set that comprises simulations with multiple solver parameters, which increase the range of solver parameters for which the multi-fidelity solver produces satisfactory results. However, as the training set is constructed with low- and high-fidelity data, we need to make sure that the high-fidelity simulations are stable, which results in a small time step. We chose to use this small time step for the low-fidelity simulations, which results in a multi-fidelity solver that also uses a similar small time step. An alternative is to construct the training set with low-fidelity simulations, computed with a large time step, and a high-fidelity counterpart, computed with a small time step. The resulting training set then comprises all low-fidelity data and the sub-sampled high-fidelity simulation data at the same time levels as the low-fidelity simulation. This approach leads to unstable multi-fidelity solutions, which indicates that the multi-fidelity solver needs to satisfy similar stability conditions as the high-fidelity solver.

Generalisation to other test-cases

We showed that the neural network, which was trained on sloshing simulation data, is able to generalise to a wet dambreak problem. This is a promising result, but using the multi-fidelity solver for still more differing cases led to unsatisfactory results. In figure 4.12 we show three example high-fidelity simulations where the multi-fidelity solver became unstable after only a few time steps. These cases all have features that the neural network has never encountered before, i.e., a dry bed, static obstacles in the flow, moving obstacles moving through the fluid. This is consistent with the results in [41, 29] where it is noted that generalisation is often an issue when using neural networks for fluid flow predictions. A carefully constructed training set may be needed to be able to generalise to a wide range of test-cases.

Physical interpretation of the multi-fidelity approach

The idea of our approach is to learn low-level features of the low-fidelity simulations and map these to the corresponding fine-grid counterpart. An in-depth analysis of why this approach works when using neural networks is challenging, due to the inherent non-linearities of the neural networks and the underlying physics.

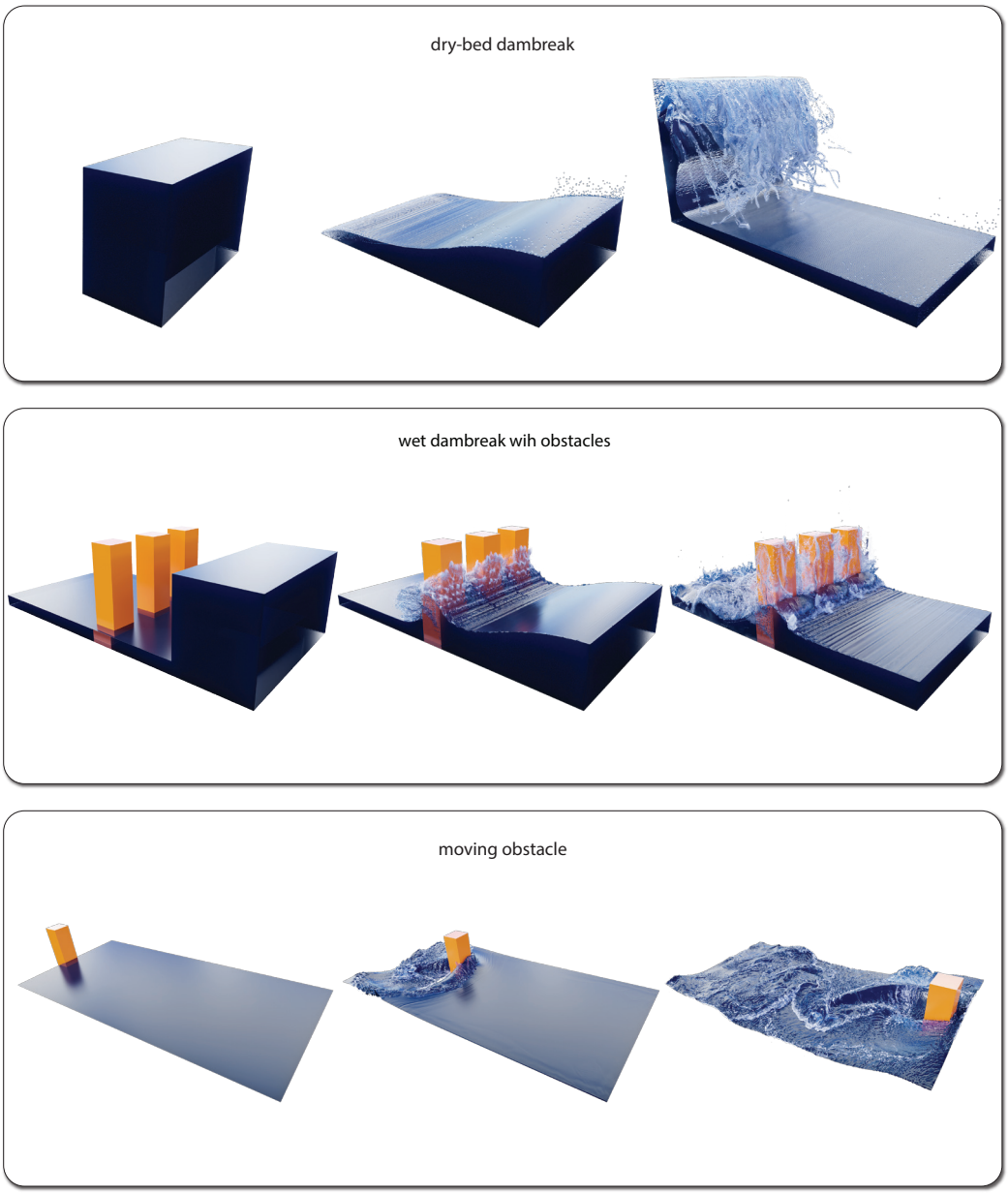


Figure 4.12: *Three example high-fidelity simulations where the multi-fidelity solver fails. The dry-bed dambreak consists of a column of fluid that is released at $t = 0$, resulting in a wave impact on the domain boundary. The dambreak with obstacle uses the same initial fluid configuration as shown in figure 4.10, but has solid obstacles (orange boxes) in the middle of the domain. The moving obstacle case consists of an obstacle (orange box) that is moving in a sinusoidal motion through a tank that is 40% filled with fluid.*

4 Conclusion

We investigated the use of deep deconvolutional neural networks to enhance a low-fidelity PIC/FLIP fluid solver. An important novelty of our approach is that it is used intrusively in the low-fidelity

solver and is able to enhance the solution at every time step. This is different from the common non-intrusive approaches that use neural networks either as a standalone solver or as a post-processing tool. The neural network in this chapter was trained on randomly generated fluid sloshing data. After training, the neural network provided a significant increase in accuracy when compared to the low-fidelity solution, for the case of a sloshing simulation that was not comprised in the training data. Even though the neural network introduced small oscillations on the fluid surface when enhancing a dambreak problem solution, it was still able to significantly enhance accuracy for this problem, which hints at the generality of our approach.

Our approach still has some issues. Firstly, the choice for the time step is rather restrictive; it needs to be kept small to keep the multi-fidelity solution stable. Secondly, our approach fails to enhance solutions that introduce new phenomena that were not encountered in the training set, e.g., obstacles in the flow. The neural network velocity predictions became unstable immediately, leading to non-physical large velocity values, but we expect that a carefully constructed training set may alleviate this issue. Lastly, a common issue when using neural networks is that an in-depth analysis of why they work is challenging, due to the inherent non-linearities of the neural networks and the underlying physics.

What did we achieve?

This chapter discussed how to intrusively enhance accuracy of a low-fidelity solver. The enhanced solutions lead to a more accurate prediction of the QoI and can be used to sample the random space efficiently without a heavy computational burden. The difference with the non-intrusive approach from chapter 3 is that the intrusive approach allows for efficient treatment of a time dependency, hence allowing to march accurate low-fidelity solutions in time. This comes at a cost, i.e., the underlying low-fidelity solver needs to be altered by adding an extra neural network mapping step, which was not the case in the non-intrusive approach.

References

[1] R. Bridson, Fluid Simulation for Computer Graphics, CRC Press, 2015.

[2] Y. Zhu, R. Bridson, Animating sand as a fluid, in: ACM SIGGRAPH 2005, SIGGRAPH '05, ACM, New York, USA, 2005, pp. 965–972.

[3] G. M. Phillips, Interpolation and Approximation by Polynomials, Springer-Verlag, 2003.

[4] B. Peherstorfer, K. Willcox, M. Gunzburger, Survey of multifidelity methods in uncertainty propagation, inference, and optimization, SIAM Review 60 (3) (2018) 550–591.

[5] X. Zhu, E. M. Linebarger, D. Xiu, Multi-fidelity stochastic collocation method for computation of statistical moments, Journal of Computational Physics 341 (2017) 386–396.

[6] A. Forrester, A. Sobester, A. Keane, Multi-fidelity optimization via surrogate modelling, Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 463 (2088) (2007) 3251–3269.

[7] A.-L. Haji-Ali, F. Nobile, L. Tamellini, R. Tempone, Multi-index stochastic collocation for random PDEs, Computer Methods in Applied Mechanics and Engineering 306 (2016) 95–122.

[8] A. Narayan, C. Gittelson, D. Xiu, A stochastic collocation algorithm with multifidelity models, SIAM Journal on Scientific Computing 36 (2) (2014) A495–A521.

[9] X. Zhu, A. Narayan, D. Xiu, Computational aspects of stochastic collocation with multifidelity models, SIAM/ASA Journal on Uncertainty Quantification 2 (1) (2014) 444–463.

[10] K. Cliffe, M. Giles, R. Scheichl, A. Teckentrup, Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients, Computing and Visualization in Science 14 (3) (2010) 444–463.

[11] A. Teckentrup, Multilevel Monte Carlo Methods and Uncertainty Quantification, PhD Thesis at University of Bath, 2013.
URL `https://books.google.nl/books?id=RunSnQEACAAJ`

[12] A. Teckentrup, P. Jantsch, C. Webster, M. Gunzburger, A multilevel stochastic collocation method for partial differential equations with random input data, SIAM/ASA Journal on Uncertainty Quantification 3 (1) (2015) 1046–1074.

[13] D. Kouri, A multilevel stochastic collocation algorithm for optimization of PDEs with uncertain coefficients, SIAM/ASA Journal on Uncertainty Quantification 2 (1) (2014) 55–81.

[14] J. Charrier, R. Scheichl, A. Teckentrup, Finite Element Error Analysis of Elliptic PDEs with Random Coefficients and Its Application to Multilevel Monte Carlo Methods, SIAM Journal on Numerical Analysis 51 (1) (2013) 322–352.

[15] H. Harbrecht, M. Peters, M. Siebenmorgen, On Multilevel Quadrature for Elliptic Stochastic Partial Differential Equations, Lecture Notes in Computational Science and Engineering 88 (2013) 161–179.

[16] I.-G. Farcaş, P. C. Sârbu, H.-J. Bungartz, T. Neckel, B. Uekermann, Multilevel adaptive stochastic collocation with dimensionality reduction, in: J. Garcke, D. Pflüger, C. G. Webster, G. Zhang (Eds.), Sparse Grids and Applications - Miami 2016, Springer International Publishing, 2018, pp. 43–68.

[17] J. Lang, R. Scheichl, D. Silvester, A fully adaptive multilevel stochastic collocation strategy for solving elliptic PDEs with random data (2019). arXiv:1902.03409.

[18] M. B. Giles, Multilevel Monte Carlo path simulation, Operations Research 56 (3) (2008) 607–617. arXiv:https://doi.org/10.1287/opre.1070.0496.

[19] M. B. Giles, Multilevel Monte Carlo methods, Acta Numerica 24 (2015) 259–328.

[20] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (2) (1991) 251 – 257.

[21] G. Lewicki, G. Marino, Approximation by superpositions of a sigmoidal function, Applied Mathematics Letters 17 (2004) 1147–1152.

[22] B. Hanin, M. Sellke, Approximating continuous functions by ReLU nets of minimal width (2017). arXiv:1710.11278.

[23] G. Marchuk, V. Shaidurov, Difference Methods and Their Extrapolations, Stochastic Modelling and Applied Probability, Springer, 1983.

[24] C. C. Aggarwal, Neural Networks and Deep Learning: A Textbook, Springer, 2018.

[25] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.

[26] P. Robbe, D. Nuyens, S. Vandewalle, Recycling samples in the multigrid multilevel (quasi-)Monte Carlo method, SIAM Journal on Scientific Computing 41 (5) (2019) S37–S60.

[27] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations (Nov. 2017). arXiv:1711.10561.

[28] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler, Deep Fluids: a generative network for parameterized fluid simulations (Jun. 2018). arXiv:1806.02071.

[29] L. Ladický, S. Jeong, N. Bartolovic, M. Pollefeys, M. Gross, Physicsforests: real-time fluid simulation using machine learning, in: ACM SIGGRAPH 2017, 2017, pp. 22–22.

[30] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[31] J. Yosinski, J. Clune, Y. Bengio, H. Lipson, How transferable are features in deep neural networks?, in: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2, NIPS'14, MIT Press, Cambridge, MA, USA, 2014, pp. 3320–3328.

[32] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, TensorFlow: A system for large-scale machine learning, in: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[33] E. Bertolazzi, G. Manzini, A second-order maximum principle preserving finite volume method for steady convection-diffusion problems, SIAM Journal on Numerical Analysis 43 (2005) 2172–2199.

[34] H. Dret, B. Lucquin, The Variational Formulation of Elliptic PDEs, Partial Differential Equations: Modeling, Analysis and Numerical Approximation, Springer International Publishing, 2016, pp. 117–143.

[35] B. Sanderse, ECNS: Energy-Conserving Navier-Stokes solver verification of steady laminar flows, ECN, 2011.

[36] D. K. Gartling, A test problem for outflow boundary conditions—flow over a backward-facing step, International Journal for Numerical Methods in Fluids 11 (7) (1990) 953–967. arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.1650110704.

[37] A. J. C. Crespo, J. M. Dominguez, B. D. Rogers, M. Gomez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. Garcia-Feal, DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH), Computer Physics Communications 187 (Supplement C) (2015) 204–216.

[38] D. Xiu, Numerical Methods for Stochastic Computations: A Spectral Method Approach, Princeton University Press, 2010.

[39] S. S. Ravindran, Reduced-order adaptive controllers for fluid flows using POD, Journal of Scientific Computing 15 (4) (2000) 457–478.

[40] C. Audouze, F. D. Vuyst, P. B. Nair, Reduced-order modeling of parameterized PDEs using time–space-parameter principal component analysis, International Journal for Numerical Methods in Engineering 80 (8) (2009) 1025–1057.

[41] L. Ladický, S. Jeong, B. Solenthaler, M. Pollefeys, M. Gross, Data-driven fluid simulations using regression forests, ACM Trans. Graph. 34 (6) (2015) 199:1–199:9.

[42] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, USA, 2016, pp. 481–490.

[43] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering 365 (2020) 113028.

[44] Y. Shin, J. Darbon, G. E. Karniadakis, On the convergence and generalization of physics informed neural networks (2020). arXiv:2004.01806.

[45] X. Jin, S. Cai, H. Li, G. E. Karniadakis, NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations (2020). arXiv:2003.06496.

[46] C. Meneveau, P. Sagaut, Large Eddy Simulation for Incompressible Flows: An Introduction, Springer, 2006.

[47] R. A. Ibrahim, Liquid Sloshing Dynamics: Theory and Applications, Cambridge University Press, 2005.

[48] V. Dumoulin, F. Visin, A guide to convolution arithmetic for deep learning (2016). arXiv:1603.07285.

[49] A. Radford, L. Metz, S. Chintala, Unsupervised representation learning with deep convolutional generative adversarial networks (2015). arXiv:1511.06434.

[50] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by Exponential Linear Units (ELUs) (Nov. 2015). arXiv:1511.07289.

[51] A. L. Maas, Rectifier nonlinearities improve neural network acoustic models, in: ICML Workshop on Deep Learning for Audio, Speech, and Language Processing, 2013.

[52] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/

[53] M. Ihmsen, N. Akinci, G. Akinci, M. Teschner, Unified spray, foam and air bubbles for particle-based fluids, The Visual Computer 28 (2012) 669–677.

Closure

This thesis provided four new ways to perform forward propagation of parametric uncertainties through a computational model. Our findings will now be synthesised, which will conclude this thesis.

Conclusions

Conclusion 1: ***Efficient sampling vs. solver enhancement? There is no winner, only synergy***
Part I of this thesis focussed on efficient sample placement for UQ in order to reduce the overall computational cost. Part II explored a completely different approach, where the solver was enhanced (non-)intrusively in order to reduce the computational cost of performing a single sample, while keeping the same accuracy. If one has to choose between using an efficient sample approach or using a neural network to enhance the solver, then one needs to keep in mind what the underlying problem is. If the random space is high-dimensional and/or when there are discontinuities in the QoI response, then it is wise to start by checking if efficiently placing samples using the methods in Part I is sufficient to create an accurate surrogate. If the only problem is the computational time required to produce a single solution, then the methodology described in Part II is beneficial. An overview of the different scenarios and which method to use is given below:

- **Does the QoI have a discontinuous response?**
  Part I chapter 1
- **Is the random space not a hypercube?**
  Part I chapter 2
- **Is the underlying model a PDE and fast to sample from?**
  Part I chapter 2
- **Is the underlying model computationally expensive to sample and is the QoI independent of time?**
  Part II chapter 3
- **Is the underlying model computationally expensive to sample and is the QoI depending on time?**
  Part II chapter 4

Additionally, as both approaches are independent, one may choose to combine both approaches, i.e., first decrease computational time of a single sample using the methodologies discussed in Part II, and then perform efficient sample placement based on the algorithms discussed in Part I.

Conclusion 2: ***Non-intrusive vs. intrusive machine learning? Go for intrusive if possible***
Part II of this thesis showed how machine learning can be used to enhance the accuracy of low-fidelity solutions of a computational solver, both in an intrusive and non-intrusive manner. We tested both approaches for enhancing solutions coming from a PIC/FLIP fluid flow solver. The non-intrusive approach showed a great amount of flexibility due to its appearance in the UQ chain as a post-processing tool. This flexibility comes at the cost of applicability, i.e., the non-intrusive approach does not properly deal with a temporal coordinate in the solution, and the method has to be applied multiple times when the QoI is time dependent. As opposed to this, intrusively using neural networks inside a legacy solver shows great potential in terms of accuracy and computational cost and even allows for some generalisation outside the training set. There is no rule of thumb (yet) on where and how to incorporate a neural network into a computational solver, as it completely

depends on the mathematical model and numerical discretisation and it is therefore difficult to extrapolate the conclusions drawn from the PIC/FLIP test case to other test cases.

Conclusion 3: ***Discontinuities are hard to deal with, but also hard to come by***
Part I showed how to properly deal with discontinuities in the QoI when sampling. Algorithms that focus on either smooth or discontinuous solution approximation become inaccurate when constructing a surrogate for the other case, which may lead to an increase in overall computational time and/or accuracy of the constructed surrogate. With our MST-ME method we demonstrated that it is possible for an algorithm to make a distinction between a smooth response or discontinuous response and devise proper sample placement in both cases. However, it is important to note that when coming up with test cases for this work, it was hard to construct a relevant test case in which there was a discontinuity present in the QoI. This finding strengthens our approach as it indeed turns out that the presence/absence of discontinuities in the QoI is uncertain, which makes it difficult to choose between an algorithm that is suited for approximating either only smooth or only discontinuous QoIs.

Conclusion 4: ***We do not always need machine learning***
In Part II of this thesis we focused on using neural networks (non-)intrusively in a solver to enhance its accuracy. The question that arises is whether we really need to use machine learning for this. The non-intrusive approach uses a neural network that functions as a mapping between a coarse and a fine grid. The multi-level approach in Part II does not need a neural network until the point where the goal is to construct a solution approximation with an error close to machine precision. To clarify, training the neural networks can sometimes be costly and does not provide sufficient benefits over a more conventional approach like MLSC when the goal is to construct approximations with accuracies that are orders of magnitude larger than machine precision. However, when the goal is to enhance a time-dependent low-fidelity solution by using the intrusive approach, the neural networks are crucial as the amount of data in the training set increases swiftly and often highly non-linear mappings are required to enhance such low-fidelity solutions.

Conclusion 5: ***When using machine learning, finding solid mathematical foundation is difficult***
As mentioned in Part II, there is mathematical foundation for the approximation properties of multilayer perceptrons. In computational engineering, where a solid mathematical foundation is beneficial, this simple neural network architecture seems like a suitable candidate to use. However, we experienced that finding an optimal neural network architecture and proper training parameters is not trivial as in some cases we lack a mathematical foundation to thoroughly explain our choices. This issue is reflected by the vastly improving field of explainable-AI, whose goal is to gain understanding on how these neural network models operate and make decisions.

Outlook

The work presented in this thesis showed new methods for efficient sampling and solver enhancement, with as goal to perform fast and efficient UQ. Besides the significant improvement already realised over conventional methods like SC, there is still a lot of room for further improvement. A list of possible extensions for the works shown in this thesis is given below:

- ***Part I: sampling for discontinuous responses***
  - Not all discontinuities in the QoI response have endpoints that coincide with the boundary of the random space. One could extend the discontinuity finding such that it is able to connect discontinuities that do not divide the random space into multiple simply connected subdomains.
  - In the MST-ME method not all samples are used when constructing the interpolant. Accounting for stability in the interpolant during sampling may result in a sample distribution that can be used entirely when constructing the final interpolant.
- ***Part I: PDE-informed sampling***
  - The method currently uses a Chebyshev basis defined on the smallest hypercube comprising the entire (non-hypercube) random space. This basis may be far from optimal when the random space has an exotic shape. Therefore, constructing a suitable basis that exploits the shape of the underlying random space may benefit the sampling approach.
- ***Part II: non-intrusive solver enhancement***
  - The Monte-Carlo sampling used to construct the training/validation data for the neural network mappings is far from optimal. Constructing sample locations based on for instance Part I, chapter 2 may lower the required computational cost.
- ***Part II: intrusive solver enhancement***
  - The proposed intrusive approach fails to generalise to fluid flow simulations that introduce new phenomena, e.g., obstacles in the flow. One may carefully construct training data to improve generalisation capabilities, without significantly increasing computational cost for training the neural network.
  - The time step that needs to be chosen to keep the multi-fidelity approach stable is rather restrictive. One may opt to use an adaptive grid to make the multi-fidelity simulations stable, even for larger time steps.

Publication list

An adaptive minimum spanning tree multielement method for uncertainty quantification of smooth and discontinuous responses
Yous van Halder, Benjamin Sanderse, Barry Koren
SIAM Journal on Scientific Computing 41 (6), A3624-A3648, 2019

PDE/PDF-informed adaptive sampling for efficient non-intrusive surrogate modelling
Yous van Halder, Benjamin Sanderse, Barry Koren
International Journal for Uncertainty Quantification, 2152-5080, 2021

Multi-level neural networks for PDEs with uncertain parameters
Yous van Halder, Benjamin Sanderse, Barry Koren
arXiv: 2004.13128 (revision under review at Journal of Computational Physics)

Multi-fidelity neural networks for real-time fluid flow predictions
Yous van Halder, Benjamin Sanderse, Barry Koren
arXiv: 2106.03491 (in preparation)

Faster Flow Predictions with Intrusive Neural Networks
Yous van Halder, Benjamin Sanderse
ERCIM News 122: Solving Engineering Problems with Machine Learning, 2020

Machine Learning for Closure Models in Multiphase-Flow Applications
Jurriaan Buist, Benjamin Sanderse, Yous van Halder, Barry Koren, Gert-Jan van Heijst
UNCECOMP 2019, Conference Proceedings, 2019