Spectral Methods for Uncertainty Quantification

Emil Brandt Kærgaard

Kongens Lyngby 2013
IMM-M.Sc.-2013-0503

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Building 303B, DK-2800 Kongens Lyngby, Denmark
www.compute.dtu.dk IMM-M.Sc.-2013-0503

Summary (English)

This thesis has investigated the field of Uncertainty Quantification with regard to differential equations. The focus is on spectral methods and especially the stochastic Collocation method. Numerical tests are conducted and the stochastic Collocation method for multivariate problems is investigated. The Smolyak sparse grid method is applied in combination with the stochastic Collocation method on two multivariate stochastic differential equations. The numerical tests showed that the sparse grids can reduce the computational effort and at the same time produce very accurate results.

The first part of the thesis introduces the mathematical background for working with Uncertainty Quantification, the theoretical background for generalized Polynomial Chaos and the three methods for the Uncertainty Quantification. These methods are the Monte Carlo sampling, the stochastic Galerkin method and the stochastic Collocation method.
The three methods have been tested numerically on the univariate stochastic Test equation to test accuracy and efficiency and further numerical tests of the Galerkin method and the Collocation method have been conducted with regard to the univariate stochastic Burgers' equation. The numerical tests have been carried out in Matlab.

The last part of the thesis involves an introduction of the multivariate stochastic Collocation method and numerical tests. Furthermore a few methods for reducing the computational effort in high dimensions is introduced. One of these methods - the Smolyak sparse grids - is applied together with the stochastic Collocation method on the multivariate Test equation and the multivariate Burgers' equation. This resulted in accurate and efficient estimates of the statis-

tics.

Summary (Danish)

Denne afhandling undersøger og anvender metoder til kvantificering af usikkerhed. Fokus i opgaven har været at undersøge spektrale metoder til usikkerhedsbestemmelse med særligt fokus på den stokastiske Collocation metode. Der er foretaget numeriske undersøgelser af metoderne og den stokastiske Collocation metode er blevet afprøvet på multivariate problemer. Den stokastiske Collocation metode er desuden blevet testet sammen med Smolyak sparse grids på to multivariate differentialligninger hvilket var en klar effektivisering og førte til nøjagtige resultater.

Den første del af afhandlingen introducerer den matematiske baggrund for at arbejde med emnet og en teoretisk introduktion af generaliseret Polynomial Chaos og de tre metoder til usikkerhedsbestemmelse. De anvendte metoder er Monte Carlo Sampling, den stokastiske Galerkin metode og den stokastiske Collocation metode.
Der er foretaget numeriske test med de tre metoder på den univariate Test equation for at undersøge effektivitet og nøjagtighed. Desuden er der foretaget yderligere numeriske undersøgelser af Galerkin metoden og Collocation metoden på den univariate stokastiske Burgers' equation. De numeriske test er foretaget i Matlab.

Den sidste del af afhandlingen omhandler introduktion og afprøvning af den multivariate stokastiske Collocation metode. Der bliver også introduceret flere metoder til at reducere den beregningsmæssige byrde der opstår når antallet af dimensioner øges. En af disse metoder kaldes Smolyak sparse grid metoden og denne er blevet anvendt sammen med den stokastiske Collocation metode på den multivariate Test equation og den multivariate Burgers' equation, hvilket

førte til gode resultater.

Preface

This thesis was prepared at DTU Compute: Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfilment of the requirements for acquiring an M.Sc. in Mathematical Modelling and Computing. The thesis was carried out in the period from December 3rd 2012 to May 3rd 2013 with associate Professor Allan P. Ensig-Karup as supervisor.

The thesis deals with spectral methods for Uncertainty Quantification and introduces a method to decrease the computational effort of these methods in high dimensions.
The thesis consists of a study of methods for Uncertainty Quantification and the application of these. The focus has been on conducting Uncertainty Quantification for differential equations and using spectral methods to investigate the uncertainty. Especially the stochastic Collocation method has been investigated and the use of this method in more than one dimension.

The thesis is written to a reader that has basic knowledge of mathematical modelling and mathematics. It is furthermore assumed that the reader has knowledge of MATLAB and of numerical tools for solving differential equations.

Lyngby, 03-May-2013

Emil Brandt Kærgaard

Acknowledgements

I would like to thank my supervisor, Associate Professor Allan P. Ensig-Karup, and Ph.D. student Daniele Bigoni for help, advice and inspiration during the project. They have provided guidance and advice through the project whenever I asked for it.

Another person who deserves thanks is Emil Kjær Nielsen who has been exploring this interesting topic alongside me. We have had some great conversations and discussions and I really appreciate his inputs and his humor.

Furthermore I would like to thank Signe Søndergaard Jeppesen for her morale support and for listening when I needed it.

Last but not least I would like to thank Emil Schultz Christensen and Kristian Rye Jensen for their readiness to help, their advice and their support.

Contents

Chapter 1

Introduction

The fields of numerical analysis and scientific computing have been in great development the last couple of decades. The introduction of efficient computers has lead to massive use of mathematical models and extensive research have been conducted to optimize the use of these models.
A topic of great interest is the errors in the numerical results and it is currently a field of active research. This has lead to an increased knowledge within the field and many improvements have been obtained. In general the errors of numerical analysis is classified into three groups [16]: Model errors, numerical errors and data errors.

The model errors arises when the applied mathematical model does not describe a given problem exactly. This type of errors will almost always be present when describing real life problems since the applied mathematical models are simplifications of the problem and since they are based on a set of assumptions.
Numerical errors arises when the mathematical models are solved numerically. The numerical methods usually involves approximations of the exact model solutions and even though these approximations can be improved by using theory about convergence and stability, the errors will still be there due to the finite precision on computers.
The data errors refers to the uncertainty in the input to the mathematical models which propagates through the model to the output. These errors are unavoid-

able when the input data is measurement data, since measurements will always contain errors. Errors in the input also arise when there is limited knowledge about the input variables or parameters.

1.1 Uncertainty Quantification

As outlined here there are lots of errors to investigate when working with numerical analysis. The focus in this thesis is on quantifying the uncertainty that propagates from the input to a mathematical model to the output. It is a field of very active research and many different approaches are applied in order to optimize the quantification of the uncertainty. In this thesis the focus will be on a class of methods that are known as spectral methods and on quantifying uncertainty in stochastic differential equations. An example is the ordinary differential equation called the Test equation which is formulated as

$$\frac{d}{dt}u(t) = -\lambda u(t), \quad u(0) = b.$$

By introducing uncertainty in the λ-parameter the solutions to the Test equation based different realizations of the stochastic λ-parameter will vary a lot. In figure 1.1 9 of such solutions are plotted and it is seen that there is a lot of variation in the solutions and the uncertainty is increased with time. The 9 solutions in figure 1.1 are plotted in the timespan $[0, 1]$ but if the timespan was increased to e.g. $[0, 2]$ the difference in the solutions would have been even greater. In fact the uncertainty grows exponentially with time. This is a very good reason for applying uncertainty quantification.

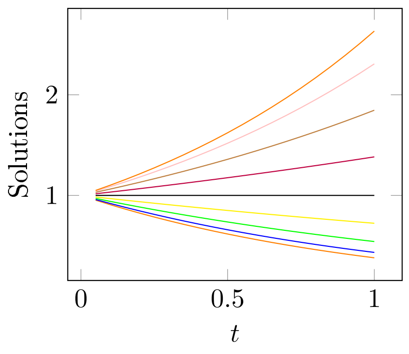

Figure 1.1: 9 deterministic solutions to the stochastic Test equation

In this thesis three methods will be used for conducting uncertainty quantification. One of them is the Monte Carlo Sampling which is a brute force method that based on a large number of samples approximates the statistics of the stochastic solutions. The samples refers solutions to the stochastic problem which are obtained by solving the system deterministically for a set of realizations of the stochastic inputs.
The two other methods are spectral methods which are based on using orthogonal polynomials to represent the stochastic solution of the problem at hand. The two methods are called the stochastic Collocation method and the stochastic Galerkin method. The stochastic Collocation method does in general not require many implementations of a solver for the deterministic problem is available.
The stochastic Galerkin method often requires analytical derivations and new implementations. But it is usually obtains more accurate results than the stochastic Collcation method so both methods have strengths and weaknesses.

1.2 Motivation and goals for the thesis

The combination of spectral methods and uncertainty quantification is a field of study which is in great development these years and a lot of research is currently going on worldwide. Besides this the uncertainty quantification has become an important numerical tool that is applied in many companies.
The studies within the field of uncertainty quantification involves many different topics and approaches. One topic that has received a lot of attention recently is how to reduce the computational effort when applying spectral methods such as stochastic Collocation method to multivariate problems.
Because of the rapid development within uncertainty quantification this thesis includes a section that introduces a few interesting approaches to decrease the computational effort based on a literature study of recent scientific articles.

The overall goal of the thesis is to investigate the field of uncertainty quantification and gain an insight in the topic. The focus is the combination of spectral methods and uncertainty quantification.
This means that the thesis will include an introduction to some of the necessary theory for investigating the field and then introduce several uncertainty quantification methods which involve at least one intrusive and one non-instrusive method. The goal of the thesis is to introduce the methods theoretically and conduct numerical tests.
The focus of the thesis is to introduce the topic and to apply spectral methods for uncertainty quantification. The spectral methods for uncertainty quantifi-

cation are attractive in many ways but the computational effort of the methods often grows rapidly in multivariate problems. It is therefore a goal of the thesis to apply at least one method to reduce the computational effort.
It is a choice not to investigate more advanced PDEs and instead include a more advanced setting where the computational effort is sought to be reduced.

It is a choice not to investigate more advanced PDEs and instead include a more advanced setting where the computational effort is sought to be reduced.

1.3 Basic literature

The book *Numerical methods for stochastic computations: a spectral method approach* by Dongbin Xiu has served as a basis for working with uncertainty quantification and the book *Spectral Methods for Uncertainty Quantification* by O. Le Maître and O. Knio has been used as a reference.
Furthermore the article *Gaussian Quadrature and the Eigenvalue Problem* by John A. Gubner has been used as a basis for some of the mathematical background.

1.4 Outline of thesis

This thesis has a focus on spectral methods for uncertainty quantification and includes theoretical and numerical applications of such.
In Chapter 2 the mathematical background is outlined. This includes orthogonal polynomials, quadratures rules and numerical tools. The mathematical introduction is continued in Chapter 3 which contains some basic concepts within probability theory. These two chapters also serves as an introduction to the notation that is used throughout the thesis.
Chapter 4 gives an introduction to generalized Polynomial Chaos (gPC), the gPC expansions and the the important properties of gPC. The generalized Polynomial Chaos serves as a basis for introducing the stochastic Galerkin method and the stochastic Collocation method in chapter 5. The theoretical background for applying these two methods is outlined in this chapter and the chapter serves as a basis for conducting numerical tests. Chapter 6 introduces an ordinary differential equation (ODE) denoted the Test Equation and a partial differential equation (PDE) called Burgers' Equation. These two differential equations will form the basis of the numerical tests.

In chapter 7 numerical tests of the univariate Test equation is conducted and the accuracy and convergence of the uncertainty quantification methods and chapter 8 contains numerical tests involving Burgers' equation. The numerical tests of the uncertainty quantification methods has confirmed some of the properties outlined in the theory and Chapter 9 gives a brief summery of the strengths and weaknesses for methods and outline which method that will be use in the rest of the thesis.
Chapter 10 is a literature study which describes some interesting topics which are recently introduced in the field of uncertainty quantification. Chapter 11 introduces the multivariate stochastic collocation method. The multivariate stochastic Collocation method is investigated in Chapter 12 through numerical tests and in chapter 13 the Smolyak sparse grids are applied in combination with the multivariate stochastic Collocation method.
Chapter 14 is a discussion of the methods and the conducted numerical tests. It includes a section which outline some possible future works.

Chapter 2

Mathematical background

The uncertainty quantification conducted in this thesis is based on theory within many fields. The two main fields of mathematics are probability theory and the theory regarding spectral methods.
In this chapter the basic tools for applying spectral methods are outlined as well as the numerical solvers for deterministic partial differential equations (PDE) used in this thesis.
In chapter 3 the stochastic theory will be outlined. This means that unless anything else is stated the theory outlined in this chapter is deterministic.

2.1 Hilbert space and inner products

Before the theory of spectral methods can be outlined some basic definitions are needed which will be introduced here. First of all L^2 denotes a Hilbert space with a corresponding inner product and its induced norm. The L^2 space is defined such that for $u \in L^2$ it holds that $\|u\|_2 = \left(\int_S |u|^2 d\mu\right)^{\frac{1}{2}} < \infty$ where μ is a Lebesgue measure and S is the support of the measure.
The measure μ is in this thesis used such that the integral $\int_D f(x)d\mu(x)$ can be replaced by the weighted integral $\int_D f(x)w(x)dx$ with $w(x) = 0$ for x outside the domain D. Furthermore the following definition will be used $\mu(\mathbb{R}) =$

$\int_{-\infty}^{\infty} w(x)dx$. The general measure theory is beyond the scope of this thesis and the interested reader can see e.g. [16] or [19] for an introduction.
The inner products in this thesis will be defined as

$$\langle u, v\rangle = \int_S u(x)v(x)d\mu(x) = \int_S u(x)v(x)w(x)dx. \tag{2.1}$$

The inner product is actually a weighted inner product and is generally referred to as $\langle\cdot,\cdot\rangle_w$ but in this thesis the notation $\langle\cdot,\cdot\rangle$ is used for simplicity. The weighted Hilbert space $L^2_w[a,b]$ can be formulated as

$$L^2_w[a,b] = \{u : [a,b] \to \mathbb{R}\,|\, \int_a^b u^2(x)d\mu(x) = \int_a^b u^2(x)w(x)dx < \infty\}.$$

Furthermore a stochastic Hilbert space is used later on and it is defined as

$$L^2_{dF_Z}(I_Z) = \{f : I_Z \to \mathbb{R}\,|\,\mathbb{E}[f^2(Z)] = \int_{I_Z} f^2(z)dF_Z(z) < \infty\}, \tag{2.2}$$

where Z is a stochastic variable.

2.2 Orthogonal Polynomials

In this section a basic introduction to the theory of orthogonal polynomials is given and further information can be found in e.g. [1]. In the following $\mathfrak{N}$ will be used as either $\mathfrak{N} = \mathbb{N}_0 = \{0, 1, 2, \dots\}$ or as an index set $\mathfrak{N} = \{0, 1, \dots, N\}$. There are many different notations for an orthogonal polynomial and in this project the following will be adopted

$$\Phi_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x_1 + a_0, \quad a_n \neq 0, \tag{2.3}$$

where the leading coefficient, a_n, is non-zero. Often the orthogonal polynomials are referred to in monic form which means that the leading coefficient is one. Since the leading coefficient is non-zero a transformation of the general form (2.3) into monic form is made by dividing with a_n

$$P_n(x) = \frac{\Phi_n(x)}{a_n} = x^n + \frac{a_{n-1}}{a_n}x^{n-1} + \cdots + \frac{a_1}{a_n}x_1 + \frac{a_0}{a_n}, \quad a_n \neq 0.$$

Another often used representation of the orthogonal polynomial which is introduced in e.g. [10] yields

$$\phi_n(x) = x^n + \sum_{k=0}^{n-1} \frac{\langle x^n, \phi_k(x)\rangle}{\langle \phi_k(x), \phi_k(x)\rangle}\phi_k(x), \quad n \geq 1.$$

When a system of polynomials $\{\Phi_n(x)\}_{n\in\mathfrak{N}}$ fulfils (2.4) it is said to be orthogonal with respect to a real positive measure μ.

$$\int_S \Phi_n(x)\Phi_m(x)d\mu(x) = \gamma_n\delta_{n,m}, \quad n,m \in \mathfrak{N}, \tag{2.4}$$

where S is the support of the measure μ, $\delta_{n,m}$ is the Kronecker delta function and γ_n is a normalization constant defined as $\int_S \Phi_n^2(x)d\mu(x) = \gamma_n$ for $n \in \mathfrak{N}$. As described in the previous section the measure μ typically has a density $w(x)$ which means that the integral (2.4) could be formulated as

$$\int_S \Phi_n(x)\Phi_m(x)w(x)dx = \gamma_n\delta_{nm}, \quad n,m \in \mathfrak{N}.$$

An orthonormal system of polynomials refers to a system of orthogonal polynomials which have normalization constants 1. To normalize a set of orthogonal polynomials the individual polynomials are divided by the corresponding normalization factors, i.e. $\tilde{\Phi}_n(x) = \frac{\Phi_n(x)}{\sqrt{\gamma_n}}$.

2.2.1 Three-term Recurrence relation

A general three-term recurrence relation can be formulated for $\Phi_n(x)$ with $x \in \mathbb{R}$ and states

$$-x\Phi_n(x) = \beta_n\Phi_{n+1}(x) + \alpha_n\Phi_n(x) + \gamma_n\Phi_{n-1}(x), \quad n \geq 1,$$

where $\beta_n, \gamma_n \neq 0$ and $\frac{\gamma_n}{\beta_{n-1}} > 0$ as introduced in [19]. Equivalently a three-term recurrence relation can be established for $\phi_n(x)$ in monic form which yields

$$\phi_{n+1}(x) = (x - a_n)\,\phi_n(x) - b_n\phi_{n-1}(x), \quad n \geq 1, \tag{2.5}$$

where

$$\begin{cases} a_n = \frac{\langle x\phi_n(x),\phi_n(x)\rangle}{\langle\phi_n(x),\phi_n(x)\rangle} \\ \\ b_n = \frac{\langle\phi_n(x),x\phi_{n-1}(x)\rangle}{\langle\phi_{n-1}(x),\phi_{n-1}(x)\rangle} = \frac{\langle\phi_n(x),\phi_n(x)\rangle}{\langle\phi_{n-1}(x),\phi_{n-1}(x)\rangle} \geq 0 \end{cases}$$

For more information on the three-term recurrences see [10] and [19].

2.2.2 Example: Hermite Polynomials

The Hermite Polynomials is an important class of polynomials and there are two often used definitions of the polynomials. Both are introduced in this thesis with different naming such that it is possible to differentiate between the two definitions. In both cases the polynomials are defined on the real line, i.e. $x \in \mathbb{R}$. One of the types of Hermite polynomials is used in Polynomial Chaos expansions and is denoted $H_n(x)$. These are defined here and the alternative definition denoted $He_n(x)$ can be seen in Appendix A. The weight functions for $H_n(x)$ are defined as a Gaussian

$$w(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}.$$

The $H_n(x)$ polynomials are defined as

$$H_n(x) = \frac{1}{(-1)^n e^{-\frac{x^2}{2}}} \frac{d^n}{dx^n}[e^{-\frac{x^2}{2}}] = n! \sum_{k=0}^{[n/2]} (-1)^k \frac{1}{k! 2^k (n-2k)!} x^{n-2k},$$

where $[n/2]$ refers to the largest integer that is smaller than or equal to $\frac{n}{2}$. The three-term recurrence relation is

$$H_{n+1}(x) = xH_n(x) - nH_{n-1}(x), \qquad (2.6)$$

where $H_0(x) = 1$ and $H_1(x) = x$. The normalization factor γ_n can be computed as

$$\gamma_n = \langle H_n(x), H_n(x) \rangle = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} H_n^2(x) e^{-\frac{x^2}{2}} dx = n!.$$

The first five $H_n(x)$ polynomials are given as

$$\begin{aligned} H_0(x) &= 1, \\ H_1(x) &= x, \\ H_2(x) &= x^2 - 1, \\ H_3(x) &= x^3 - 3x, \\ H_4(x) &= x^5 - 10x^3 + 15x. \end{aligned} \qquad (2.7)$$

In figure 2.1 these polynomials have been plotted

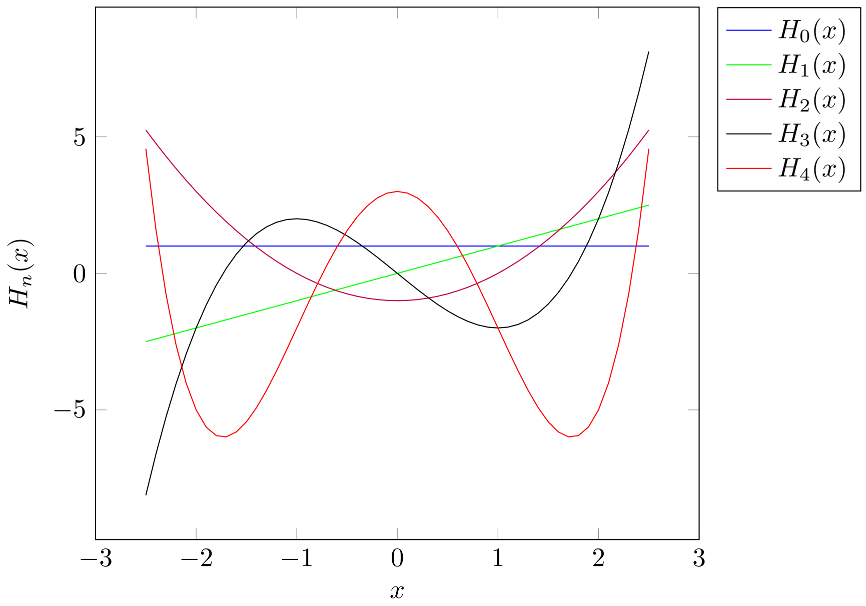

Figure 2.1: The first five Hermite polynomials.

The intuitive inspection of figure 2.1 seems to justify the implementation of the polynomials given in appendix B since the plotted polynomials are of the right order. They also fits very well with the analytical expressions for the first five polynomials given in (2.7) and resembles the figures in [16] and [19].

2.2.3 Example: Jacobi polynomials

The Jacobi polynomials are a widely used class of polynomials and they are defined as the eigenfunctions for a certain Sturm-Liouville problem which will not be described further here. A more in-depth description of the origin of the Jacobi Polynomials is found in [15].
The Jacobi polynomials, $\{P_n^{\alpha,\beta}\}$, can be described by the following three-term recurrence relation

$$a_{n+1,n}^{\alpha,\beta} P_{n+1}^{\alpha,\beta}(x) = (a_{n,n}^{\alpha,\beta} + x) P_n^{\alpha,\beta}(x) - a_{n-1,n}^{\alpha,\beta} P_{n-1}^{\alpha,\beta}(x), \tag{2.8}$$

where the first two polynomials are given by

$$\begin{aligned} P_0^{\alpha,\beta}(x) &= 1 \\ P_1^{\alpha,\beta}(x) &= \frac{1}{2}(\alpha - \beta + (\alpha + \beta + 2)x) \end{aligned}$$

and the coefficient for $n = 0$ is $a^{\alpha,\beta}_{-1,0} = 0$ and for $n > 0$ the coefficients are

$$\begin{cases} a^{\alpha,\beta}_{n-1,n} = \frac{2(n+\alpha)(n+\beta)}{(2n+\alpha+\beta+1)(2n+\alpha+\beta)} \\ \\ a^{\alpha,\beta}_{n,n} = \frac{\alpha^2-\beta^2}{(2n+\alpha+\beta+2)(2n+\alpha+\beta)} \\ \\ a^{\alpha,\beta}_{n+1,n} = \frac{2(n+1)(n+\alpha+\beta+1)}{(2n+\alpha+\beta+2)(2n+\alpha+\beta+1)}. \end{cases}$$

The weight function for the Jacobi polynomials is $w(x) = (1-x)^\alpha(1+x)^\beta$ and the normalization constant is given by

$$\gamma^{\alpha,\beta}_n = ||P^{\alpha,\beta}_n||^2_J = 2^{\alpha+\beta+1}\frac{(n+\alpha)!(n+\beta)!}{n!(2n+\alpha+\beta)(n+\alpha+\beta)!},$$

where $\|\cdot\|_J = \|\cdot\|_{L^2_w[-1,1]}$ is the norm which belongs to the space of Jacobi polynomials.

2.2.4 Example: Legendre Polynomials

The Legendre polynomials are a subclass of the Jacobi polynomials with $\alpha = \beta = 0$ and weight $w = 1$ and they will be used extensively in this thesis.
The Legendre polynomials $\{L_n(x)\} \in [-1,1]$ are usually normalized such that $L_n(1) = 1$ and in this normalized form the polynomials can be formulated as

$$L_n(x) = \frac{1}{2^n}\sum_{k=0}^{[\frac{n}{2}]}(-1)^k \begin{pmatrix} n \\ k \end{pmatrix} \begin{pmatrix} 2n-2k \\ n \end{pmatrix} x^{n-2k}, \tag{2.9}$$

where $[\frac{n}{2}]$ denote $\frac{n}{2}$ rounded down to nearest integer. For the weight function $w(x)$ the Legendre polynomials are an orthogonal basis for $L^2_w[-1,1]$. The first two Legendre polynomials are $L_0(x) = 1$ and $L_1(x) = x$ and the three-term recurrence relation is

$$(n+1)L_{n+1}(x) = (2n+1)xL_n(x) - nL_{n-1}(x). \tag{2.10}$$

It is often written in monic form as well which yields

$$L_{n+1}(x) = \frac{2n+1}{n+1}xL_n(x) - \frac{n}{n+1}L_{n-1}(x). \tag{2.11}$$

With the definition (2.11) of the Legendre pPolynomials the normalization constant γ_n can be computed as

$$\gamma_n = \langle L_n, L_n\rangle = \int_{-1}^{1} L^2_n(x)w(x)dx = \frac{1}{2n+1}.$$

The first five Legendre polynomials are given by

$$
\begin{aligned}
L_0(x) &= 1, \\
L_1(x) &= x, \\
L_2(x) &= \frac{1}{2}(3x^2 - 1), \\
L_3(x) &= \frac{1}{2}(5x^3 - 3x), \\
L_4(x) &= \frac{1}{8}(35x^4 - 30x^2 + 3),
\end{aligned}
\tag{2.12}
$$

$$\tag{2.13}$$

The polynomials have been plotted in figure 2.2.

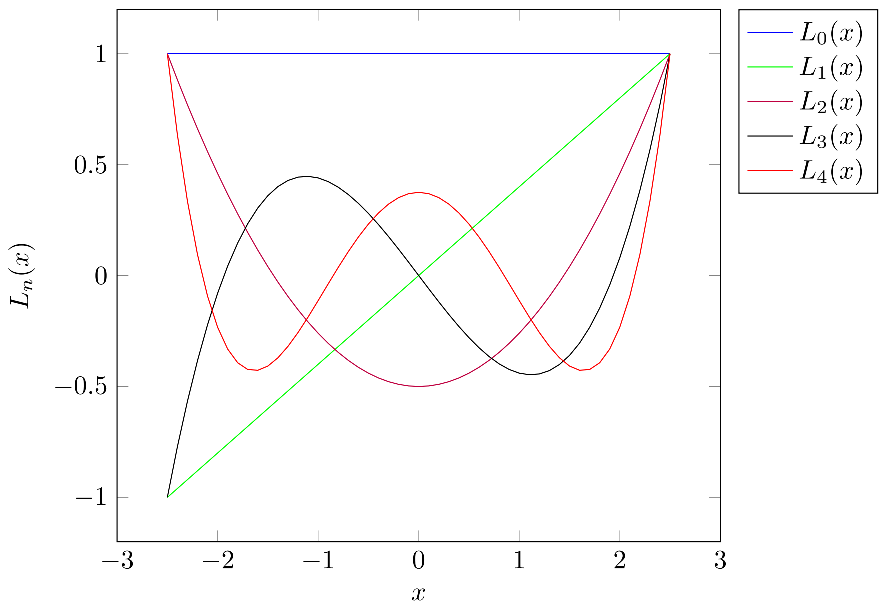

Figure 2.2: The first five Legendre polynomials.

The functions in figure 2.2 corresponds very well with the analytical expressions in (2.12) and resembles the figures in [16] and [19]. The implementation of the polynomials can be found in appendix B.

2.3 Gauss Quadrature

Quadrature rules are an important tool when conducting integration numerically and it is a widely used concept. It is a way to approximate an integral $\int_{I_x} f(x)\,\mathrm{d}\mu(x) = \int_{I_x} f(x)\,w_\mu(x)\,\mathrm{d}x$ where I_x is the domain of x and this is done by computing another integral $\int_{I_x} g(x)\,w_\mu(x)\,\mathrm{d}x$ where the function g is chosen such that its integral is known and it resembles f. Furthermore g is often chosen such that the integral can be evaluated as

$$\int_{I_x} g(x)\,w_g(x)\,\mathrm{d}x = \sum_{k=1}^{n} w_k g(x_k), \tag{2.14}$$

where w_k are weights and x_k are nodes that belongs to the range of integration. A class of functions that are widely used in quadratures as the approximating functions g is the class of polynomials and it can be shown that for a polynomial P of degree less than n and for the right nodes x_k and weights w_k it holds that

$$\int_{I_x} P(x)\,w_P(x)\,\mathrm{d}x = \sum_{k=1}^{n} w_k P(x_k). \tag{2.15}$$

Once the nodes x_k have been chosen, the weights w_k can be computed such that the relation (2.15) holds. This means that if g is chosen to be a polynomial the relation (2.14) holds which is a very nice property in numerical analysis. The orthogonal polynomials introduced earlier can be used in this context as g and the weights $w(x)$ plays a crucial role in the choice of polynomials. In order to resemble the integral $\int_{I_x} f(x)\,w_\mu(x)\,\mathrm{d}x$ in the best way the polynomial integration weights should be as "close" to the weights w_μ as possible.

2.3.1 Computing nodes and weights for Gauss Quadrature

The validity of (2.15) can be extended to polynomials f of degree $2n$ by choosing the nodes in a clever way - which is the idea behind Gaussian Quadrature. The idea is to use orthogonal polynomials on monic form, $\{\phi_n\}$, of degree up to N and compute the quadrature points, x_j, as the zeros of ϕ_{n+1} and the weights w_j can hereafter be computed, see e.g. [10] or [15]. In Theorem (2.1) a way of computing the Gauss quadrature points and weights are outlined as in [10].

Theorem 2.1 *By using a_n and b_n in the three-term recurrence relation* (2.5) *the quadrature nodes, x_j, and weights, w_j, can be computed by eigenvalue de-*

composition of the symmetric, tridiagonal Jacobi matrix

$$J_n = \begin{bmatrix} a_0 & \sqrt{b_1} & & & \\ \sqrt{b_1} & a_1 & \sqrt{b_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{b_{n-2}} & a_{n-2} & \sqrt{b_{n-1}} \\ & & & \sqrt{b_{n-1}} & a_{n-1} \end{bmatrix}.$$

This leads to defining $V^T J_n V = \Lambda = diag(\lambda_1, \dots, \lambda_n)$ where λ_j is the j'th eigenvalue, V is a matrix containing the corresponding eigenvectors and $V^T V = I$. From this definition the quadrature nodes and weights can be computed as

$$x_j = \lambda_j, \quad w_j = \mu(\mathbb{R}) v_{j,0}^2,$$

where $v_{j,0}$ is the first element of the j'th column of V, i.e. the first element of the j'th eigenvector.

The proof of the theorem can be found in [10]. As seen from the theorem the quadrature weights depends on $\mu(\mathbb{R})$. It is important to note that the definition of an orthogonal polynomial often relies on a normalized weight which means that $\mu(\mathbb{R}) = \int_{\infty}^{\infty} w_p(x)\, dx = 1$ where $w_p(x)$ is the polynomial integration weights. This is not necessarily a problem but using the normalized polynomial weights will lead to different quadrature weights than the ones that typically are represented in the literature.
The presentation of the Gauss Legendre quadrature and Gauss Hermite quadrature will therefore involve the non-normalized weights, i.e. $\mu(\mathbb{R}) \neq 1$.
Another thing to notice is that the general three-term recurrence relation used in theorem 2.1 is on monic form. This is not the case for many of the three-term recurrence relations used to describe the most common orthogonal polynomials. The recurrence relation for the Legendre polynomials given in (2.10) is an example of a recurrence that has to be modified in order to identify a_n and b_n.

2.3.2 Example: Gauss Hermite Quadrature

The Hermite polynomials $H_n(x)$ are defined to have a leading coefficient of 1 and in this section the polynomial weights are scaled with $\sqrt{2\pi}$ such that $d\mu(x) = e^{-\frac{x^2}{2}}$ and $\mu(\mathbb{R}) = \sqrt{2\pi}$. The scaling has been made to obtain the quadrature nodes that typically is occurs in the literature, see e.g. [10]. The recurrence relation for $H_n(x)$ is given by (2.6) and since the leading coefficient is 1 the recurrence relation is already in the appropriate monic form and it seen that $a_n = 0$ and $b_n = n$. Substituting these expressions into the Jacobi matrix J_n yields

$$J_n = \begin{bmatrix} 0 & \sqrt{1} & & & \\ \sqrt{1} & 0 & \sqrt{2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{n-2} & 0 & \sqrt{n-1} \\ & & & \sqrt{n-1} & 0 \end{bmatrix}.$$

The Gauss Hermite quadrature weights and nodes can be computed by an eigenvalue decomposition of this Jacobi matrix and the implementation can be found in appendix B in section B.1.1.

2.3.2.1 Example: Gauss Legendre Quadrature

The Legendre polynomials have a non-normalized $\mu(x)$ which yields $d\mu(x) = dx$ and thereby $\mu(\mathbb{R}) = \int_{-1}^{1} 1dx = 2$. Furthermore the leading coefficient is $\frac{(2n)!}{2^n(n!)^2}$ and the three-term recurrence relation is given by (2.10). First the term $L_{n+1}(x)$ is isolated on the left-hand side by dividing with $(n+1)$ as in (2.11) which yields

$$L_{n+1}(x) = \frac{2n+1}{n+1} x L_n(x) - \frac{n}{n+1} L_{n-1}(x).$$

In order to get this expression on the form of ϕ_n described in (2.5) a division with the leading order coefficient is carried out. This means that $\phi_n(x) = L_n(x)\frac{2^n(n!)^2}{(2n)!}$. Dividing with the leading coefficient for the $(n+1)$'th term in the three-term recurrence results in

$$\begin{aligned} \frac{2^{n+1}((n+1)!)^2}{(2(n+1))!} L_{n+1}(x) &= \frac{2^{n+1}((n+1)!)^2}{(2(n+1))!} \frac{2n+1}{n+1} x L_n(x) \\ &\quad - \frac{2^{n+1}((n+1)!)^2}{(2(n+1))!} \frac{n}{n+1} L_{n-1}(x). \end{aligned}$$

The derivations has been divided into two parts - one to compute the coefficient in front of $L_n(x)$ denoted c_1 and one for the coefficient in front of $L_{n-1}(x)$

denoted c_2. The derivations for computing c_1 are carried out below.

$$\begin{aligned}
c_1 &= \frac{2^{n+1}((n+1)!)^2}{(2(n+1))!}\frac{2n+1}{n+1}x \\
&= \frac{2^n 2((n+1)!(n+1)!)}{(2n+2)!}\frac{2n+1}{n+1}x \\
&= 2^n\frac{2n!(n+1)n!(n+1)}{(2n+2)(2n+1)(2n)!}\frac{2n+1}{n+1}x \\
&= 2^n\frac{2(n+1)n!n!}{(2n+2)(2n)!}x. \\
&= 2^n\frac{n!n!}{(2n)!}x
\end{aligned}$$

The second coefficient is computed as

$$\begin{aligned}
c_2 &= \frac{2^{n+1}((n+1)!)^2}{(2(n+1))!}\frac{n}{n+1} \\
&= \frac{2^{n-1}2^2((n+1)n(n-1)!)((n+1)n(n-1)!)}{(2n+2)(2n+1)(2n)(2n-1)(2n-2)!}\frac{n}{n+1} \\
&= 2^{n-1}\frac{2(2n+2)n((n-1)!)^2(n+1)n}{(2n+2)(2n+1)(2n)(2n-1)(2n-2)!}\frac{n}{n+1} \\
&= 2^{n-1}\frac{((n-1)!)^2 n}{(2n+1)(2n-1)(2n-2)!}n \\
&= \frac{2^{n-1}((n-1)!)^2}{(2n-2)!}\frac{n^2}{(2n+1)(2n-1)}.
\end{aligned}$$

Substituting these results into the expression obtained earlier yields

$$\begin{aligned}
\phi_{n+1}(x) &= c_1L_n(x) - c_2L_{n-1}(x) \\
&= 2^n\frac{(n!)^2}{(2n)!}xL_n(x) - \frac{n^2}{(2n+1)(2n-1)}\frac{2^{n-1}((n-1)!)^2}{(2n-2)!}L_{n-1}(x) \\
&= x\phi_n(x) - \frac{n^2}{4n^2-1}\phi_{n-1}(x).
\end{aligned}$$

which means that $a_n = 0$ and $b_n = \frac{n^2}{4n^2-1} = \frac{1}{4-n^{-2}}$. The Gauss Legendre quadrature has been implemented in MATLAB and can be seen in appendix B.1.1.

2.3.3 Gauss-Lobatto Quadrature

Another widely used class of quadrature is the Gauss-Lobatto quadrature that basicly is the same as described above just with the difference that the endpoints

are included in the quadrature nodes. This means that for example Legendre Gauss-Lobatto quadrature with n quadrature points involves the zeros of a Legendre polynomial as well as the end points -1 and 1.
The Gauss-Lobatto quadrature is generally not exact for as high orders of polynomials as the regular Gauss quadrature but it includes the end points which can be very useful in some cases, e.g. when solving boundary value problems, see [15]. Therefore the choice between Gauss quadrature and Gauss-Lobatto quadrature often depends on whether the boundary nodes plays a significant role or not for the solution.

2.4 Polynomial Interpolation

Polynomial interpolation is an approximation of a function f by use of polynomials, P_M, which satisfies that $f(x_i) = P_M(x_i)$ for a given set of nodes $\{x_i\}_{i=0}^{N}$. For the one dimensional case this yields

$$P_M(x_i) = a_M x_i^M + \cdots + a_1 x_i + a_0 = f(x_i), \quad i = 0, 1, \ldots, N.$$

When using polynomial interpolation the Lagrange polynomials are a widely used class of polynomials and the Lagrange polynomials, $h_i(x)$, are defined as

$$h_i(x) = \prod_{j=0, j\neq i}^{M} \frac{x - x_j}{x_i - x_j}, \quad i = 0, 1, \ldots, N.$$

The Lagrange polynomials has a special property, namely

$$h_j(x_i) = \delta_{i,j} = \begin{cases} 1, & i = j, \\ 0, & i \neq j, \end{cases}$$

where $\delta_{i,j}$ is the Kronecker delta function. The Lagrange polynomials are used to establish a nodal representation on the form

$$f(x) = \sum_{j=0}^{N} f(x_j) h_j(x),$$

and this form is called Lagrange form. Typically $M = N$ since this yields a unique interpolating polynomial $\mathcal{I}f(x) = Q_N(x)$. That $Q_N(x)$ is unique is seen by assuming that another interpolating polynomial $G_N(x)$ exists such that $G_N(x_i) = f(x_i)$ for $i = 0, 1, \ldots, N$. The difference between the two polynomials is a new polynomial of order N, i.e. $D_N(x) = Q_N(x) - G_N(x)$. This polynomial will have $N + 1$ distinctive roots which implies that $D_N(x)$ is the zero polynomial and thereby that $Q_N(x)$ is unique.

Besides having a nodal representation in Lagrange form of the interpolating polynomial a modal representation can be used as well. In the nodal representation introduced here it is the Lagrange polynomials $h_N(x)$ that are unknown but in modal form the coefficients are unknown and the polynomials are known - typically they are orthogonal polynomials like Legendre or Hermite polynomials. This means that the modal representation of f can be formulated as

$$f_N(x) = \sum_{j=0}^{N} \hat{f}_j \Phi_j(x),$$

where the coefficients typically are represented as

$$\hat{f}_j = \frac{1}{\gamma_j} \int_{D_x} f(x)\Phi_j(x)w(x)dx,$$

where D_x is the domain of x, $w(x)$ is a weight chosen according to the polynomials Φ_j in order to secure orthogonality and γ_j is the normalization constant for the polynomials.

2.4.1 Relationship between nodal and modal representations

Interpolating a function f in a discrete setting with polynomials is done by using a modal or a nodal interpolating polynomial representation. By using Lagrange polynomials $h(x)$ as the nodal basis functions and appropriately chosen polynomials $\Phi(x)$ for the modal representation the following relationship between the modal and nodal representations can be established [7]

$$f(x_i) = \sum_{j=0}^{N} \hat{f}_j \Phi_j(x_i) = \sum_{j=0}^{N} f_j h_j(x_i), \quad i = 0, \ldots, N$$

where x_i are the nodal interpolation points, which in this project typically are generated by use of a quadrature rule. The relationship between the modal and nodal coefficients in the discrete interpolations can be expressed by use of the Vandermonde matrix $\mathcal{V}$ which elements are computed as $\mathcal{V}_{i,j} = \Phi_j(x_i)$. The relationship between the coefficients can be expressed as

$$\mathbf{f} = \mathcal{V}\hat{\mathbf{f}}, \tag{2.16}$$

where $\mathbf{f}$ and $\hat{\mathbf{f}}$ are vectors containing the nodal and modal coefficients, respectively. Since the nodal and modal bases are related as $\mathbf{f}^T h(x) = \hat{\mathbf{f}}^T \Phi(x)$ the basis functions can also be related by use of the Vandermonde matrix. By use of (2.16) it is seen that the following holds

$$\mathbf{f}^T h(x) = (\mathcal{V}\hat{\mathbf{f}})^T h(x) = \hat{\mathbf{f}}^T \mathcal{V}^T h(x). \tag{2.17}$$

This means that the bases can be related as

$$\begin{array}{rcl} \hat{\mathbf{f}}^T \mathcal{V}^T h(x) & = & \hat{\mathbf{f}}^T \Phi(x) \\ & \Updownarrow & \\ \mathcal{V}^T h(x) & = & \Phi(x) \end{array}$$

From this result it is clear that the Vandermonde matrix can transform the discrete nodal basis functions into the discrete modal basis functions and vice versa.

2.5 Spectral methods

The theory introduced here about orthogonal polynomials serves as the basis for spectral methods. The orthogonal polynomials serves as a basis for polynomial expansions on which the class of spectral methods is based on. The spectral methods in general has spectral convergence which is one the main properties that makes the methods attractive. The spectral methods for uncertainty quantification will be introduced in chapter 5.

2.6 Numerical solvers

In this section a numerical method for for solving PDE's in space will be outlined. Furthermore a method for solving PDE's in time is introduced.

2.6.1 Solver for differential equations in space

When solving the differential equations in space there are a lot of different methods and approaches. In this section a spectral method for solving a non-periodic PDE is introduced.

2.6.1.1 Deterministic Collocation Method

The deterministic Collocation method is based on polynomial interpolation of the solution u which means that a set of interpolation points, x_i, has to be chosen [7]. These points are typically known as collocation points and the Collocation method relies on a discrete nodal representation of the solution on the form

$$\mathcal{I}_N u = \sum_{i=0}^{N} u_n h_n.$$

The collocation points used in this thesis will be chosen to be Gauss Lobatto nodes. The deterministic Collocation method is based on a linear representation of the solution and a linear operator $\mathcal{L}$ is introduced. This means that a differential equation on the form

$$\frac{d}{dx}\left(a\frac{du}{dx}\right) + b\frac{du}{dx} + cu = f(x), \quad x \in D_x, \tag{2.18}$$

where D_x is the domain of x can be represented by a linear differential operator defined as

$$\mathcal{L} = \frac{d}{dx}\left(a\frac{d}{dx}\right) + b\frac{d}{dx} + c. \tag{2.19}$$

By introducing an interpolation derivative $\mathcal{D}$ as

$$\mathcal{D}v = \frac{d}{dx}\mathcal{I}_N v \tag{2.20}$$

a discrete expression for $\mathcal{L}$ denoted $\mathcal{L}_N$ can be formulated for evaluation of an approximation to the solution u in the interior collocation points. This means that

$$\mathcal{L}_N = \mathcal{D}\left(A\mathcal{D}\right) + B\mathcal{D} + C, \tag{2.21}$$

can be used to evaluate $\mathcal{L}_N(\mathcal{I}_N u(x_i)) = f(x_i)$ for $i = 1, \ldots, N-1$. Some adjustment are however necessary since the boundary conditions has not been enforced yet.
The boundary conditions can be enforced in different ways by modifying the established system of equations and this will be described in the implementation section 8.2.1. For further information see [7].
In order to use the deterministic Collocation method later on it has to be established how the spatial differential matrix can be computed. This is outlined in the following section.

2.6.1.2 Construction of a spatial differential matrix

By use of the Vandermonde matrix it is possible to construct a discrete operator for computing an approximation of the first derivative in nodal space. The first derivative in modal space can in discrete form be formulated as

$$\frac{d}{dx}\left(\sum_{j=0}^{N}\hat{f}_j\Phi_j(x)\right)=\sum_{j=0}^{N}\hat{f}_j\frac{d}{dx}\Phi_j(x).$$

In matrix form this can be expressed as

$$\frac{d\mathbf{f}}{dx}=\mathcal{V}_x\hat{\mathbf{f}},\quad \mathcal{V}_{x,ij}=\left.\frac{d\Phi_j}{dx}\right|_{x_i}.$$

In nodal space an approximation of the first derivative in space can be expressed as

$$\frac{d}{dx}\left(\sum_{j=0}^{N}f_jh_j(x)\right)=\sum_{j=0}^{N}f_j\frac{d}{dx}h_j(x).$$

In matrix form the nodal approximation can be formulated as

$$\frac{d\mathbf{f}}{dx}=\mathcal{D}\mathbf{f},\quad \mathcal{D}_{i,j}=\left.\frac{dh_j}{dx}\right|_{x_i},$$

where $\mathbf{f}$ is a vector containing the function evaluations. Now an expression for the derivative matrix $\mathcal{D}$ can be derived.

$$\frac{d\mathbf{f}}{dx}=\mathcal{D}\mathbf{f}=\mathcal{D}\left(\mathcal{V}\hat{\mathbf{f}}\right)=\mathcal{V}_x\,\hat{\mathbf{f}},$$

where $\hat{\mathbf{f}}$ is a vector containing the modal coefficients and from this it is seen that $\mathcal{D}=\mathcal{V}_x\mathcal{V}^{-1}$. The matrices $\mathcal{V}$ and $\mathcal{V}_x$ can be expressed as

$$\mathcal{V}=\begin{bmatrix}\Phi_0(x_1) & \Phi_1(x_1) & \cdots & \Phi_N(x_1)\\ \Phi_0(x_2) & \Phi_1(x_2) & \cdots & \Phi_N(x_2)\\ \vdots & \vdots & \ddots & \vdots\\ \Phi_0(x_N) & \Phi_1(x_N) & \cdots & \Phi_N(x_N)\end{bmatrix}$$

and

$$\mathcal{V}_x=\begin{bmatrix}\Phi_0(x_1)' & \Phi_1(x_1)' & \cdots & \Phi_N(x_1)'\\ \Phi_0(x_2)' & \Phi_1(x_2)' & \cdots & \Phi_N(x_2)'\\ \vdots & \vdots & \ddots & \vdots\\ \Phi_0(x_N)' & \Phi_1(x_N)' & \cdots & \Phi_N(x_N)'\end{bmatrix}.$$

The implementation of these matrices can be found in appendix B.

2.6.1.3 Legendre polynomials and the differential matrix

In this thesis the Legendre polynomials will be used as basis polynomials in the deterministic Collocation method. In practise the orthogonal basis polynomials are often normalized [7] which means that the Vandermonde matrix can be expressed as

$$\mathcal{V} = \begin{bmatrix} \widetilde{L}_0(x_1) & \widetilde{L}_1(x_1) & \cdots & \widetilde{L}_N(x_1) \\ \widetilde{L}_0(x_2) & \widetilde{L}_1(x_2) & \cdots & \widetilde{L}_N(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{L}_0(x_N) & \widetilde{L}_1(x_2) & \cdots & \widetilde{L}_N(x_N) \end{bmatrix}, \tag{2.22}$$

where $\widetilde{L}_i(x)$ is the normalized i'th order Legendre polynomial. The Vandermonde matrix for the first derivatives in space with regard to a Legendre basis can be expressed as

$$\mathcal{V}_x = \begin{bmatrix} \widetilde{L}_0(x_1)' & \widetilde{L}_1(x_1)' & \cdots & \widetilde{L}_N(x_1)' \\ \widetilde{L}_0(x_2)' & \widetilde{L}_1(x_2)' & \cdots & \widetilde{L}_N(x_2)' \\ \vdots & \vdots & \ddots & \vdots \\ \widetilde{L}_0(x_N)' & \widetilde{L}_1(x_N)' & \cdots & \widetilde{L}_N(x_N)' \end{bmatrix}. \tag{2.23}$$

This means that when an expression for the derivative of the Legendre polynomials has been established the spatial derivative matrix for a Legendre basis can be computed. As a subclass of the Jacobi polynomials the derivatives of the Legendre polynomials can be derived from the expression for the derivatives of the Jacobi polynomial.
In general the derivative of the Jacobi polynomials can be computed as

$$\frac{d^k}{dx^k} P_n^{\alpha,\beta}(x) = \frac{\Gamma(n+\alpha+\beta+1+k)}{2^k \Gamma(n+\alpha+\beta+1)} P_{n-k}^{\alpha+k,\beta+k}(x),$$

where $\Gamma(a) = (a-1)!$. As mentioned earlier the orthonormal polynomials are for computational reasons used instead of the standard polynomials and the normalized Jacobi polynomials can be expressed as

$$\widetilde{P}_n^{\alpha,\beta}(x) = \frac{P_n^{\alpha,\beta}(x)}{\gamma_n^{\alpha,\beta}}. \tag{2.24}$$

This means that the general expression for the derivative of the normalized Jacobi polynomials is

$$\frac{d^k}{dx^k} \widetilde{P}_n^{\alpha,\beta}(x) = \frac{\Gamma(n+\alpha+\beta+1+k)}{2^k \Gamma(n+\alpha+\beta+1)} \frac{\gamma_{n-k}^{\alpha+k,\beta+k}}{\gamma_n^{\alpha,\beta}} \widetilde{P}_{n-k}^{\alpha+k,\beta+k}(x).$$

The first order derivative of the orthonormal Jacobi polynomials can be derived to be

$$\frac{d}{dx}\widetilde{P}_n^{\alpha,\beta}(x) = \sqrt{n(n+\alpha+\beta+1)}\widetilde{P}_{n-1}^{\alpha+1,\beta+1}(x).$$

This means that the derivative of the Legendre polynomials can be expressed as

$$\frac{d}{dx}\widetilde{L}_n(x) = \sqrt{n(n+1)}\widetilde{P}_{n-1}^{1,1}(x).$$

Now the basis for using the deterministic Collocation method has been established and the implementations of the matrices $\mathcal{V}$, $\mathcal{V}_x$ and $\mathcal{D}$ can be seen in appendix B.

2.6.2 Solvers for time dependent problems

There exists a lot of different solvers for solving time dependent problems and each of them has strengths and weaknesses. In this section a solver is outlined for time dependent problems where the time dependent term is differentiated once with respect to the time t. This means that the problem can be outlined as

$$\frac{du(x,t)}{dt} = \mathcal{L}(u), \quad D \times T,$$

where $\mathcal{L}$ is a differential operator, D is the spatial domain and T is the time domain. There exists a class of solvers for time dependent differential equations which is called Runge-Kutta methods and one of these has been implemented in MatLab. The implemented method is an explicit Runge-Kutta method and is denoted ERK [7].

2.6.2.1 ERK: Explicit fourth-order Runge-Kutta

This method is an explicit fourth-order Runge-Kutta which means that it is a one-step method that relies only on the current solution when computing the next solution. The method can be outlined as

$$\begin{aligned}
U &= u(x,t_i)\\
G &= U\\
P &= f(t_i,U)\\
U &= U + \tfrac{1}{2}\Delta t P\\
G &= P\\
P &= f(t_i + \tfrac{1}{2}\Delta t, U)\\
U &= U + \tfrac{1}{2}\Delta t(P-G)\\
G &= \tfrac{1}{6}G\\
P &= f(t_i + \tfrac{1}{2}\Delta t, U) - \tfrac{1}{2}P\\
U &= U + \Delta t P\\
G &= G - P\\
P &= f(t_i + \Delta t, U) + 2P\\
u(x,t_{i+1}) &= U + \Delta t(G + \tfrac{1}{6}P)
\end{aligned}$$

where $u(x,t_i)$ denotes the current solution and $u(x,t_{i+1})$ the solution in the next time step. The method has been implemented in MATLAB in the file `ERK.m` and can be seen in appendix B.

Chapter 3

Basic concepts within Probability Theory

This chapter will give a brief introduction to some basic concepts within probability theory which are used in relation to Uncertainty Quantification. The interested reader can find a more thorough description in [16] or [19].

3.1 Distributions and statistics

In order to cope with the stochastic problems investigated in this thesis some basic definitions about distributions and statistics has to be introduced. First of all a random variable X has a (cumulative) distribution function, $F_X(x)$, and a (probability) density function. The distribution function, F_X, is defined as a collection of probabilities that fulfils

$$F_X(x) = P(X \leq x) = P(\{\omega : X(\omega) \leq x\}). \tag{3.1}$$

Due to the definition of the probability P it holds that $0 \leq F_X \leq 1$. The probability density function, $f_X(x)$, is closely linked to the cumulative distribution function through the relations

$$F_X(a) = \int_{-\infty}^{a} f_X(x)dx \quad \text{and} \quad f_x(x) = \frac{d}{dx}F_X(x). \tag{3.2}$$

The cumulative distribution function is as indicated by the name a cumulated probability in the sense that it computes the probability of X being in an interval and does not obtain any value in a single point. The density function on the other hand is intuitively a way of computing the probability in a point or in an infinitesimal interval.

3.1.1 Example: Gaussian distribution

A common and important class of distribution is the Gaussian distribution which is also called the normal distribution and is denoted $\mathcal{N}(\mu, \sigma^2)$, where $\mu \in \mathbb{R}$ is the mean, $\sigma^2 > 0$ is the variance and σ is called the standard deviation. The probability density function (PDF) can be formulated as

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad x \in \mathbb{R}. \tag{3.3}$$

The distribution is often used with the parameters $\mu = 0$ and $\sigma^2 = 1$, i.e. $\mathcal{N}(0, 1)$, which is referred to as the standard normal distribution. The PDF for the standard normal distribution is plotted in figure 3.1.

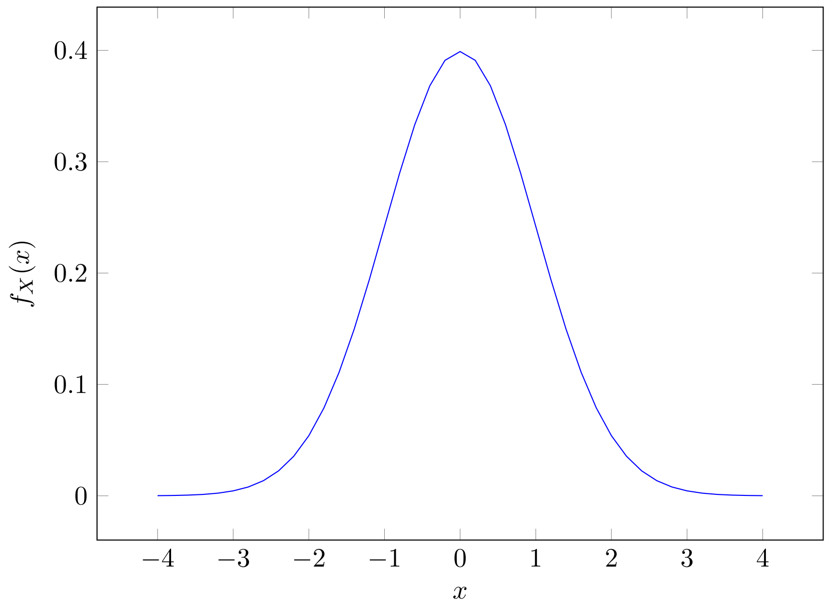

Figure 3.1: The PDF for the standard normal distribution.

3.1.2 Example: Uniform distribution

The uniform distribution is also an important class and its main characteristic is that all outcomes within the support of the distribution is equally probable. Furthermore the support is defined as an interval on the real line and characterized by $I_U = [a, b]$.
The PDF of the uniform distribution is defined as

$$f_X(x) = \begin{cases} \frac{1}{b-a} & x \in [a, b] \\ 0 & x \notin [a, b] \end{cases}. \tag{3.4}$$

The PDF for the uniform distribution with support $I_X = [0, 1]$ is plotted in figure 3.2

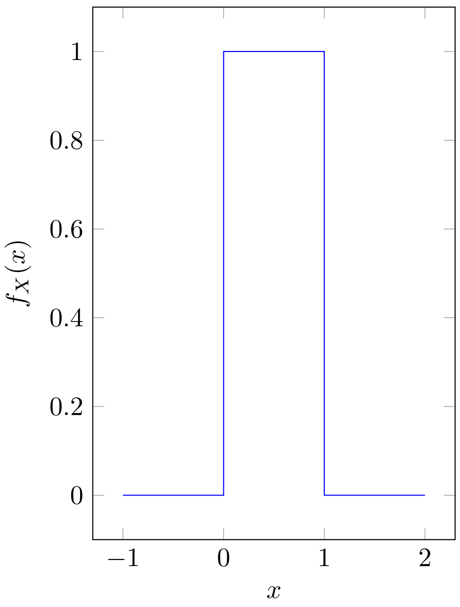

Figure 3.2: The PDF for the uniform distribution.

3.1.3 Multiple dimensions

In multiple dimensions it is useful to define the joint density function and the marginal density function. The definitions given here will for simplicity be defined for two variables but it can be extended to any dimension. The definitions will be based on [17].
The joint density function is defined as a function $f(x, y)$ that for $-\infty < x, y < \infty$ fulfils

$$\begin{cases} f(x, y) \geq 0 & \forall x, y, \\ \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y)\, dxdy = 1. \end{cases} \tag{3.5}$$

The multivariate cumulative distribution function can be defined as

$$F(a, b) = \int_{-\infty}^{a} \int_{-\infty}^{b} f(x, y)\, dxdy \tag{3.6}$$

A new type of function can be introduced for the multivariate case namely the marginal density function which is defined as $f_x(x) = \int_{-\infty}^{\infty} f(x, y)\, dy$. The function $f_x(x)$ has to fulfil

$$\begin{cases} f_x(x) \geq 0 & \forall x \\ \int_{-\infty}^{\infty} f_x(x)\, dx = 1. \end{cases} \tag{3.7}$$

The marginal density in y is defined equivalently and denoted $f_y(y)$.

3.1.4 Statistics

In this thesis there is mainly computed two statistics in the numerical investigations, namely the expectation and the variance. The expectation of a random variable X with density function F_x can be computed as

$$\mu_X = \mathbb{E}[X] = \int_{I_X} x \, dF_X(x) = \int_{I_X} x f_X \, d(x),$$

where I_X is the domain of the variable and f_X is the PDF. The variance of X can be computed as

$$\sigma_X^2 = \text{var}(X) = \int_{I_X} (x - \mu_X)^2 \, dF_X(x) = \int_{I_X} (x - \mu_X)^2 \, f_X d(x).$$

Furthermore for $m \in \mathbb{N}$ the m'th moment of X can be computed as

$$\mathbb{E}[X^m] = \int_{I_X} x^m \, dF_X(x) = \int_{I_X} x^m f_X \, d(x).$$

It is also possible to define a moment generating function and an example of this can be seen in [19].

3.1.5 Inner product and statistics

In many cases it is possible to express the statistics in terms of inner products. This of course requires the definition of an inner product and a proper space where this inner product is defined. The inner product was defined in a deterministic setting as (2.1) and in a stochastic setting it can be defined as

$$\langle u, v \rangle_{dF_X} = \int_{I_X} u(X) v(X) \, dF_X = \int_{I_X} u(X) v(X) \, f_X d(x).$$

By comparing this definition with the definition of the statistics it is seen that the statistics can be represented by use of the inner product. This means for instance that the expectation can be computed as

$$\mathbb{E}[X] = \langle \sqrt{X}, \sqrt{X} \rangle_{dF_X} = \int_{I_X} x \, dF_X(x) = \int_{I_X} x f_X \, d(x).$$

The corresponding space is $L^2_{dF_X}(I_X) = \{f : I_X \to \mathbb{R} \,|\, \mathbb{E}[f^2] < \infty\}$ with the norm defined by $\|f\|_{dF_X} = \sqrt{\mathbb{E}[f^2]}$.

3.2 Input parameterizations

The objective of this thesis is to study the effects of having uncertainty on the input variables to a mathematical model. In practice this leads to some restrictions on the input variables. To be usable in a computer they have to be finite and they should be independent, since most methods within the field require this in practise. Two approaches for achieving this will be outlined here and they are based on [16] and [19].

When the stochastic input is system parameters the input is already parametrized. The focus is therefore on ensuring the independence of the parameters. The procedure can be outlined as in Algorithm 1.

Algorithm 1 Procedure for ensuring independence of stochastic system parameters.

```
1: Let the system parameters be defined as $Y = (Y_1, \dots Y_n)$ with $n \geq 1$ and
   with distribution function $F_Y(y) = P(Y \leq y)$ where $y \in \mathbb{R}^n$.
2: Find a suitable transformation function $T$ such that $Y = T(Z)$ for $Z =
   (Z_1, \dots, Z_d) \in \mathbb{R}^d$ being a set of mutually independent random variables.
```

Similarly the approach can be outlined when the input parameters are stochastic processes which is seen in Algorithm 2.

Algorithm 2 Procedure for parameterization of random processes.

```
1: Let the stochastic process that models the stochastic inputs be defined as
   $(Y_k, k \in D_k)$ where $k$ is an index belonging to the index set $D_k$. The index
   $k$ can belong to either a time or a space domain.
2: Find a suitable transformation function $T$ such that $Y_k = T(Z)$ for $Z =
   (Z_1, \dots, Z_d)$ with $d \geq 1$ being a set of mutually independent random variables.
```

Often the stochastic process is not finite dimensional which means some kind of finite approximation has to be established. This introduces the problem with precision contra efficiency of the numerical solvers. An increasing number of terms in the finite approximation usually leads to increasing precision but also leads to more computational work. For more information see [16] and [19].

CHAPTER 4

Generalized Polynomial Chaos

The Generalized Polynomial Chaos (gPC) is outlined in this section. The gPC expansions described in the following are based on smooth orthogonal polynomials such as Hermite polynomials or Legendre polynomials. In fact the original polynomial chaos was based on Hermite polynomials and proved to be quite effective, when the stochastic parameters are Gaussian distributed but generally it is slow for non-Gaussian distributed variables. Therefore the gPC was introduced where the orthogonal basis polynomials are chosen according to the distribution of the stochastic parameters.
The gPC is a way of representing a stochastic process $X(\omega)$ parametrically. This is done by using an expansion with orthogonal polynomials such that a random field $X(\omega)$ can be represented by

$$X(\omega) = \sum_{j=0}^{\infty} \hat{c}_j \Phi_j(\omega),$$

where Φ represent the given basis polynomial and ω the random variable(s). The polynomials are chosen such that the weights that ensures the orthogonal property $\langle \Phi_i, \Phi_j \rangle = \int_D \Phi_i(x)\Phi_j(x)w(x)dx = \langle \Phi_i^2 \rangle \delta_{i,j}$ resembles the probability distribution function of the random variables. If a random variable, ω, is standard normal distributed it means that the probability density function is

$\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}\sigma^2}$ which corresponds to the weight function w_H of the Hermite polynomials. In this way a relation between the distribution of a random variable and the chosen basis functions can be outlined as in Table 4.1.

Distribution of Z	gPC basis polynomials	Support
Gaussian	Hermite	$(-\infty, \infty)$
Gamma	Laguerre	$[0, \infty)$
Beta	Jacobi	$[a, b]$
Uniform	Legendre	$[a, b]$

Table 4.1: Correspondence between distributions and the generalized Polynomial Chaos.

4.1 Generalized Polynomial Chaos in one dimension

In a single dimensional stochastic variable space with the random variable Z the gPC basis functions are chosen to be orthogonal polynomials, $\{\Phi_k(x)\}$ where the type of polynomials are coupled with the distribution of Z according to table 4.1.
The approximation of a function f by gPC can be either in weak or strong form depending on how much information is available about f and Z. Both the weak and strong gPC approximations are introduced in appendix A.
The type of gPC approximation that is typically used in this thesis is the gPC projection, where the N'th degree orthogonal gPC projection for a function $f \in L^2_{dF_Z}(I_Z)$ is

$$P_N f(Z) = \sum_{i=0}^{N} \hat{f}_i \Phi_i(Z), \quad \hat{f}_i = \frac{1}{\gamma_i}\mathbb{E}[f(Z)\Phi_i(Z)],$$

where $\{\gamma_i\}$ are the normalization factors. The convergence of the orthogonal projection follows from the theory but the convergence rate depends heavily on the smoothness of f [19]. The smoother f the faster the convergence.

4.2 Multivariate Generalized Polynomial Chaos

The focus is now on gPC in multiple dimensions. The variables $Z = (Z_1, \dots, Z_d)$ are assumed to be mutually independent and if this is not the case with the

stochastic variables in the original problem a parametrization can be conducted in order to achieve the independence of the variables.
The variables Z_i have support I_{Z_i} and marginal distribution $P_{Z_i}(z_i) = P(Z_i < z_i)$ and since the variables are independent it holds that

$$F_Z(z_1, \dots, z_d) = P(Z_1 \leq z_1, \dots, Z_d \leq z_d) = \prod_{i=1}^{d} F_{Z_i}(Z_i),$$

and $I_Z = I_{Z_1} \times \cdots \times I_{Z_d}$.
For the variable Z_i a corresponding set of univariate gPC basis functions are introduced, i.e. $\{\phi_k(Z_i)\}_{k=0}^{N} \in \mathbb{P}_N(Z_i)$, which are polynomials in Z_i of degree up to N.

In order to introduce a compact notation for the multivariate gPC expansion a so-called multi-index is introduced. It is a vector of indices for the d variables, $\mathbf{i} = (i_1, \dots, i_d)$, and $|\mathbf{i}| = i_1 + \cdots + i_d$ by definition. As an example the first multi-indices for $d = 3$ are listed in Table 4.1.

$\lvert\mathbf{i}\rvert$	Multi-index $\mathbf{i}$	Single Index k
0	(0 0 0)	1
1	(1 0 0)	2
	(0 1 0)	3
	(0 0 1)	4
2	(2 0 0)	5
	(1 1 0)	6
	(1 0 1)	7
	(0 2 0)	8
	(0 1 1)	9
	(0 0 2)	10
3	(3 0 0)	11
	(2 1 0)	12
	...	...

(4.1)

As seen from Table 4.1 it is possible to use a single index instead of the multi-index. Both types of indices are commonly used in the literature and in most cases it is just a matter of preference.
Now it is possible to define a set of basis functions for all d variables and these are introduced as $\Phi_{\mathbf{i}}(Z) = \phi_{i_1}(Z_1) \cdots \phi_{i_d}(Z_d)$ and $0 \leq |\mathbf{i}| \leq N$.
The expectation of the multivariate gPC can be expressed as

$$\mathbb{E}[\Phi_{\mathbf{i}}(Z)\Phi_{\mathbf{j}}(Z)] = \int \Phi_{\mathbf{i}}(z)\Phi_{\mathbf{j}}(z) dF_Z(z) = \gamma_{\mathbf{i}}\delta_{\mathbf{i},\mathbf{j}},$$

where $\delta_{\mathbf{i},\mathbf{j}} = \delta_{i_1,j_1} \dots \delta_{i_d,j_d}$ is the d-variate Kronecker delta function and $\gamma_{\mathbf{i}} = \mathbb{E}[\Phi_{\mathbf{i}}^2] = \gamma_1 \dots \gamma_d$ is the normalization factor. The basis functions $\Phi_{\mathbf{i}}$ belongs to the polynomial space

$$\mathbb{P}_N^d(Z) = \{p : I_Z \to \mathbb{R} | \, p(Z) = \sum_{|\mathbf{i}| \leq N} c_{\mathbf{i}} \Phi_{\mathbf{i}}(Z)\},$$

which has the dimension $\dim \mathbb{P}_N^d = \begin{pmatrix} N+d \\ N \end{pmatrix}$. Otherwise the polynomials $\Phi_{\mathbf{i}}(Z)$ can be defined to be of degree up to N in each dimension which implies a polynomial space, $\tilde{\mathbb{P}}_N^d$, of dimension $\dim \tilde{\mathbb{P}}_N^d = N^d$. This choice of space usually results in too many basis functions to be evaluated in practise for large dimensions.

As in the univariate case a multivariate gPC projection can be defined as

$$P_N f = \sum_{|\mathbf{i}| \leq N} \hat{f}_{\mathbf{i}} \Phi_{\mathbf{i}}(Z),$$

where the coefficients can be computed as

$$\hat{f}_{\mathbf{i}} = \frac{1}{\gamma_{\mathbf{i}}} \mathbb{E}[f \Phi_{\mathbf{i}}] = \frac{1}{\gamma_{\mathbf{i}}} \int f(z) \Phi_{\mathbf{i}}(z) dF_Z(z), \quad \forall |\mathbf{i}| \leq N.$$

The d-variate gPC projection is conducted in the space $L^2_{dF_Z}$ which is defined as (2.2).

4.3 Statistics for gPC expansions

The gPC expansions can be used not only to approximate a function f but also to estimate the statistics of f. If $u(x,t,Z)$ is a random process with $x \in D_x$, $t \in T$ and $Z \in \mathbb{R}^d$ then the N'th order gPC expansion can be expressed as

$$u^N(x,t,Z) = \sum_{|\mathbf{i}| \leq N} \hat{u}_{\mathbf{i}}(x,t) \Phi_{\mathbf{i}}(Z) \in \mathbb{P}_N^d,$$

for any fixed $x \in D_x$ and $t \in T$. The orthogonality of the basis functions enables the following computations of the statistics. For instance the mean can be approximated as $\mathbb{E}[u(x,t,Z)] \approx \mathbb{E}[u^N(x,t,Z)]$ and the computation of

$\mathbb{E}[u^N(x,t,Z)]$ yields

$$\begin{aligned}
\mathbb{E}[u^N(x,t,Z)] &= \int \left(\sum_{|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(z) \right) dF_Z(z) \\
&= \int \left(\sum_{|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(z) \right) \Phi_{\mathbf{0}}(z) dF_Z(z) \\
&= \hat{u}_{\mathbf{0}}(x,t).
\end{aligned}$$

The orthogonality of the basis functions has been utilized as well as the fact the zero-order polynomial $\Phi_{\mathbf{0}}(Z)$ is defined to be ones which explains how it could be introduced in the second equality. The variance can be computed as

$$\mathrm{var}(u(x,t,Z)) = \mathbb{E}[(u(x,t,Z) - \mu_u(x,t))^2].$$

By using u^N the variance can be approximated by

$$\begin{aligned}
\mathbb{E}[(u^N(x,t,Z) - \mu_{u^N}(x,t))^2] &= \int \left(\sum_{|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(z) - \hat{u}_{\mathbf{0}}(x,t) \right)^2 dF_Z(z) \\
&= \int \left(\sum_{0<|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(z) \right)^2 dF_Z(z) \\
&= \sum_{0<|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}}^2(x,t)\gamma_{\mathbf{i}}.
\end{aligned}$$

where the orthogonality ensures the validity of the second to last equality sign since

$$\int \left(\hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(z)\right)\left(\hat{u}_{\mathbf{j}}(x,t)\Phi_{\mathbf{j}}(z)\right) dF_Z(z) = 0 \quad \text{for } i \neq j.$$

Other statistics can be approximated as well by applying their definitions to the gPC approximation u^N [19].

Chapter 5

Stochastic Spectral Methods

Uncertainty Quantification (UQ) will in this project be with regard to solving Partial Differential Equations (PDEs). The PDEs in this thesis can in general for a time domain T and spatial domain $D \subset \mathbb{R}^\ell$ with $\ell = 1, 2, 3, \dots$ be formulated as

$$\begin{aligned} u_t(x,t,\omega) &= \mathcal{L}(u), \quad D \times T \times \Omega \\ \mathcal{B}(u) &= 0,, \quad \partial D \times T \times \Omega \\ u &= u_0, \quad \partial D \times [t=0] \times \Omega, \end{aligned}$$

where $\omega \in \Omega$ are the random inputs of the system in a probability space $(\omega, \mathcal{F}, P)$, $\mathcal{L}$ is a differential operator, $\mathcal{B}$ is the boundary condition (BC) operator and u_0 is the initial condition (IC).
In many cases it is required to restate the random variables ω such that a parametrization, $Z = (Z_1, \dots, Z_d) \in \mathbb{R}^d$ with $d \geq 1$, consisting of independent random variables is used instead. This means that the PDE system is on the form

$$\begin{aligned} u_t(x,t,Z) &= \mathcal{L}(u), \quad D \times T \times \mathbb{R}^d \\ \mathcal{B}(u) &= 0, \quad \partial D \times T \times \mathbb{R}^d \\ u &= u_0, \quad \partial D \times [t=0] \times \mathbb{R}^d, \end{aligned} \tag{5.1}$$

This general formulation will be used in the following when introducing the techniques for UQ.

5.1 Non-intrusive methods

The non-intrusive methods are generally speaking a class of methods which relies on realizations of the stochastic system - i.e. deterministic solutions of the underlying non-stochastic system. This is an interesting feature of the non-intrusive methods for stochastic systems, since well-known solvers can be used without any particular modifications. Another interesting part is when the deterministic solutions are decoupled the solutions can be computed in parallel.
A drawback with the non-intrusive methods is the computational effort that grows with the number of deterministic solutions to be computed. This drawback will be further described later and is an important topic in UQ.

5.1.1 Monte Carlo Sampling

The Monte Carlo Sampling (MCS) is based on constructing a system of independent and identical distributed (i.i.d.) variables Z. Then a system like (5.1) is solved as a deterministic system for M different realizations of Z and thereby obtaining M solutions of the type $u^{(i)}(x,t) = u(x,t,Z^i)$, where Z^i refers to the i'th realization of Z.
When the M solutions have been computed the solution statistics can be estimated. For example the mean of the solution can be estimated according to the Central Limit Theorem (CLT) as

$$\bar{u} = \frac{1}{M}\sum_{i=1}^{M} u^i.$$

This is as mentioned only an estimation of the true mean, $\bar{u} \approx \mathbb{E}(u) = \mu_u$, and an error estimate of MCS follows from the CLT [19].
Since the M solutions $u(x,t,Z^i)$ are i.i.d. it means that for $M \to \infty$ the distribution of $\hat{u}$ converges towards a Gaussian distribution $\mathcal{N}(\mu_u, \frac{\sigma_u^2}{M})$, where μ_u and σ_u are the exact mean and standard deviation of the solution, respectively. This means that the standard deviation of the Gaussian distribution is $M^{-\frac{1}{2}}\sigma_u$ and from this the convergence rate is established as $\mathcal{O}(M^{-\frac{1}{2}})$ [19].
It is important to note that only two requirements are to be met in order to use MCS. Namely that the system is on the right form and that a solver for the deterministic system is at hand. When these requirements are fulfilled a

convergence rate of $\mathcal{O}(M^{-\frac{1}{2}})$ can be obtained independently of the dimension of the random space, which is quite unique. It is however worth noting that the convergence rate is rather slow and if the deterministic solver is time consuming then it will take an immense amount of time to obtain a decent accuracy on the estimates of the statistics.
It should also be mentioned that there exists several methods which are based on the Monte Carlo method but has e.g. better efficiency. These methods are generally known as Quasi-Monte Carlo methods but it lies outside the scope of this thesis to investigate these methods.

5.1.2 Stochastic Collocation Method

The stochastic collocation method (SCM) is a stochastic expansion method that in general relies on expansion through Lagrange interpolation polynomials. The overall idea is to choose a set of collocation points $Z_M = \{Z^j\}_{j=1}^M$ in the random space. Then (5.1) is enforced in each of the nodes Z^j which means that the following system is solved for $j = 1, \dots, M$

$$\begin{aligned} u_t(x,t,Z^j) &= \mathcal{L}(u), \quad D \times T \times \mathbb{R}^d \\ \mathcal{B}(u) &= 0,, \quad \partial D \times T \times \mathbb{R}^d \\ u &= u_0, \quad \partial D \times [t=0] \times \mathbb{R}^d. \end{aligned}$$

This system is deterministic for each j and hence the SCM involves solving M deterministic systems. This is a very broad definition of the SCM and it would include the Monte Carlo sampling. Usually when using the SCM a clever choice of collocation points is made - e.g. choosing the points by a quadrature rule and exploit the belonging quadrature weights when computing the statistics.
The solution of the PDE (5.1) can be representated by use of Lagrange interpolating functions which have been described earlier. Hence the solution u is represented by an interpolation

$$\tilde{u}(Z) \equiv \mathcal{I}(u) = \sum_{j=1}^{M} u(Z^j) h_j(Z), \tag{5.2}$$

where h_j are the Lagrange polynomials and $u(Z^j) = u(x,t,Z^j)$. It is important to remember that the Lagrange polynomials are defined in a appropriate interpolation space V_I and that $h_i(Z^j) = \delta_{ij}$ for $i,j \in [1,\dots,M]$. This means that the interpolation $\tilde{u}(Z)$ is equal to the exact solution in each of the M collocation points.
From M deterministic solutions in the collocation points the statistics of the interpolation can be computed and thereby represent the statistics of the stochas-

tic solution to (5.1). The mean of the interpolation $\hat{u}$ can for instance be computed as

$$\mathbb{E}[\tilde{u}] = \sum_{j=1}^{M} u(Z^j) \int_\Gamma h_j(z)\rho(z)dz, \tag{5.3}$$

where Γ is the random space in which Z is defined and $\rho(z)$ is a distribution specific weight - namely the probability density function of the distribution of Z. The evaluation of the expectation can be non-trivial and knowledge of the Lagrange polynomials is needed. This can be obtained by use of an inverted Vandermonde-type matrix but it often requires quite a lot of work [20].
Another approach is to use a quadrature rule to evaluate the integral which leads to

$$\mathbb{E}[\tilde{u}] = \sum_{j=1}^{M} u(Z^j) \sum_{k=1}^{M} h_j(z_k)\rho(z_k)w_k, \tag{5.4}$$

where z_k are the quadrature points and w_k are the quadrature weights. The approximation of the integral in (5.3) by quadrature is exact since the Lagrange polynomials are of order M and the quadrature is exact for polynomials of this order.
The attained expression for the mean of the interpolation can be further simplified by choosing the collocation points smartly. The quadrature nodes and weights are chosen so they represent the distribution of the random parameters. This means that the quadrature points could be chosen as collocation points, i.e. $z_j = Z^j$. Hence the characteristic of the Lagrange polynomials, $h_i(Z^j) = \delta_{ij}$ for $i, j \in [1, \ldots, M]$, can be exploited once again and (5.4) reduces to

$$\mathbb{E}[\tilde{u}] = \sum_{j=1}^{M} u(Z^j)\rho(Z^j)w_j.$$

In the same way the variance of the interpolation can be computed as

$$\begin{aligned}
\text{var}[\tilde{u}] &= \mathbb{E}[(\tilde{u} - \mathbb{E}[\tilde{u}])^2] \\
&= \int_\Gamma (\tilde{u} - \mathbb{E}[\tilde{u}])^2 \rho(z)dz \\
&= \int_\Gamma \left(\sum_{j=1}^{M} u(Z^j)h_j(z) - \mathbb{E}[\tilde{u}] \right)^2 \rho(z)dz \\
&= \sum_{k=1}^{M} \left(\sum_{j=1}^{M} u(Z^j)h_j(z_k) - \mathbb{E}[\tilde{u}] \right)^2 \rho(z_k)w_k.
\end{aligned}$$

where the integral of the expectation is evaluated by use of the appropriate quadrature rules. Again the quadrature points and collocation points could be the same points which leads to

$$\begin{aligned}\mathrm{var}[\tilde{u}] &= \sum_{k=1}^{M}\left(\sum_{j=1}^{M} u(Z^j)\delta_{ij} - \mathbb{E}[\tilde{u}]\right)^2 \rho(Z^k)w_k. \\ &= \sum_{k=1}^{M}\big(u(Z^k) - \mathbb{E}[\tilde{u}]\big)^2 \rho(Z^k)w_k.\end{aligned}$$

In the same way other statistics could be computed as well if needed. The approach outlined here is an interpolation approach but there is another common approach within the stochastic Collocation methods which is referred to as a pseudo-spectral approach which uses discrete projection. This approach will however not be pursued further but more information can be found in e.g. [19].

5.2 Intrusive methods

Unlike the non-intrusive methods the intrusive methods rely on modifying the initial problem and in this project the focus will be on the intrusive polynomial chaos (PC) methods that relies on a weighted residual formalism.
These methods leads to spectral convergence but on the contrary to the non-intrusive methods implies extra analytical and numerical work. The modification of the stochastic system to be solved can be very troublesome and the resulting stochastic system often requires solvers that are different from the usual deterministic solvers [16]. Hence it is a trade-off between the extra analytical and numerical work and the much appreciated convergence.
When using stochastic Galerkin-like methods many of the properties from the deterministic Galerkin method is inherited - e.g. that the representation is optimal in the mean-square error [16].

5.2.1 Stochastic Galerkin method

The stochastic gPC Galerkin method (SGM) is a intrusive spectral method that relies on spectral approximations using generalized Polynomial Chaos (gPC). The idea of the SGM method is to define a solution in the space of polynomials such that the residue of the system (5.1) is orthogonal to the space of polynomials. The general approach is outlined in algorithm 3.

Algorithm 3 Outline for the Stochastic Galerkin Method.

```
1: Identify the source of uncertainty and compute a set of independent random
   variables with an appropriate PDF to represent this.
2: Construct a generalized Polynomial Chaos basis and use this to construct a
   gPC expansion to represent the system.
3: Modify the governing system by substituting with the gPC expansion.
4: Apply the Galerkin procedure to obtain a set of coupled deterministic equa-
   tions instead of stochastic equations.
5: Use appropriate numerical methods to solve the system of equations.
6: Post-processing - compute the needed statistics from the obtained solution.
```

For further information about this general outline see [12]. In the general system (5.1) the uncertainty has been identified and is described by the independent parameters $Z = (Z_1, \ldots, Z_d) \in \mathbb{R}^d$ where $d \geq 1$.
The gPC basis is chosen so it matches the distribution of the random parameters Z and this correspondence is outlined for the most common distributions in Table 4.1. The gPC basis is denoted $\{\Phi_k(Z)\}_{k=0}^N$ and belongs to the space $\mathbb{P}_N^d$ that consists of all the polynomials in Z of degree up to N. As described earlier the gPC basis consists of orthogonal polynomials in the random space which means that $\mathbb{E}[\Phi_k \Phi_j] = \delta_{kj}\gamma_k$.
The approximation by a gPC expansion of the solution to (5.1) can be described as

$$u(x,t,Z) \approx u^N(x,t,Z) = \sum_{|\mathbf{i}|=0}^{N} \hat{u}_{\mathbf{i}}(x,t)\Phi_{\mathbf{i}}(Z), \tag{5.5}$$

where $\mathbf{i}$ is the multi-index described earlier and

$$\hat{u}_{\mathbf{i}}(x,t) = \frac{1}{\gamma_{\mathbf{i}}}\mathbb{E}[u(x,t,Z)\Phi_{\mathbf{i}}].$$

This means that the number of terms in (5.5) is $P + 1 = \frac{(N+d)!}{(d)!(N)!}$. By using the single index described earlier and by defining $\{\Phi_p\}_{p=0}^{P+1}$ as the collection of basis polynomials the gPC expansion of the solution can be expressed as in [12] by

$$u^N(x,t,Z) = \sum_{p=0}^{P+1} \hat{u}_p(x,t)\Phi_p(Z), \tag{5.6}$$

where

$$\hat{u}_p(x,t) = \frac{1}{\gamma_p}\mathbb{E}[u(x,t,Z)\Phi_p].$$

Now the Galerkin procedure is performed by a Galerkin projection of (5.5) or (5.6) onto the space spanned by the polynomial basis. This is done by successively applying the inner product between the given equation and each of the basis functions, i.e. evaluating the expectation for each index. By use of multi-index $\mathbf{k}$ where $\mathbf{k} \leq N$ this leads to a new formulation of the system (5.1), namely

$$\begin{aligned}
\mathbb{E}[\frac{\partial}{\partial t} u^N(x,t,Z)\Phi_{\mathbf{k}}(Z)] &= \mathbb{E}[\mathcal{L}(u^N)\Phi_{\mathbf{k}}(Z)], \quad D \times T \times \mathbb{R}^d \\
\mathbb{E}[\mathcal{B}(u^N)\Phi_{\mathbf{k}}(Z)] &= 0, \quad \partial D \times T \times \mathbb{R}^d \\
\hat{u}_{\mathbf{k}} &= \hat{v}_{0,\mathbf{k}}, \quad \partial D \times [t=0] \times \mathbb{R}^d,
\end{aligned} \tag{5.7}$$

where the gPC projection coefficients $\hat{v}_{0,\mathbf{k}}$ of the initial condition more explicitly can be written as $\hat{v}_{0,\mathbf{k}} = \frac{\mathbb{E}[u_0 \Phi_{\mathbf{k}}(Z)]}{\gamma_{\mathbf{k}}}$. By evaluation of the expectations the system is reduced to a set of deterministic equations that usually are coupled. This means that the system is independent of the random parameters Z.
It follows from the gPC theory how to compute the statistics of the stochastic solution by computing the solution to the deterministic system (5.8). The mean and variance can for instance be computed as

$$\mu_{u(x,t)} \approx \mathbb{E}[u^N(x,t,Z)] = \hat{u}_{\mathbf{0}}(x,t), \quad \sigma_u^2 \approx \sigma_{\hat{u}}^2 = \sum_{|\mathbf{i}|=1}^{N} \gamma_{\mathbf{i}} \hat{u}_{\mathbf{i}}^2(x,t).$$

For further information see [19] and [12].

Chapter 6

Test problems

In this section two test problems are introduced. The first is a simple Ordinary Differential Equation (ODE) known as the (stochastic) Test Equation which will serve as a toy problem for testing convergence and efficiency of the methods. The second model problem is the partial differential equation (PDE) denoted Burgers' Equation which is more complex and involves a non-linear term. In this chapter the book [18] is used as a general reference when evaluating integrals.

6.1 Test Equation

In this section the ODE known as the Test Equation is introduced. The ODE looks like this with deterministic parameters

$$\frac{d}{dt}u(t) = -\lambda u(t), \quad u(0) = b.$$

With stochastic parameters the ODE looks like this

$$\frac{d}{dt}u(t,\omega) = -\alpha(\omega)u(t,\omega), \quad u(0,\omega) = \beta(\omega). \tag{6.1}$$

The analytical solution to the stochastic ODE (6.1) can be computed as follows

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}u(t,\omega) &= -\alpha(\omega)u(t,\omega) \Leftrightarrow \\
\frac{1}{u(t,\omega)}\,\mathrm{d}u(t,\omega) &= -\alpha(\omega)\mathrm{d}t \Leftrightarrow \\
\int_{u(0,\omega)}^{u(t,\omega)} \frac{1}{u(t,\omega)}\,\mathrm{d}u(t,\omega) &= \int_0^t -\alpha(\omega)\,\mathrm{d}t \Leftrightarrow \\
\ln\left(\frac{u(t,\omega)}{u(0,\omega)}\right) &= -\alpha(\omega)t \Leftrightarrow \\
\frac{u(t,\omega)}{u(0,\omega)} &= e^{-\alpha(\omega)t} \Leftrightarrow \\
u(t,\omega) &= u(0,\omega)e^{-\alpha(\omega)t}.
\end{aligned}$$

Since the initial condition is $u(0,\omega) = \beta(\omega)$ the analytical solution to the ODE (6.1) is

$$u(t,\omega) = \beta(\omega)e^{-\alpha(\omega)t}. \tag{6.2}$$

In the case where the distribution function $F_{\alpha\beta}(a,b) = P(\alpha \leq a, \beta \leq b)$ of α and β is known it is possible to compute the statistical moments of the solution (6.2). When the two parameters α and β are independent the distribution function simplifies to $F_{\alpha\beta}(a,b) = F_\alpha(a)F_\beta(b)$ which means that the expectation of the solution (6.2) can be computed as

$$\mathbb{E}[u(t,\omega)] = \mathbb{E}[\beta]\mathbb{E}[e^{-\alpha t}]. \tag{6.3}$$

The expectation is in general computed as

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} x f_X(x)\,\mathrm{d}x,$$

where $f_X(x)$ is the density function of the given stochastic variable. The variance of u for independent α and β can be found by computing

$$\mathbb{E}[u^2] - \mathbb{E}[u]^2 = \mathbb{E}[\beta^2]\mathbb{E}[(e^{-\alpha t})]^2 - (\mathbb{E}[\beta]\mathbb{E}[e^{-\alpha t}])^2.$$

This means that as long as α and β are independent the theory outlined here makes it possible to compute the statistics for the Test equation.

6.1.1 Normal distributed parameters

In this section the stochastic moments of the parameters are computed under the assumption that they are independent and Gaussian distributed, $\alpha(\omega), \beta(\omega) \sim$

$\mathcal{N}(\mu, \sigma^2)$. The density function for a Gaussian distribution is given as

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]. \tag{6.4}$$

The expectation of $e^{-\alpha t}$ can therefore be computed as follows

$$\begin{aligned}
\mathbb{E}\left[e^{-\alpha t}\right] &= \int_{-\infty}^{\infty}(e^{-xt})\left(\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(x-\mu)^2}\right)\mathrm{d}x \\
&= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(x^2+\mu^2-2x\mu)-xt}\mathrm{d}x \\
&= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(x^2+\mu^2-2x\mu+2\sigma^2 xt)}\mathrm{d}x \\
&= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}(x^2+\mu^2-2(\mu-\sigma^2 t)x)}\mathrm{d}x.
\end{aligned} \tag{6.5}$$

Now the variable $\hat{\mu} = \mu - \sigma^2 t$ is introduced. By using this variable it is possible to rewrite (6.5) in a convenient way when using that by the definition of $\hat{\mu}$ the following holds.

$$\begin{aligned}
(x-\hat{\mu})^2 &= x^2 + \hat{\mu}^2 - 2x\hat{\mu} \\
&= x^2 + (\mu^2 + (\sigma^2 t)^2 - 2\mu\sigma^2 t) - 2(\mu - \sigma^2 t)x \\
&= x^2 + \mu^2 - 2(\mu - \sigma^2 t)x + \sigma^4 t^2 - 2\mu\sigma^2 t.
\end{aligned} \tag{6.6}$$

It is noticed that the exponent of the exponential function equals (6.6) except for $\sigma^4 t^2 - 2\mu\sigma^2 t$. Hence (6.5) can be rewritten as

$$\begin{aligned}
\mathbb{E}[u] &= \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}((x-\hat{\mu})^2-(\sigma^4 t^2 - 2\mu\sigma^2 t))}\mathrm{d}x \\
&= e^{-\frac{1}{2\sigma^2}(-\sigma^4 t^2 + 2\mu\sigma^2 t))}\int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2\sigma^2}((x-\hat{\mu})^2)}\mathrm{d}x \\
&= e^{\frac{1}{2}\sigma^2 t^2 - \mu t}.
\end{aligned} \tag{6.7}$$

In the second equation it is recognized that the expression inside the integral is a density function for a normal distribution $\mathcal{N}(\hat{\mu}, \sigma^2)$ and the integral over a density function equals one. The variance of $e^{-\alpha t}$ can be computed as

$$\mathrm{var}(e^{-\alpha t}) = \mathbb{E}[(e^{-\alpha t})^2] - (\mathbb{E}[e^{-\alpha t}])^2,$$

and from (6.7) it follows that $(\mathbb{E}[e^{-\alpha t}])^2 = e^{\sigma^2 t^2 - 2\mu t}$. To compute $\mathbb{E}[(e^{-\alpha t})^2] = \mathbb{E}[e^{-2xt}]$ the same procedure as used to compute the expectation will lead to $\mathbb{E}[u^2] = e^{2\sigma^2 t^2 - 2\mu t}$. Hence the mean and variance are given as

$$\mathrm{var}(e^{-\alpha t}) = \mathbb{E}\left[e^{-\alpha t}\right] = e^{\frac{1}{2}\sigma^2 t^2 - \mu t}. \tag{6.8}$$

and

$$\mathbb{E}[(e^{-\alpha t})^2] - (\mathbb{E}[e^{-\alpha t}])^2 = e^{2\sigma^2 t^2 - 2\mu t} - e^{\sigma^2 t^2 - 2\mu t}. \tag{6.9}$$

The statistical moments of a Gaussian distributed β-parameter can also be computed. The mean of β is

$$\begin{aligned}\mathbb{E}[\beta] &= \int_{-\infty}^{\infty} x \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) \mathrm{d}x \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \int_{-\infty}^{\infty} x e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \mathrm{d}x \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \mu \sqrt{\frac{\pi}{\frac{1}{2\sigma^2}}} \\ &= \mu.\end{aligned}$$

The integral in the second equation has been evaluated according to [18] and the result is in good correspondence with the theory - the mean of a normal distributed parameter should be equal to the distribution mean. To compute the variance of β the following from [18] is used

$$\int_{-\infty}^{\infty} e^{-ax^2} \mathrm{d}x = \sqrt{\frac{\pi}{a}}. \tag{6.10}$$

When differentiating both sides of (6.10) with regard to a the following useful result is achieved

$$\frac{\mathrm{d}}{\mathrm{d}a} \int_{-\infty}^{\infty} e^{-ax^2} \mathrm{d}x = \int_{-\infty}^{\infty} -x^2 e^{-ax^2} \mathrm{d}x = \frac{\mathrm{d}}{\mathrm{d}a} \sqrt{\frac{\pi}{a}} = -\frac{1}{2a} \sqrt{\frac{\pi}{a}}.$$

This means that

$$\int_{-\infty}^{\infty} x^2 e^{-ax^2} \mathrm{d}x = \frac{1}{2a} \sqrt{\frac{\pi}{a}}. \tag{6.11}$$

In order to compute the variance of β the term $\mathbb{E}\left[\beta^2\right]$ has to be computed which can expressed as

$$\mathbb{E}\left[\beta^2\right] = \int_{-\infty}^{\infty} x^2 \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \right) \mathrm{d}x.$$

Now a new variable is introduced, namely $\hat{x} = x - \mu$ which implies $x = \hat{x} + \mu$. This also means that

$$x^2 = \mu^2 + \hat{x}^2 + 2\mu\hat{x}. \tag{6.12}$$

Furthermore $a = \frac{1}{2\sigma^2}$ is introduced which means that $\mathbb{E}\left[\beta^2\right]$ can be expressed as

$$\begin{aligned}
\mathbb{E}\left[\beta^2\right] &= \int_{-\infty}^{\infty} x^2 \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\right) \mathrm{d}x \\
&= \int_{-\infty}^{\infty} \left(\mu^2 + \hat{x}^2 + 2\mu\hat{x}\right) \frac{1}{\sqrt{2\pi\sigma^2}} e^{-a\hat{x}^2} \,\mathrm{d}\hat{x} \\
&= \frac{\mu^2 \int_{-\infty}^{\infty} e^{-a\hat{x}^2} \,\mathrm{d}\hat{x} + \int_{-\infty}^{\infty} \hat{x}^2 e^{-a\hat{x}^2} \,\mathrm{d}\hat{x} + 2\mu \int_{-\infty}^{\infty} \hat{x} e^{-a\hat{x}^2} \,\mathrm{d}\hat{x}}{\sqrt{2\pi\sigma^2}}
\end{aligned} \tag{6.13}$$

The last integral in (6.13) can be evaluated to be zero by substituting with $\hat{x} = x - \mu$ which gives

$$\begin{aligned}
\int_{-\infty}^{\infty} (x-\mu) e^{-a(x-\mu)^2} \,\mathrm{d}x &= \int_{-\infty}^{\infty} x e^{-a(x-\mu)^2} \,\mathrm{d}x - \mu \int_{-\infty}^{\infty} e^{-a(x-\mu)^2} \,\mathrm{d}x \\
&= \mu\sqrt{\frac{\pi}{a}} - \mu\sqrt{\frac{\pi}{a}} = 0.
\end{aligned}$$

The first of the two obtained integrals is the same as appeared in the computation of the mean and is known to be equal to $\mu\sqrt{2\pi\sigma^2}$ which is the same as $\mu\sqrt{\frac{\pi}{a}}$. The last integral is the same as in (6.10) and therefore evaluates to $\sqrt{\frac{\pi}{a}}$. Now it is possible to evaluate (6.13) since he first integral in (6.13) is recognized to be of the type (6.10), the second is the same as in (6.11) and the last integral evaluates to zero. Hence (6.13) becomes

$$\begin{aligned}
\mathbb{E}\left[\beta^2\right] &= \frac{1}{\sqrt{2\pi\sigma^2}} \left(\frac{1}{2a}\sqrt{\frac{\pi}{a}} + \mu^2\sqrt{\frac{\pi}{a}} + 2\mu \cdot 0\right) \\
&= \frac{1}{\sqrt{2\pi\sigma^2}} \left(\frac{2\sigma^2}{2}\sqrt{\pi 2\sigma^2} + \mu^2\sqrt{2\pi\sigma^2}\right) \\
&= \sigma^2 + \mu^2,
\end{aligned}$$

where the second equation results from substituting with $a = \frac{1}{2\sigma^2}$. This means that variance of β is $\mathrm{var}(\beta) = (\sigma^2 + \mu^2) - \mu^2$. The mean and variance of $\beta \sim \mathcal{N}(\mu_\beta, \sigma_\beta^2)$ is therefore

$$\begin{aligned}
\mathbb{E}[\beta] &= \mu, \\
\mathrm{var}(\beta) &= \sigma^2.
\end{aligned}$$

6.1.2 Uniform distributed parameters

The parameters can also be uniform distributed and in this section the statistical moments for this case are computed. The density function for the uniform distribution is

$$f_X(x) = \begin{cases} \frac{1}{b-a} & x \in (a,b), \\ 0 & \text{otherwise.} \end{cases} \tag{6.14}$$

First the statistics of $e^{-\alpha t}$ are computed and the expectation can be computed as

$$\begin{aligned} \mathbb{E}\left[e^{-\alpha t}\right] &= \int_a^b e^{-xt}\frac{1}{b-a}\,\mathrm{d}x \\ &= \frac{1}{b-a}\left(\frac{1}{-t}\right)\left[e^{-xt}\right]_a^b \\ &= -\frac{e^{-bt}-e^{-at}}{t(b-a)}. \end{aligned} \tag{6.15}$$

Now $\mathbb{E}\left[\left(e^{-\alpha t}\right)^2\right]$ is computed as

$$\begin{aligned} \mathbb{E}\left[\left(e^{-\alpha t}\right)^2\right] &= \int_a^b e^{-2xt}\frac{1}{b-a}\,\mathrm{d}x \\ &= \frac{1}{b-a}\left(\frac{1}{-2t}\right)\left[e^{-2xt}\right]_a^b \\ &= -\frac{e^{-2bt}-e^{-2at}}{2t(b-a)}. \end{aligned} \tag{6.16}$$

This means that the variance is

$$\begin{aligned} \mathbb{E}\left[\left(e^{-\alpha t}\right)^2\right] - \mathbb{E}\left[e^{-\alpha t}\right]^2 &= -\frac{e^{-2bt}-e^{-2at}}{2t(b-a)} - \left(-\frac{e^{-bt}-e^{-at}}{t(b-a)}\right)^2 \\ &= -\frac{e^{-2bt}-e^{-2at}}{2t(b-a)} - \frac{\left(e^{-bt}-e^{-at}\right)^2}{t^2(b-a)^2}. \end{aligned} \tag{6.17}$$

The expectation and variance of β when β is uniformly distributed can also be computed and is

$$\begin{aligned} \mathbb{E}[\beta] &= \int_a^b x\frac{1}{b-a}\,\mathrm{d}x = \frac{1}{b-a}\left[\frac{1}{2}x^2\right]_a^b \\ &= \frac{1}{2}\frac{b^2-a^2}{b-a} = \frac{1}{2}\frac{(b+a)(b-a)}{b-a} \\ &= \frac{1}{2}(b+a). \end{aligned} \tag{6.18}$$

In order to compute the variance of β the term $\mathbb{E}(\beta^2)$ has to be computed and it results in

$$\begin{aligned}
\mathbb{E}\left[\beta^2\right] &= \int_a^b x^2 \frac{1}{b-a}\,\mathrm{d}x \\
&= \frac{1}{b-a}\left[\frac{1}{3}x^3\right]_a^b \\
&= \frac{1}{3}\frac{b^3-a^3}{b-a} \\
&= \frac{1}{3}\frac{(b^2+a^2+ab)(b-a)}{b-a} \\
&= \frac{1}{3}(b^2+a^2+ab). \qquad (6.19)
\end{aligned}$$

In this computation it has been used that

$$\begin{aligned}
(b^2+a^2)(b-a) &= b^3+ba^2-ab^2-a^3 \\
&= b^3-a^3-ab(b-a)
\end{aligned}$$

which means that

$$b^3-a^3 = (b^2+a^2)(b-a)+ab(b-a) = (b^2+a^2+ab)(b-a).$$

The variance of β can now be computed as

$$\begin{aligned}
\mathrm{var}(\beta) &= \frac{1}{3}(b^2+a^2+ab) - \left(\frac{1}{2}(b+a)\right)^2 \\
&= \frac{1}{3}(b^2+a^2+ab) - \frac{1}{4}(b^2+a^2+2ab) \\
&= \frac{1}{12}(b^2+a^2) - \frac{1}{6}(ab) \\
&= \frac{1}{12}(b-a)^2. \qquad (6.20)
\end{aligned}$$

As expected the computation of the mean and variance of the uniformly distributed β-parameter again yields $\mathbb{E}[\beta] = \mu_\beta$ and $\mathrm{var}(\beta) = \sigma_\beta^2$ where μ_β and σ_β are the distribution parameters for μ_β which are defined as $\mu_\beta = \frac{1}{2}(b+a)$ and $\sigma_\beta^2 = \frac{1}{12}(b-a)^2$.

6.1.3 Multivariate Test Equation

The theory outlined earlier makes it possible for computing the statistics for a multivariate Test equation for independent α and β and either Gaussian or uniform distributed. The procedure is the same for all the combinations of distributions of the two variables and therefore only two cases will be introduced.

6.1.3.1 Gaussian distributed α and β

When α and β are independent and Gaussian distributed the mean can be computed by use of the terms computed in chapter 6.1.1. The mean can therefore be computed as

$$\begin{aligned}\mathbb{E}[u] &= \mathbb{E}[\beta]\mathbb{E}[e^{-\alpha t}] \\ &= \mu_\beta e^{\frac{1}{2}\sigma^2 t^2 - \mu t}.\end{aligned}$$

The terms for computing the variance can also be reused to computed the variance of the multivariate ODE. The terms are

$$\begin{aligned}\mathbb{E}[(e^{-\alpha t})^2] &= e^{2\sigma^2 t^2 - 2\mu t} \\ (\mathbb{E}[e^{-\alpha t}])^2 &= e^{\sigma^2 t^2 - 2\mu t} \\ \mathbb{E}[\beta^2] &= \sigma^2 + \mu_\beta^2 \\ \mathbb{E}[\beta]^2 &= \mu_\beta^2.\end{aligned}$$

This means that the variance can be computed as

$$\mathrm{var}(u) = (\sigma^2 + \mu_\beta^2)e^{2\sigma^2 t^2 - 2\mu t} - \mu_\beta^2 e^{\sigma^2 t^2 - 2\mu t}. \tag{6.21}$$

6.1.3.2 Gaussian distributed α and uniformly distributed β

Now it is assumed that α is Gaussian distributed and β is uniformly distributed and that they are mutually independent. The mean and variance can be computed in the same manner as when both of them where Gaussian distributed. The mean is computed as

$$\begin{aligned}\mathbb{E}[u] &= \mathbb{E}[\beta]\mathbb{E}[e^{-\alpha t}] \\ &= \mu_\beta e^{\frac{1}{2}\sigma^2 t^2 - \mu t}.\end{aligned}$$

Again the terms used for computing the variance has already been computed to be

$$\begin{aligned}
\mathbb{E}[(e^{-\alpha t})^2] &= e^{2\sigma^2 t^2 - 2\mu t} \\
(\mathbb{E}[e^{-\alpha t}])^2 &= e^{\sigma^2 t^2 - 2\mu t} \\
\mathbb{E}[\beta^2] &= \frac{1}{3}(b^2 + a^2 + ab) \\
\mathbb{E}[\beta]^2 &= (\frac{1}{2}(b + a))^2 = \mu_\beta^2.
\end{aligned}$$

This means that the variance can be computed as

$$\operatorname{var}(u) = \frac{1}{3}(b^2 + a^2 + ab)e^{2\sigma^2 t^2 - 2\mu t} - \mu_\beta^2 e^{\sigma^2 t^2 - 2\mu t}.$$

In the same way other combinations of the independently distributed variables α and β could be computed.

6.2 Burgers' Equation

The viscous Burgers' equation is a classic example of a non-linear PDE and it is defined as

$$\begin{cases} u_t + uu_x = \nu u_{xx}, \quad x \in [-1, 1], \\ u(-1) = 1, \\ u(1) = -1, \end{cases} \tag{6.22}$$

where ν is the viscosity and u is the solution field. Burgers' equation is an interesting PDE to study since it contains both time and space derivatives, a non-linear term and has several interesting properties. One property that is interesting from an UQ point of view is that the PDE is supersensitive towards disturbances in the boundary condition [19]. This feature will be investigated in the numerical experiments later on.
The solution to Burgers' equation is plotted in figure 6.1 for $t = 0$ and it is seen that it is more or less constant 1 until it reaches a transition layer and shifts to be close to -1.

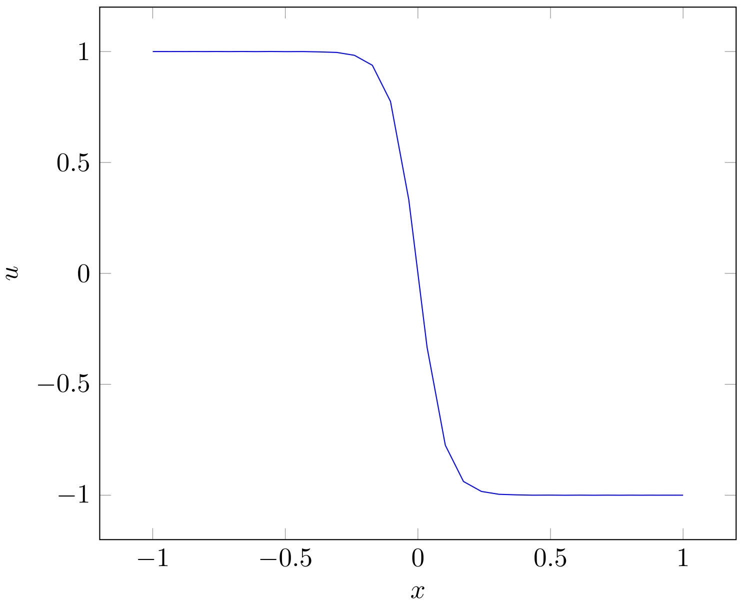

Figure 6.1: The deterministic solution for Burgers' equation for $t = 0$.

The focus of this thesis is on Uncertainty Quantification (UQ) and therefore a stochastic Burgers' equation is introduced. The stochastic Burgers' equation used in the 1D test cases is formulated as

$$\begin{cases} u_t + uu_x = \nu u_{xx}, & x \in [-1, 1], \\ u(-1) = 1 + \delta(Z), \\ u(1) = -1, \end{cases} \tag{6.23}$$

where $\delta(Z)$ is a random perturbation of the boundary. This means that the left boundary condition has been perturbed by additive noise. The effects of having a stochastic ν and right boundary have also been investigated but these experiments are conducted later on.
The effects of a stochastic boundary condition is clear when the solution is plotted for the deterministic system with the left boundary condition $u(-1) = 1$ and the system with $u(-1) = 1.01$.

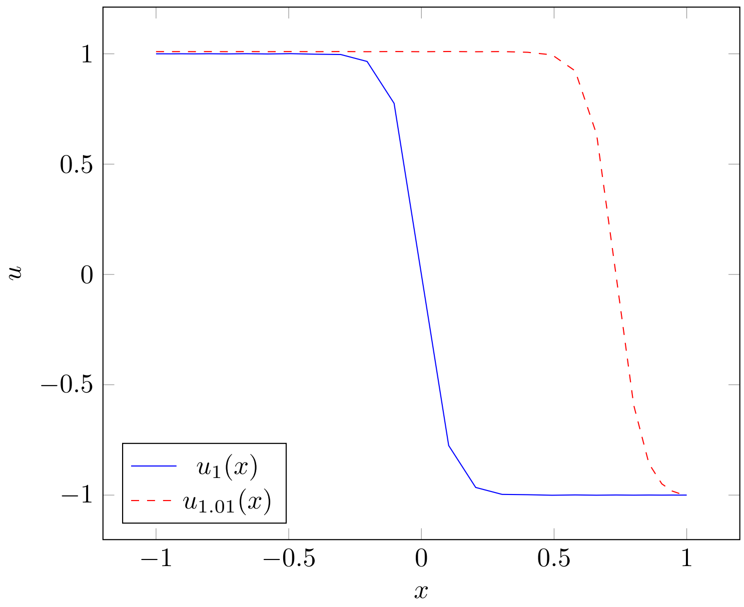

Figure 6.2: The deterministic solution for Burgers' equation with boundary conditions $u(-1) = 1$ and $u(-1) = 1.01$.

Figure 6.2 illustrates that the solution is very dependent on how the boundary conditions are formulated since the transition layer moves quite a lot even though the left BC is perturbed by only 1 %.

Chapter 7

Test of UQ methods on the Test Equation

In this chapter the three UQ methods will be used to solve the Test Equation and test their qualities and drawbacks. The implementations of the methods will be outlined here and all the code can be found in appendix B.

7.1 Monte Carlo Sampling

In this section the results from the Monte Carlo Sampling will be examined. The approach can be outlined as

Algorithm 4 Pseudocode for Monte Carlo Sampling applied to the Test Equation

```
1: Compute $N$ realizations of the stochastic variable $\alpha$.
2: Solve the deterministic system to obtain the $N$ deterministic solutions $u$.
3: Compute mean and variance.
```

The realizations of the random variables are computed by use of the Matlab function `RandVar`. The code is given below

```
if strcmp(DistType,'normal')
     X = mu + sigma.*randn(row,column,M);
elseif strcmp(DistType,'uniform')
     X = a + (b-a).*rand(row,column,M);
end
```

The `mu` and `sigma` refers to the parameters for the normal distribution and `a` and `b` refers to the parameters of the uniform distribution.
The deterministic solution has been computed by use of the Runge-Kutta method ERK, which is implemented in `ERK.m`. This implementation and the implementation of the statistics can be seen in appendix B.

7.1.1 Gaussian distributed α-parameter

The first case to be investigated is when the initial condition is assumed to be deterministic, $\beta(\omega) \equiv 1$ and $\alpha \sim \mathcal{N}(\mu, \sigma)$. In the following μ refers to the analytical mean, i.e. $\mathbb{E}[u(t,\omega)]$ and $\bar{u}$ refers to the estimated mean. Furthermore u^i refers to the i'th deterministic solution and σ_α is the standard deviation of the stochastic α-parameter.

The deterministic solutions varies a lot when α is changed which is demonstrated in the following plot where $M = 10$ deterministic solutions are plotted together with the exact mean in figure 7.1.

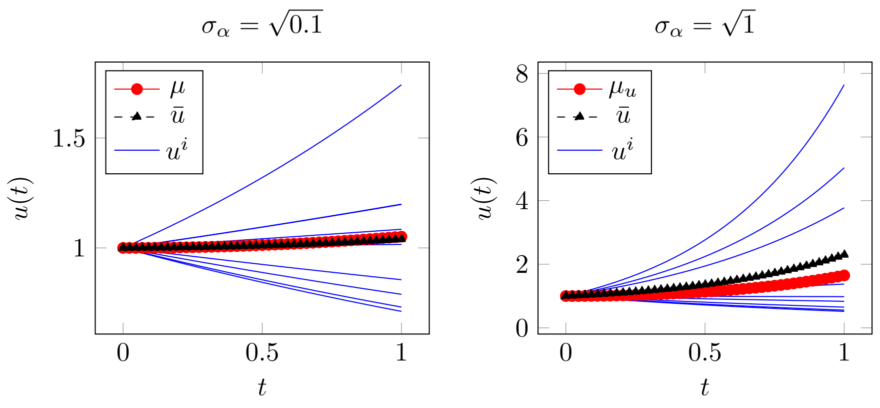

Figure 7.1: Exact mean, estimated mean and 10 solutions generated by using α distributed with variance 0.1 and 1.

It is seen that the crude estimate of the mean based on only 10 samples does not provide a reliable estimate of the solution. In figure 7.2 it is seen how an increasing number of samples, M, leads to a better estimate of the mean and variance. The estimated mean based on 100 samples is denoted $\bar{u}_{100}$ and the estimated mean based on 1000 samples is denoted $\bar{u}_{1000}$. Equivalently $\bar{s}_{100}$ and $\bar{s}_{100}$ denotes the estimated variances for 100 and 1000 samples, respectively.

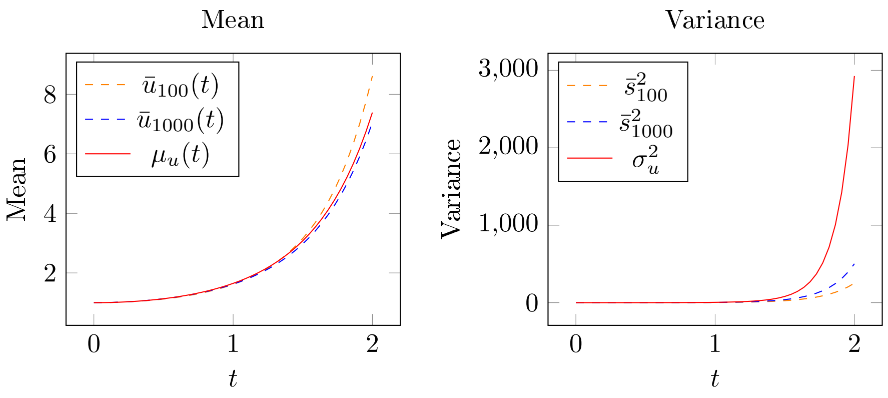

Figure 7.2: Estimated means and variances for $M = [100, 1000]$.

In figure 7.2 it is seen that the variance grows rapidly with time due to the exponential nature of the analytical variance that is computed to be (6.9). The estimations of the variance are not very good at time $t = 2$ but this was expected since $M = 1000$ samples only gives crude estimations of the mean and variance with high uncertainty in the results but it is much better estimates than those obtained for 10 samples.
In order to get results with high accuracy a much larger number of samples, M, must be used. In figure 7.3 it is seen how the error between the estimated and the exact statistics behaves for increasing M.

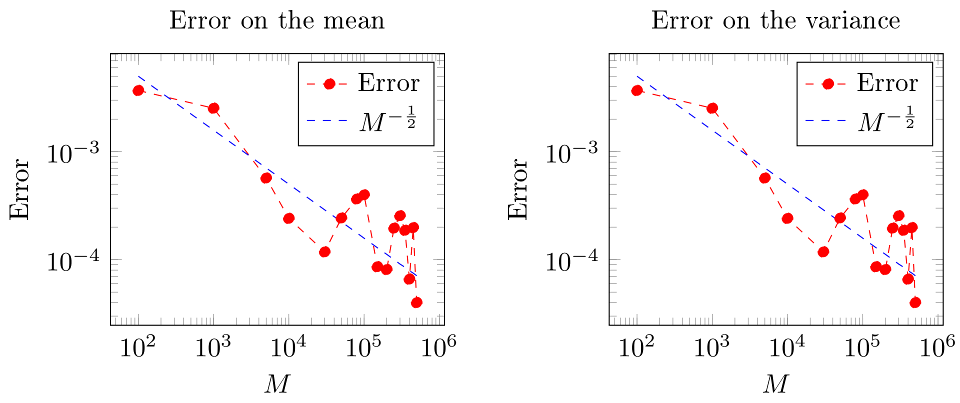

Figure 7.3: Error on the mean and variance for increasing M.

In figure 7.3 it is seen that the error on the mean and variance follows the overall tendency of $\frac{1}{\sqrt{M}}$ which was expected. There are some fluctuations in the result but this is nothing out of the ordinary and there is good correspondence between theory and practice.
The focus of uncertainty quantification in this thesis is to investigate the uncertainty of the output of a mathematical model. To illustrate the uncertainty of the stochastic Test equation the mean is plotted together with plus/minus the standard deviation, i.e. $\bar{\mu} \pm \sigma$, in figure 7.4.

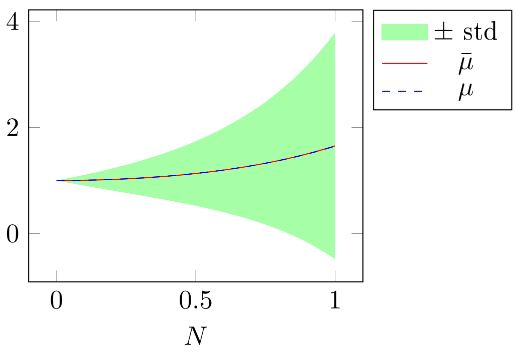

Figure 7.4: The estimated mean and the standard deviation.

7.1.2 Uniformly distributed α-parameter

Now it is assumed that α is uniformly distributed, $\alpha \sim \mathcal{U}(-1,1)$, and again the initial condition is assumed to be deterministic, $\beta(\omega) \equiv 1$. This means that the analytical expectation and variance can be written according to (6.16) and (6.17).
The same overall tendencies and behaviours can be observed when using an uniformly distributed α as when the parameter is normal distributed.
Figure 7.5 contains a plot of the mean plus/minus the standard deviation and it is seen that the standard deviation is much smaller for this test case than for the case where α was normal distributed.

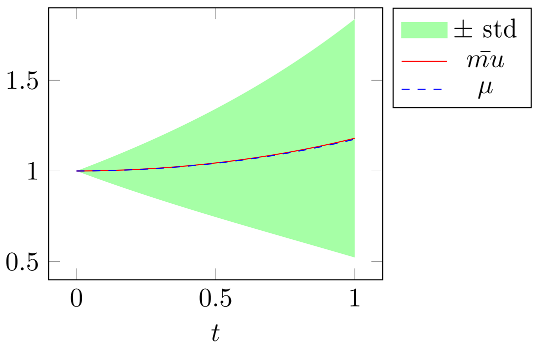

Figure 7.5: The estimated mean, the analytical mean and the standard deviation.

In figure 7.6 the convergence of the MC method is plotted and it behaves as expected. The tendency in the convergence is similar to the theoretical $\mathcal{O}(M^{-\frac{1}{2}})$ even though there is large variations in the accuracy of the obtained estimates.

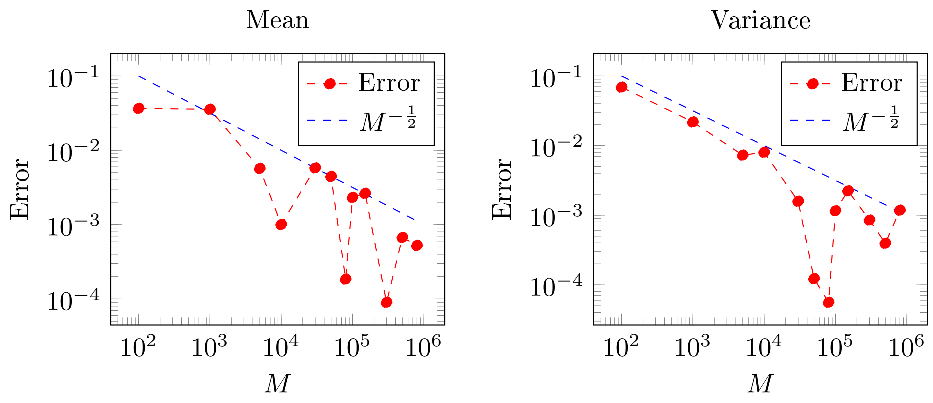

Figure 7.6: Error on the mean and variance for increasing M.

7.2 Stochastic Collocation Method

From the theory it is known that the Stochastic Collocation Method should be much more efficient than the Monte Carlo sampling for the 1D test case. Therefore the Stochastic Collocation Method (SCM) will be used on the same problem as before in order to check whether it delivers good results and if it is more efficient.
The application of the SCM for the Test Equation when α is the only stochastic parameter is outlined in algorithm 5.

Algorithm 5 Pseudocode for SCM applied to the Test Equation

```
1: Compute $N+1$ quadrature nodes $[z_i]_{i=0}^N$ and weights $[w_i]_{i=0}^N$.
2: Compute $[\alpha_i]_{i=0}^N$ from $[z_i]_{i=0}^N$
3: Solve the deterministic system to obtain the $N+1$ deterministic solutions
   $[u_i]_{i=0}^N$.
4: Compute the mean $\mathbb{E}[u] = \sum_{i=0}^N u_i w_i$
5: Compute the variance $\mathrm{var}(u) = \sum_{i=0}^N \left(u_i - \mathbb{E}[u]\right)^2 w_i$.
```

Which quadrature nodes and weights to be used depends on the distribution of α as outlined in the theory. The deterministic solutions can be found by use of an appropriate solver in time - e.g. a Runge-Kutta method. The mean and variance are computed as stated in the pseudocode and other statistics could be computed as well.

7.2.1 Implementation

In this section some of the implementations used for SCM for Test Equation are introduced. The computation of the quadrature nodes and weights are implemented as

```
if strcmp(DistType,'normal')
    % Gaussian Distribution: Hermite Gauss Quadrature.

    [z,zW] = HermiteQuadN(zN); % Computing nodes and
        weights

    z = mu+sigma*z; % Computing the alpha-parameter
    n = 0:zN-1;
 elseif strcmp(DistType,'uniform')
    % Uniform Distribution: Legendre Gauss Quadrature.

    [z,zW] = legendrequad(zN); % Computing nodes and
        weights
    zW = 1/2*zW;
end
```

where `zN` refers to the number of nodes in `alpha`. The implementation of the quadrature functions can be found in appendix B.
The deterministic solutions are computed by using the Runge-Kutta method `ERK.m`. The deterministic solution is computed by

```
for ti = 1:length(tspan)-1
    U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,z);
end
```

where `rhsSCMtest` is the right-hand-side function implemented as

```
function un = rhsSCMtest(t,U,alpha)

un = alpha.*U;
```

The statistics and the exact solutions have been computed as described in the theory and the implementation can be found in appendix B as well as the implementation of a test script.

7.2.2 Gaussian distributed α-parameter

In this section the α-parameter is chosen to be standard normal distributed, $\alpha \sim \mathcal{N}(0, 1)$, which means that the Gauss Hermite quadrature rules have been applied to compute the appropriate quadrature nodes and weights. The errors in mean and variance have been computed as a function of N, where N is the number of quadrature nodes and thereby the number of deterministic solutions. These errors can be seen in figure 7.7, where it is seen that the error of the estimated mean is not improving when more than 9 quadrature nodes are used. The error of the estimated variance requires more quadrature nodes to reach its minimum at about 10^{-8} and reaches this minimum when 15 quadrature nodes are used. This means that the SCM reaches its best approximation of the mean with 9 quadrature nodes and of the variance with 15 quadrature nodes at time $t = 1$. This is remarkably faster than when using the Monte Carlo Sampling.

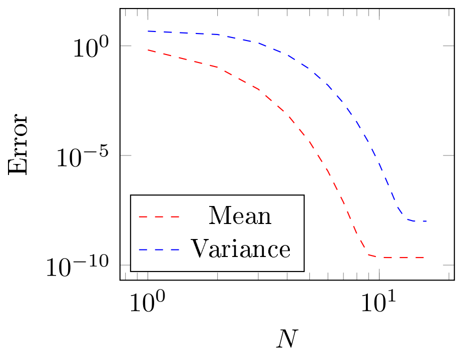

Figure 7.7: Error on the mean and variance for increasing N, at time $t = 1$.

To indicate the uncertainty in the results the mean can be plotted together with the standard deviation as was seen in figure 7.4 and figure 7.5. The computed mean and standard deviation are plotted in figure 7.8 as along with a plot of the deterministic solutions.

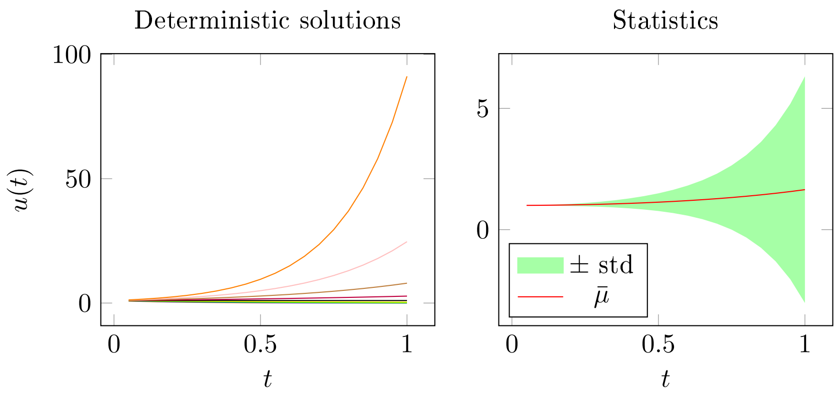

Figure 7.8: Left: The deterministic solutions. Right: The mean estimated and standard deviation.

From figure 7.8 it is seen that for time $t = 1$ there is a lot of uncertainty in the result. This uncertainty would be increased in time since the variance increases exponentially in time.

7.2.3 Uniformly distributed α-parameter

In this section it will be investigated how good the SCM is to estimate the statistics when an uniform distributed α-parameter, $\alpha \sim \mathcal{U}(-1, 1)$, is used.
In figure 7.9 the errors in mean and variance are plotted as function of the total number of nodes M and it seen that also for an uniformly distributed α-parameter the estimates will stabilize - this time at about $8.4 \cdot 10^{-13}$ and $2.8 \cdot 10^{-12}$ for the mean and variance, respectively.

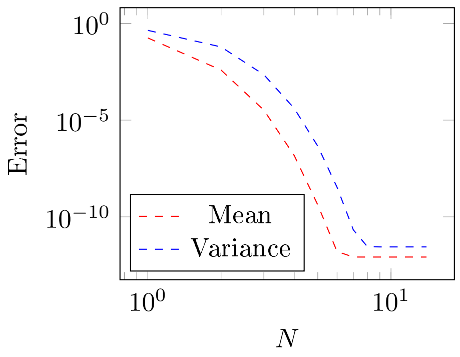

Figure 7.9: Error on the mean and variance for increasing N.

From figure 7.9 it is seen that for 7 quadrature nodes the error in the mean is minimized and equivalently for the variance for 8 quadrature nodes. This differs from the results obtained for a normal distributed α-parameter and it seems like the choice of distribution effects both the statistics and the efficiency of SCM. In figure 7.10 9 deterministic solutions and the corresponding mean and variance are plotted.

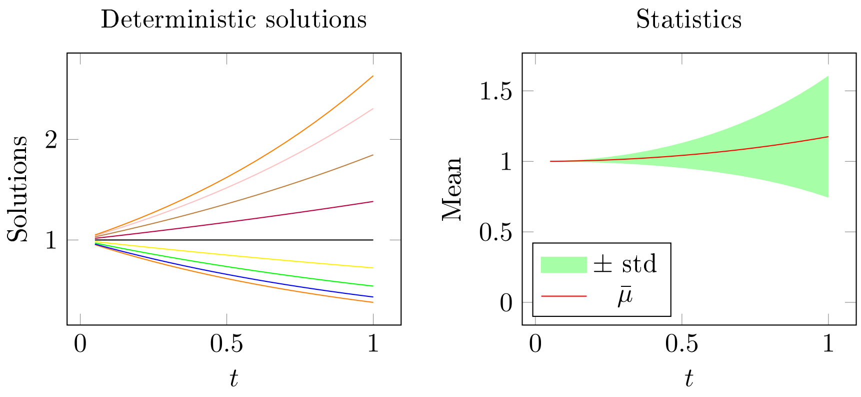

Figure 7.10: Left: The deterministic solutions. Right: The estimated mean and standard deviation.

Hence the SCM is more efficient than the MC and obtains quite good results for even for a small number of deterministic solutions.

7.3 Stochastic Galerkin Method

It is interesting to see how well an intrusive gPC Galerkin Method works compared to the MC method and the SCM. From the theory it is clear that for complicated problems this method can be very cumbersome to apply, since it requires a lot of analytical derivation and extra implementations. However for the simple Test equation this does not pose problem.
The stochastic parameter α can be represented by the following univariate gPC projection.

$$\alpha^N(Z) = \sum_{i=0}^{N} a_i \Phi_i(Z), \tag{7.1}$$

where the coefficients a_i are to be determined. Similarly can the initial condition β be represented by a gPC projection.

$$\beta^N(Z) = \sum_{i=0}^{N} b_i \Phi_i(Z),$$

where b_i are the coefficients. The gPC approximation, v_N, of the solution u is formulated as

$$v^N(t, Z) = \sum_{j=0}^{N} \hat{v}_j \Phi_j(Z),$$

It is assumed that that α is stochastic and β is deterministic. Applying the gPC Galerkin procedure leads to

$$\mathbb{E}\left[\frac{dv^N}{dt}\Phi_k\right] = \mathbb{E}\left[-\alpha^N v^N \Phi_k\right], \quad k = 0, \dots, N. \tag{7.2}$$

By substituting the expressions for v_N and α_N into (7.2) and evaluating the expectations the following result is obtained.

$$\frac{d\hat{v}_N}{dt} = -\frac{1}{\gamma_k}\sum_{i=0}^{N}\sum_{j=0}^{N} a_i \hat{v}_j e_{ijk}, \quad k = 0, \dots, N, \tag{7.3}$$

where $\gamma_k = \mathbb{E}[\Phi_k^2]$ and $e_{ijk} = \mathbb{E}[\Phi_i(Z)\Phi_j(Z)\Phi_k(Z)]$ with $0 \le i, j, k \le N$. The orthogonality of the gPC basis polynomials is utilized when computing the left-hand side of (7.2). Usually γ_k is computable or at hand when choosing basis functions Φ but an expression for $e_{i,j,k}$ is not always at hand which means that quadrature rules at times are used for approximating this constant.
The system (7.3) can be represented by use of vector notation,

$$\frac{d\mathbf{v}}{dt}(t) = \mathbf{A}^T\mathbf{v}, \quad \mathbf{v}(0) = \mathbf{b},$$

where $\mathbf{b} = (b_0, \ldots, b_N)^T$ is the deterministic initial condition and the matrix $\mathbf{A}$ is defined by $\mathbf{A} = (A_{ij})_{\forall j,k}$, where A_{ij} is defined by

$$A_{ij} = -\frac{1}{\gamma_k} \sum_{j=0}^{N} a_i e_{ijk}.$$

This means that $\mathbf{A}$ is a $(N+1) \times (N+1)$ matrix. The system (7.3) is a system of $N+1$ ordinary differential equations for the unknown coefficients $\{\hat{v}_k\}$ with initial condition $\hat{v}_k(0) = b_k$. A pseudocode for the implementation of the gPC Galerkin Method can be seen in algorithm 6.

Algorithm 6 Pseudocode for SGM applied to the stochastic Test Equation

```
1: Compute the initial condition $b$.
2: Compute the coefficient vector $a$.
3: Compute the matrix $\mathbf{A}$.
4: Use a deterministic solver to solve system in time and hereby obtaining a
   solution vector $U$.
5: The mean is contained in the first column of the solution $U$.
6: Compute variance $\mathrm{var}(u) = \sum_{i=1}^{N} \frac{1}{\gamma_i} U_i^2 w_i$.
```

Here U_i refers to the i'th column of the solution U.

7.3.1 Implementation

The computation of the initial condition and the coefficient vector follows the theory described previously and the implementation can be seen in appendix B. The computation of the matrix $\mathbf{A}$ is implemented as

```
A = zeros(N+1,N+1);
for j = 0:N
    for k=0:N
        [e,gamma] = TestGalPar(j,k,N);
        A(j+1,k+1) = -1/gamma*(a'*e);
    end
end
```

where `TestGalPar` is implemented as

```
function [e,gamma] = TestGalPar(j,k,N)

```

```
% Pre-allocation
e = zeros(N+1,1);

% Computation of a vector containing the e-values.
for i = 0:N
    s = (i+j+k)/2;

    if s<i || s<j || s<k || mod((i+j+k),2)~= 0
        e(i+1) = 0;
    else
        e(i+1) = factorial(i)*factorial(j)*factorial(k)
           ...
                /(factorial(s-i)*factorial(s-j)*
                    factorial(s-k));
    end
end

% Computation of the normalization factor gamma.
gamma = factorial(k);
```

The deterministic system is solved by use of the implemented function `ERK`.

7.3.2 Gaussian distributed α-parameter

Now the α-parameter is assumed to be Gaussian distributed, $\alpha \sim \mathcal{N}(\mu, \sigma)$, which implies that the gPC basis, $\{\Phi_k\}_{k=0}^N$, is chosen to be Hermite polynomials, $\{H_k(Z)\}_{k=0}^N$, and $Z \sim \mathcal{N}(0,1)$. Since α can be represented as $\alpha = \mu + \sigma Z$ the gPC representation (7.1) can be expressed as

$$\alpha^N(Z) = \sum_{i=0}^N a_i H_i(Z),$$

where

$$\begin{cases} a_0 = \mu \\ a_1 = \sigma \\ a_i = 0 \quad i \geq 2 \end{cases} \tag{7.4}$$

In this case α^N is an exact representation of α for $N \geq 1$ which is not always the case. Furthermore v^N can be represented as

$$v^N(t, Z) = \sum_{i=0}^N \hat{v}_i H_i(Z).$$

By using Hermite polynomials the constants in (7.3) can be expressed as

$$\begin{cases} \gamma_k = k! & k \geq k \\ e_{ijk} = \frac{i!j!k!}{(s-i)!(s-j)!(s-k)!} & s \geq i, j, k \text{ and } 2s = i + j + k \text{ is even.} \end{cases} \tag{7.5}$$

The convergence of the estimated mean and variance is illustrated by plotting the errors as functions of N as seen in figure 7.11.

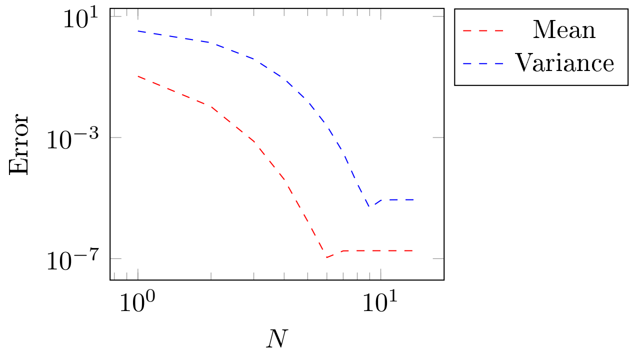

Figure 7.11: Error of the estimated mean and variance for increasing N.

It is seen that for the SCM 6 and 10 basis polynomials are needed for obtaining the minimum error of the estimated mean and variance respectively. The number of polynomials used also corresponds more or less to the number of deterministic solutions that have to be computed. This means that the SGM results in slightly less deterministic solutions to be computed than the SCM. It is however worth to note that for this test case, the rhs of the SCM is simpler and faster to evaluate than the rhs for the SGM.

7.3.3 Uniformly distributed α-parameter

Now the α parameter is assumed to be uniformly distributed, $\alpha \sim \mathcal{U}(a, b)$, with $a = -1$ and $b = 1$. This means that another the gPC basis, $\{\Phi_k\}_{k=0}^N$, has to be used, namely Legendre polynomials, $\{L_k(Z)\}_{k=0}^N$ with $Z \sim \mathcal{U}(a, b)$. The approximation of α can be expressed by an expansion with Legendre polynomials, $L(Z)$, as

$$\alpha^N(Z) = \sum_{i=0}^{N} a_i L_i(Z),$$

where

$$\begin{cases} a_0 = \frac{b+a}{2}, \\ a_1 = \frac{b-a}{2}, \\ a_i = 0 \quad i \geq 2. \end{cases} \tag{7.6}$$

Again α^N is an exact representation of α for $N \geq 1$. The representation of v^N can with Legendre polynomials be expressed as

$$v^N(t, Z) = \sum_{i=0}^{N} \hat{v}_i L_i(Z).$$

By using Legendre polynomials the constants in (7.3) can be expressed as $\gamma_k = \frac{2}{2k+1}$ and e_{ijk} is approximated by Gauss Legendre quadrature.
The convergence plot can be seen in figure 7.12.

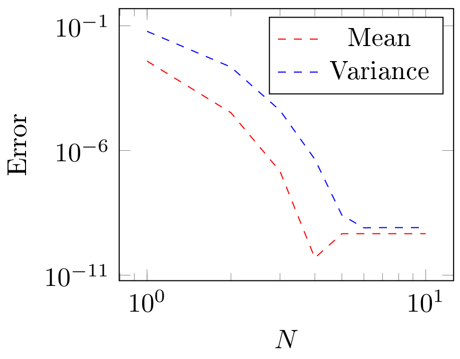

Figure 7.12: Error on the mean and variance for increasing N.

It is seen that 5 and 6 polynomials are needed for obtaining the minimum error of the estimated mean and variance respectively. Again this is slightly less than for the SCM but with this simple test case the difference in efficiency is not that important since both are very fast and easily evaluated. The SGM does however require a little more analytical work and some more implementation.

7.4 Conclusion

In this section it has been established that it is possible to obtain good estimates of the statistics with all three methods. Furthermore the Monte Carlo method has a convergence rate which fits the theoretical $M^{-\frac{1}{2}}$ and the SCM and SGM shows spectral convergence. The numerical tests conducted in this section demonstrates why the spectral methods are an attractive tool to use in the context of UQ.

Chapter 8

Burgers' Equation

Burgers' equation has been described previously and with a stochastic left boundary condition it can be formulated as (6.23). In this chapter the results from the numerical tests with the stochastic Collocation method and the stochastic gPC Galerkin method will be presented. In the following it will be assumed that $\delta(Z) \sim U(-1,1)$ and thus the polynomials used in the gPC Galerkin method will be Legendre polynomials and the collocation points and weights will be computed from a Gauss Legendre quadrature.

8.1 Stochastic gPC Galerkin method

In order to apply the Galerkin method a one dimensional gPC expansion is formulated as

$$v^N(x,t,Z) = \sum_{j=0}^{N} \hat{v}_j(x,t)\Phi_j(Z). \tag{8.1}$$

This expansion is substituted into (6.23) and the Galerkin procedure is applied which leads to

$$\mathbb{E}\left[\frac{\partial v^N}{\partial t}\Phi_k\right] + \mathbb{E}\left[v^N\frac{\partial v^N}{\partial x}\Phi_k\right] = \nu\mathbb{E}\left[\frac{\partial^2 v^N}{\partial x^2}\Phi_k\right], \quad k = 0,\ldots,N. \tag{8.2}$$

By substituting with the expression (8.1) for v_N the orthogonality of the basis functions can be applied. This means that the first term in the left-hand-side (lhs) of (8.2) can be rewriting as

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial v^N}{\partial t}\Phi_k\right] &= \mathbb{E}\left[\sum_{j=0}^{N}\frac{\partial \hat{v}_j}{\partial t}\Phi_j\Phi_k\right] \\
&= \sum_{j=0}^{N}\frac{\partial \hat{v}_j}{\partial t}\mathbb{E}\left[\Phi_j\Phi_k\right] \\
&= \sum_{j=0}^{N}\frac{\partial \hat{v}_j}{\partial t}\gamma_k\delta_{j,k} \\
&= \frac{\partial \hat{v}_k}{\partial t}\gamma_k,
\end{aligned} \tag{8.3}$$

where $\delta_{j,k}$ is the Kronecker delta function and $\gamma_k = \mathbb{E}\left[\Phi_k^2\right]$. The non-linear term in the lhs can be rewriting as

$$\begin{aligned}
\mathbb{E}\left[v^N\frac{\partial v^N}{\partial x}\Phi_k\right] &= \mathbb{E}\left[\sum_{i=0}^{N}\hat{v}_i\Phi_i\sum_{j=0}^{N}\frac{\partial \hat{v}_j}{\partial x}\Phi_j\Phi_k\right] \\
&= \mathbb{E}\left[\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{v}_i\frac{\partial \hat{v}_j}{\partial x}\Phi_i\Phi_j\Phi_k\right] \\
&= \sum_{j=0}^{N}\sum_{i=0}^{N}\hat{v}_i\frac{\partial \hat{v}_j}{\partial x}\mathbb{E}\left[\Phi_i\Phi_j\Phi_k\right] \\
&= \sum_{i=0}^{N}\sum_{j=0}^{N}\hat{v}_i\frac{\partial \hat{v}_j}{\partial x}e_{i,j,k},
\end{aligned} \tag{8.4}$$

where $e_{i,j,k} = \mathbb{E}\left[\Phi_j\Phi_i\Phi_k\right]$. The rhs of (8.2) can be rewriting as

$$\begin{aligned}
\nu\mathbb{E}\left[\frac{\partial^2 v^N}{\partial x^2}\Phi_k\right] &= \nu\mathbb{E}\left[\sum_{j=0}^{N}\frac{\partial^2 \hat{v}_j}{\partial x^2}\Phi_j\Phi_k\right] \\
&= \nu\sum_{j=0}^{N}\frac{\partial^2 \hat{v}_j}{\partial x^2}\mathbb{E}\left[\Phi_j\Phi_k\right] \\
&= \nu\sum_{j=0}^{N}\frac{\partial^2 \hat{v}_j}{\partial x^2}\gamma_k\delta_{j,k} \\
&= \nu\frac{\partial^2 \hat{v}_k}{\partial x^2}\gamma_k
\end{aligned} \tag{8.5}$$

This means that the system can be defined as

$$\frac{\partial \hat{v}_k}{\partial t} + \frac{1}{\gamma_k}\sum_{i=0}^{N}\sum_{j=0}^{N}\hat{v}_i\frac{\partial \hat{v}_j}{\partial x}e_{i,j,k} = \nu\frac{\partial^2 \hat{v}_k}{\partial x^2}, \quad k = 0,\ldots,N. \tag{8.6}$$

8.1.1 Numerical approximations

This section contains a description of how to approximate differentiation in space and time numerically. In space there will be used central difference schemes and in time an explicit four step Runge-Kutta method will be applied. The central approximation for differentiation in space is given as

$$\frac{\partial}{\partial x}u(x_i,t_j) = \frac{u(x_{i+1},t_j) - u(x_{i-1},t_j)}{2\delta_x},$$

where δ_x is the distance between the points in the space grid. The second order derivative is estimated by the following 3-point central difference approximation

$$\frac{\partial^2}{\partial x^2}u(x_i,t_j) = \frac{u(x_{i+1},t_j) + u(x_{i-1},t_j) - 2u(x_i,t_j)}{\delta_x^2}.$$

In both central approximations it has been assumed that the space grid points are equidistant. If this is not the case the $2\delta_x$ and δ_x^2 should be replaced with $|x_i - x_{i-1}| + |x_{i+1} - x_i|$ and $|x_i - x_{i-1}||x_{i+1} - x_i|$, respectively.

These central difference approximations leads to the following stencils

$$A_{dx} = \frac{1}{2\delta_x}\begin{bmatrix} -1 & 0 & 1 & 0 & & & \\ 0 & -1 & 0 & 1 & 0 & & \\ & & \ddots & & \ddots & & \\ & & 0 & -1 & 0 & 1 \end{bmatrix} \tag{8.7}$$

and

$$A_{dx^2} = \frac{1}{\delta_x^2}\begin{bmatrix} 1 & -2 & 1 & 0 & & \\ 0 & 1 & -2 & 1 & 0 & \\ & & \ddots & \ddots & \ddots & \\ & & 0 & 1 & -2 & 1 \end{bmatrix} \tag{8.8}$$

There is $N_x + 1$ space points and $N + 1$ polynomials in the expansion. In order to cope with the multiple sums in (8.6) a Kronecker product can be used to obtain the following matrices

$$\mathbf{A}_k = \begin{bmatrix} e_{0,0,k}A_s & e_{0,1,k}A_s & \cdots & e_{0,N,k}A_s \\ e_{1,0,k}A_s & e_{1,1,k}A_s & \cdots & e_{1,N,k}A_s \\ \vdots & & \ddots & \vdots \\ e_{N,0,k}A_s & e_{N,1,k}A_s & \cdots & e_{N,N,k}A_s \end{bmatrix} \quad \text{and} \quad \mathbf{B}_k = \frac{1}{\delta_x^2}\begin{bmatrix} -2 & 1 & 0 & & \\ 1 & -2 & 1 & 0 & \\ & \ddots & \ddots & \ddots & \\ & & & & 1 \\ & & & 1 & -2 \end{bmatrix},$$

where

$$A_s = \frac{1}{2\delta_x}\begin{bmatrix} 0 & 1 & 0 & & \\ -1 & 0 & 1 & 0 & \\ & \ddots & & \ddots & \\ & & & 0 & 1 \\ & & & -1 & 0 \end{bmatrix}$$

By introducing these matrices the system (8.6) can be written as

$$\frac{\partial \bar{v}_k}{\partial t} = \nu \mathbf{B}_k \bar{v} - \frac{1}{\gamma_k}\bar{v}\mathbf{A}_k\bar{V} + \mathbf{g}_k, \quad k = 0, \ldots, N. \tag{8.9}$$

where $\bar{V}$ and $\bar{v}$ are all the interior points of v and $\mathbf{g}_k$ is a vector added to the solution to apply the boundary condition and is defined as

$$\mathbf{g}_k = \begin{bmatrix} \frac{1}{\delta_x^2}v_k^0 + \frac{1}{\gamma_k}\frac{1}{2\delta_x}\sum_{i=0}^N\sum_{j=0}^N v_i^0 v_j^0 e_{i,j,k} \\ 0 \\ \vdots \\ 0 \\ \frac{1}{\delta_x^2}v_k^{N_x} - \frac{1}{\gamma_k}\frac{1}{2\delta_x}\sum_{i=0}^N\sum_{j=0}^N v_i^{N_x} v_j^{N_x} e_{i,j,k} \end{bmatrix},$$

where v_k^0 is the left boundary point of the k'th expansion and $v_k^{N_x}$ is the right boundary point of the k'th expansion. The matrix $\bar{V}$ and the vector $\bar{v}$ can be represented as

$$\bar{V} = \begin{bmatrix} v_{1,0} & \ldots & v_{1,N} & 0 & \\ 0 & v_{2,0} & \ldots & v_{2,N} & 0 \\ & & \ddots & \ddots & \\ \ldots & 0 & v_{2,0} & \ldots & v_{2,N} \end{bmatrix} \quad \text{and} \quad \bar{v} = \begin{bmatrix} v_{1,0} \\ v_{2,0} \\ \vdots \\ v_{N_x-1,0} \\ v_{1,1} \\ \vdots \\ v_{N_x-1,1} \\ v_{1,2} \\ \vdots \\ v_{N_x-1,N} \end{bmatrix}.$$

The computation of the IC is outlined in the next section about the implementation. To solve the problem the implemented ERK-method is used and post-processing consists of computing the interesting statistics from the computed solution.
The SGM for Burgers equation is outlined in the pseudocode in algorithm 7.

Algorithm 7 Pseudocode for SGM applied to the stochastic Burgers Equation

```
1: Compute the points in space $x$.
2: Compute the initial condition.
3: Solve the system in time (reach steady state).
4: Compute the mean $\mathbb{E}[u] = U_0$
5: Compute the variance $\mathrm{var}(u) = \sum_{i=1}^{N} \gamma_i (U_i)^2$.
```

8.1.2 Implementation

All the implementations for SGM used on Burgers' equation can be found in appendix and in this section some of the the implementation are introduced. First the implementation of the initial condition is outlined and it yields

```
% Mean and standard deviation.
b_mean = (1.1-1)/2; std = sqrt(1/12*(1.1-1)^2);

% Boundary conditions
b1 = 1+b_mean; b2 = -1;

v=-x;
v(1)=b1; v(xL)=b2;

% Initial condition for all N
U1 = zeros((xL-2)*(N+1),1);
U1(1:xL-2)=v(2:xL-1);
```

The initial condition is established by computing a zero-vector $U1$ of length $(N+1)\cdot(N_x-2)$ which corresponds to having a space vector containing all the interior points for each of the $N+1$ polynomials.
This vector is modified by replacing the first $N_x - 2$ elements with the interior space points with reversed sign.
A test script have been implementation and can be seen in appendix B. This implementation relies heavily on the rhs-function `rhsGalBurg` which get the current solution and the matrices described in the previous section as input and

then computes (8.9). The implementation is seen in appendix B.
The computations outlined in this section have been based on computing the solution in each time step for the interior points. To compute the values in the end points for each of the N terms in the gPC expansion the following implementation is applied

```
U = zeros(size(v));
for iB = 0:N
    U(xL*iB+1) = b1;
    U(xL*(iB+1)) = b2;
    U(xL*iB+2:xL*(iB+1)-1) = U1((xL-2)*iB+1:(xL-2)*(iB+1)
        );
end
```

where `U1` refers to the computed solution in the interior points and `b1` and `b2` refers to the values in the left and right boundary, respectively.

8.1.3 Numerical experiments

When solving Burgers' equation a steady state solution has to be obtained - this holds for both the deterministic and stochastic case. As an illustration the deterministic Burgers' equation is solved with initial condition $-\mathbf{x}$ for different times.

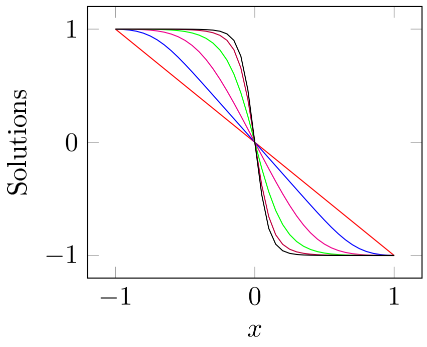

Figure 8.1: The computed solutions for different bounds on the integration in time.

It is seen that the further out in time the deterministic system is solved the closer it comes to the steady state solution which was plotted in figure 6.1.
The SGM has been used in a setting with $N_x = 41$ space points and with

$N + 1 = 4$ terms in the gPC expansion. The estimated mean and variance have been plotted in figure 8.2. These statistics have been plotted together with upper and lower bounds on the deterministic solutions and the standard deviation has been plotted around the mean to indicate the uncertainty in the mean.

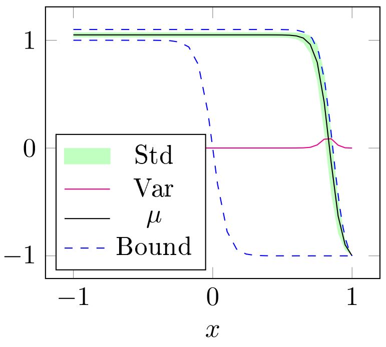

Figure 8.2: The mean, variance and bounds for the computed solution.

The statistics have naturally been computed such that steady state is reached.

8.2 Stochastic Collocation Method

The stochastic Collocation method is applied to the stochastic Burgers' equation. As mentioned earlier the collocation points and weights used for are computed by use of Gauss Legendre quadrature and the used deterministic solver is the ERK-method in combination with a deterministic spectral Collocation method for solving the spatial part of the PDE.
As described earlier a deterministic Collocation method can involve computing differential matrices in order to handle the differentiation in space. To handle this a set of deterministic collocation points has to be chosen and the Vandermonde matrices $\mathcal{V}$ and $\mathcal{V}_x$ has to be computed.
This setup means that there will be two sets of collocation points involved - the stochastic Gauss Legendre collocation points z and the deterministic Gauss Lobatto collocation points, x.
The polynomials used in the Vandermonde matrices $\mathcal{V}$ and $\mathcal{V}_x$ are Legendre

polynomials which means that the matrices can be expressed as

$$V = \begin{bmatrix} L_1(x_1) & L_1(x_2) & \cdots & L_1(x_N) \\ L_2(x_1) & L_2(x_2) & \cdots & L_2(x_N) \\ \vdots & \vdots & \ddots & \vdots \\ L_N(x_1) & L_N(x_2) & \cdots & L_N(x_N) \end{bmatrix}$$

and

$$V_x = \begin{bmatrix} L_1(x_1)' & L_1(x_2)' & \cdots & L_1(x_N)' \\ L_2(x_1)' & L_2(x_2)' & \cdots & L_2(x_N)' \\ \vdots & \vdots & \ddots & \vdots \\ L_N(x_1)' & L_N(x_2)' & \cdots & L_N(x_N)' \end{bmatrix}.$$

From the theory it is known that the differentiation matrix can be expressed as $\mathcal{D} = \mathcal{V}_x \mathcal{V}^{-1}$. This means that a discrete representation of Burgers equation (6.22) can be expressed as

$$\frac{\partial u}{\partial t} = \nu \mathcal{D}^2 u - u \mathcal{D} u, \quad x \in [-1, 1], \tag{8.10}$$

The rhs function either consists of a loop that does the matrix vector multiplication for each $i = 1, \ldots, N_z$, that is

$$\nu \mathcal{D}^2 u_i - u_i \mathcal{D} u_i, \quad i = 1, \ldots, N_z$$

or by constructing two big, sparse matrices $\mathcal{D}_{kron}$ and $\mathcal{D}^2_{kron}$ that are the computed by taking the Kronecker product between an identity matrix of appropriate size and $\mathcal{D}$ and $\mathcal{D}^2$, respectively. This leads to a rhs function that does the following computation

$$\nu \mathcal{D}^2_{kron} u - u \mathcal{D}_{kron} u$$

The post-processing of the deterministic solutions follows from the theory and the statistics are computed as seen in Algorithm 8 that outlines SCM For Burgers' equation.

Algorithm 8 Pseudocode for SCM applied to the stochastic Burgers' equation

```
1: Compute the collocation points $x$.
2: Compute the parameters for the rhs, i.e. $\mathcal{D}$ and $\mathcal{D}^2$.
3: Compute the initial condition $IC$ and quadrature weights $w$.
4: Use a deterministic solver to solve the system in time (reach steady state).
5: Compute the mean $\mathbb{E}[u] = \sum_{i=0}^{N} u_i \rho w_i$
6: Compute the variance $\mathrm{var}(u) = \sum_{i=0}^{N} \left(u_i - \mathbb{E}[u]\right)^2 \rho w_i$.
```

All the code for SCM for Burgers' equation can be seen in Appendix B and the most important parts are outlined in the following section.

8.2.1 Implementations

The collocation points are computed by use of quadrature and the implementation can be seen in Appendix B. The implementation of the Vandermonde-matrices and the differentiation matrices are also included in the appendix. The initial condition is implemented as

The stochastic collocation points, z, are used for computing an initial condition. The implementation can be found in Appendix B. The deterministic solutions are computed in the MATLAB function `BurgDetSolv` which is implemented as

```
function [U,t,time] = BurgDetSolv(U,dt,param,ErrorBar,
    MaxIter)


t = zeros(MaxIter,1);

iter = 0;
diff = 1;

tic
while diff > ErrorBar &&  iter <= MaxIter
     UTemp = U;
     U = ERK(t(iter+1),U,@rhsColBurg,dt,param);
     t(iter+2) = t(iter+1)+dt;
     diff = max(abs(UTemp-U));
     iter = iter+1;
end
time = toc;
```

It is seen that the deterministic solutions are computed by using a while-loop where the function `ERK` is called in each iteration.
The while-loop is used to ensure that a state "close" to the steady state is reached. The assumption is that the less change in the computed solutions the closer are the solutions to steady state and the difference between two consecutive solutions is therefore used as a measure for how close to steady state the current solution is.
The boundary conditions are applied in the rhs-function by setting the end-point values for each solutions to zero. The implementation of the rhs is seen in appendix B as well as the rest of the implementations.

8.2.2 Numerical experiments

As mentioned in the implementation it is important to reach steady state and a user-specified bound has been introduced to ensure that the maximum difference between the elements in two consecutive solutions is smaller than this.
In the numerical test with SCM a bound of 10^{-6} was used as well as $N_x = 46$ deterministic collocation points and $N_z = 10$ stochastic collocation points. This resulted in a series of deterministic solutions plotted in figure 8.3.

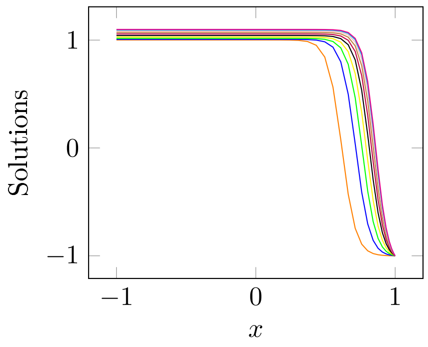

Figure 8.3: The 10 deterministic solutions used to compute the statistics for Burgers Equation.

In figure 8.4 the mean, variance and standard deviation have been plotted together with upper and lower bounds for the solution. The bounds for the solution corresponds to the deterministic solution for the minimum and maximum values of the stochastic left boundary, hence $\delta(Z) = 0$ and $\delta(Z) = 0.1$.

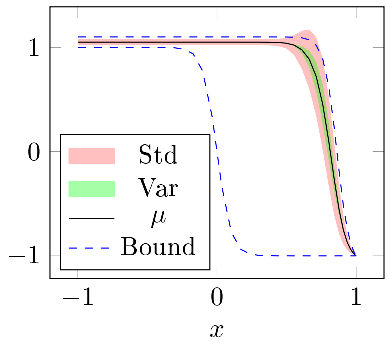

Figure 8.4: The mean, variance and bounds for the computed solution.

It is seen from figure 8.4 that the solution of Burgers' equation is very sensitive towards disturbances in the boundary since the transition layer moves quite a lot. The upper and lower bounds gives a good impression of the span of the solution when the boundary is disturbed with between 0 and 10 percent noise. The steady-state estimation of the mean is also quite different from the unperturbed solution (the lower bound) and this illustrates that even though errors on the estimations are small and the estimated variance is relatively small as well, the computed statistics does not give an especially good impression of the noise-free solution.
It is worth to note that this is another dimension to the UQ analysis. There are errors on the computed estimates and when the estimates are accurate the variance might be very large and thereby make the solutions very uncertain. Furthermore if UQ is conducted to get an impression of the unperturbed deterministic solution it should be noted that the computed statistics might deviate a lot from this solution.

Chapter 9

Discussion: Choice of method

The methods used in this thesis is the Monte Carlo Sampling, the stochastic Galerkin method and the stochastic Collocataion method and these will in the following be compared.
From the outlined theory and the numerical experiments it is clear that the Monte Carlo sampling has relatively slow convergence compared to the two other methods for low dimensional problems and it can be cumbersome to use when the deterministic system is very time consuming to solve.
It is however an interesting method since it can be used as a reference when exploring new problems and since its convergence rate is independent of the number of stochastic variables. As mentioned in the theory the computational effort of the stochastic gPC Galerkin method and especially of the stochastic collocation method grows rapidly as the number of stochastic variables in the system is increased. This means that at a certain point the Monte Carlo Sampling will become more efficient than the two spectral methods.
The difference in accuracy is also one of the distinctions between the Galerkin method and the Collocation method. The Galerkin method is based on minimising the residue of the stochastic governing equations. The collocation method is based on a different approach namely to introduce a set of nodes and ensure that the error in these nodes are zero. When using this approach there will be problems with aliasing errors which means that the collocation method in general is less accurate than the Galerkin method. This is especially true for

multidimensional spaces. In the univariate case the aliasing error can be kept at the same order as the finite order Galerkin method [19].
This means that based on an accuracy perspective the Galerkin method is to be preferred over the Collocation method but there is another important aspect namely the implementation.
The Galerkin method implies derivation of a gPC Galerkin system where the equations for the expansion coefficients are coupled. This usually means that new implementations have to be made in order to cope with the coupled systems. Furthermore the derivation of the gPC equations can be very tricky and in some cases it is not possible to make the derivations [19].
This is very different from the Collocation method where the deterministic systems solved for each node are decoupled and could be solved with parallel programming [19]. The Collocation method is relatively easy to implement as long as there exists decent deterministic solvers for the problem at hand. When this criteria is fulfilled the Collocation method can be boiled down to choosing the set of collocation nodes, solve the deterministic problem at each node and perform post-processing by applying either the interpolation approach or the discrete projection approach described earlier.
A consequence of this is that even though the original problem might be nonlinear or complex problem it is still relative straight forward to solve as long as a deterministic solver is at hand or can be derived. This means that the Collocation method might be less accurate than the Galerkin method but it is much easier to apply and for this reason the Collocation method is very popular.

The Galerkin method is the most accurate and involves the least number of equations in a multidimensional space. But the Collocation method is much easier to implement and the derivation of the gPC Galerkin equations can be very difficult and even impossible in some cases [19]. This means that the choice of method is not trivial since each method has its strengths and weaknesses and the choice depends a lot on the dimensionality and complexity of the problem.

The SCM is interesting because of the ease of implementation and if the curse of dimensionality could be reduced it would be a useful tool in high dimensions. Therefore the rest of the thesis will involve the SCM for multivariate problems.

Chapter 10

Literature Study

As introduced in earlier the stochastic Collocation method has some nice properties but it also has some drawbacks with regard to the amount of computational work. Due to this a literature study has been conducted with focus on how to optimize the use of SCM.
In section 10.1 a brief outline of the studied topics is given and in the following sections of this chapter some of the topics are introduced more thoroughly.

10.1 Brief Introduction to topics and articles

Some of the effects of the curse of dimensionality can be reduced by use of sparse grids instead of a full tensor product grid. The Smolyak grids are widely used in the field of UQ and they will be introduced later in the thesis. Some interesting work worth noting is the MatLab scripts and the documentation in [14] and [13], respectively. This work makes it possible to easily access implementation of different sparse grids and to compare them. Furthermore the work of John Burkardt [2] with regard to Matlab implementation of the Gauss Legendre sparse grid is also very useful as well as the work of Florian Heiss and Viktor Winschel [11] which contains implementation of Smolyak sparse grids. Interested readers can learn more about the topic in e.g. [3], [5], [16] and [19].

The article [5] claims that the common way of using the sparse grids might not be optimal and introduces another way of using the sparse grids such that the errors in the coefficients to the higher order polynomials are reduced. The new approach is denoted Sparse Pseudospectral Approximation Method (SPAM) and is outlined in section 10.2.

Two articles introducing an interesting approach to minimize the size of the systems to be solved and thereby avoiding the curse of dimensionality are [6] and [21]. In these articles an approach known from signal processing is applied in the field of Uncertainty Quantification.
The approach is called Compressive Sensing - or Compressive Sampling - and is based on constructing an approximation to the solution by use of an orthogonal projection. The main idea is that this orthogonal projection in some cases will form a sparse system which can be utilized to minimize the computational effort [6]. The approach is outlined in section 10.3.
A well-known topic from the statistical modelling is the analysis-of-variance (ANOVA) which also have been used in UQ. ANOVA is a method where the total variance is investigated by decomposing the original problem into subproblems and then computing the variance for each subproblem. It is a method often used adaptively in UQ to reduce the computational work as in introduced in [9] or [22]. The interested reader can find an overview of some relevant and recent articles and topics within ANOVA for uncertainty quantification in the introduction of [23] .

10.2 Sparse Pseudospectral Approximation Method

The method outlined here is described in the article [5] and it is an alternative way of utilizing the sparse grid procedure introduced earlier. The alternative procedure is introduced to avoid errors in the coefficients of the high-order polynomials which is a result of the usual application of the Smolyak method according to [5].
The reason why these errors emerge is according to [5] that the coupling between the nodal Lagrange representation and the modal polynomial representation in multiple dimensions - which is described for the one-dimensional case in chapter 2 - does not hold in the sparse settings.
In 10.2.1 some notation and concepts are introduced and the new approach is outlined in 10.2.2.

10.2.1 Introduction of notation and concepts

In this section some concepts and definitions from [5] are introduced. In the following a pseudospectral projection refers to a projection where the integrals are approximated by numerical integration rules, e.g. quadrature rules. In chapter 4 a truncated polynomial projection of a multivariate function f was introduced as

$$P_N f(s) = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} \bar{f}_{i_1,\ldots,i_d} \Phi_{i_1}(s_1) \cdots \Phi_{i_d}(s_d) = \sum_{|\mathbf{i}| \in I_{\mathbf{i}}} \bar{f}_{\mathbf{i}} \Phi_{\mathbf{i}}(s),$$

where $s = (s_1, \ldots, s_d)$ refers to a d-dimensional point, $I_{\mathbf{i}} = \{\mathbf{i} \,|\, \mathbf{i} \in \mathbb{N}^d, 1 \leq i_k \leq n_k, k = 1, 2, \ldots, d\}$ is the set of admissible multi-index and $\bar{f}_{\mathbf{i}}$ are the coefficients defined as [5]

$$\bar{f}_{\mathbf{i}} = \frac{1}{\gamma_{\mathbf{i}}} \int f(s) \Phi_{\mathbf{i}}(s) w(s) ds, \quad \forall |\mathbf{i}| \leq N.$$

The pseudospectral projection is defined equivalently but the coefficients are defined by use of numerical integration rules and yields

$$\begin{aligned} \hat{f}_{\mathbf{i}} &= \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} f(\lambda_{i_1}, \ldots, \lambda_{i_d}) \Phi_{i_1}(s_1) \cdots \Phi_{i_d}(s_d) \frac{1}{\gamma_{i_1}} \cdots \frac{1}{\gamma_{i_d}} \\ &= \sum_{|\mathbf{i}| \in I_{\mathbf{i}}} f(\lambda_{\mathbf{i}}) \Phi_{\mathbf{i}}(\lambda_{\mathbf{i}}) \frac{1}{\gamma_{\mathbf{i}}}. \end{aligned}$$

where $\{\lambda_{\mathbf{i}}\}$ refers to the chosen quadrature points. The article also refers to Lagrange interpolation and introduces that the correspondence between the nodal and modal representations introduced in chapter 2 can be expanded to the multidimensional case when using the tensor grid representations [5].

The approach of using the pseudospectral projection combined with the Smolyak sparse often consist of first making the truncation of the spectral projection and then compute the approximations of the coefficients by use of a sparse grid quadrature rule [5]. According to [5] this leads to errors in the coefficients because with sparse grid integration the truncated pseudospectral approximation is not equivalent to the Lagrange interpolating polynomial as was the case when using the tensor product approximation.

10.2.2 Overall approach

The idea of sparse pseudospectral approximation method (SPAM) is that since the sparse grid is introduced as a linear combination of tensor product quadrature rules the same can be applied to compute a linear combination of tensor product pseudospectral expansions [5]. This is the concept of the sparse pseudospectral approximation.
When introducing the Smolyak sparse grid a set of admissible indices I and a coefficient function $c(\mathbf{i})$ is introduced as well. The set I is defined as [5]

$$I = \{\mathbf{i} \in \mathbb{N}^d \,|\, l+1 \leq |\mathbf{i}| \leq l+d\},$$

where l is an integer called the level parameter. The coefficients can be defined as

$$c(\mathbf{i}) = (-1)^{l+d-|\mathbf{i}|} \left(\begin{array}{c} d-1 \\ l+d-|\mathbf{i}| \end{array} \right).$$

These are used in SPAM as well. The approximation of a multivariate function $f(s)$ can be expressed as

$$f(s) \approx \mathcal{A}_p(f) = \sum_{\mathbf{i}\in I} c(\mathbf{i}) \hat{f}_{\mathbf{i}}^T \Phi_{\mathbf{i}}(s).$$

where coefficients $\hat{f}_{\mathbf{i}}$ and the polynomials $\Phi_{\mathbf{i}}$ are defined as for the pseudospectral projection introduced in section 10.2.1. The set of polynomials $\{\Phi_{\mathbf{i}}(s)\}$ can be used to create a set of basis polynomials for $\mathcal{A}_p$ by defining

$$\mathcal{P} = \bigcup_{\mathbf{i}\in I} \{\Phi_{\mathbf{i}}(s)\}.$$

With this definition of the basis the $\mathcal{A}_p(f)$ can be written as

$$\mathcal{A}_p(f) = \sum_{\Phi_{\mathbf{i}}\in \mathcal{P}} \hat{f}_{\Phi} \Phi(s),$$

where the coefficients $\hat{f}_{\Phi}$ are formulated as

$$\hat{f}_{\Phi} = \sum_{\mathbf{i}\in I} c(\mathbf{i}) \hat{f}_{\mathbf{i},\mathbf{m}}.$$

Furthermore the coefficients $\hat{f}_{\mathbf{i},\mathbf{m}}$ are defined as

$$\hat{f}_{\mathbf{i},\mathbf{m}} = \begin{cases} \hat{f}_{\mathbf{i}} & \text{if } \Phi(s) = \Phi_{\mathbf{i}}(s) \text{ and } \mathbf{i} \in I \\ 0 & \text{otherwise.} \end{cases}$$

In [5] there is a theorem stating that this approximation is equivalent to the sparse grid interpolation approximation and there are numerical experiments which indicates that SPAM might be better than the usual pseudospectral projection approach when it comes to computing the coefficients for the high order polynomials

10.2.3 Further reading

For a more thorough introduction to SPAM and the theory behind it the interested readers are referred to the article [5]. The article also contains numerical tests that should illustrate the purpose and advantages of SPAM.

10.3 Compressive sampling: Non-adapted sparse approximation of PDES

As mentioned previously in the literature study the compressive sampling is an approach used in signal processing and recently it has been applied in the field of UQ. The introduction given here is mainly based on the articles [6] and [21]. Compressive sampling is an interesting and flexible approach that is based on representing the solution by an expansion and then utilize the sparsity of the expansion.
The main assumption is that the expansion is sparse in the sense that relatively few coefficients are significant such that an underdetermined system can represent the actual system accurately or even exactly. It is an approach that is well suited for the cases where not enough data is available and for high dimensional multivariate problems.
One of the very interesting aspects of the approach is that it is non-adapted in the sense that there is no tailoring of the sampling process and thereby no structuring of the nodes. This is an advantage since the structuring of the nodes is often what leads to a rapid growth in the total number of nodes when the dimensionality of the problem is increased.

10.3.1 Overall approach

The approach outlined here is based on approximation of the solution to a system of the type outlined in 5.1. The approach outlined here is an interpolating collocation type of method and is based on constructing a multivariate polynomial expansion that approximates the solution $u(x, t, Z)$. The truncated expansion can be expressed as

$$u_N = \sum_{|\mathbf{i}|\leq N} \hat{u}_{\mathbf{i}} \Phi_{\mathbf{i}}(\mathbf{Z}^m), \quad m = 1, \dots, M, \tag{10.1}$$

where $\mathbf{Z}$ refers to the set of multivariate samples introduced in chapter 4, $\Phi_{\mathbf{i}}(Z)$ are the multivariate polynomials also introduced in chapter 4 and the coefficients can be computed as

$$\hat{u}_{\mathbf{i}} = \frac{1}{\gamma_{\mathbf{i}}}\mathbb{E}[u\Phi_{\mathbf{i}}] = \frac{1}{\gamma_{\mathbf{i}}}\int u(z)\Phi_{\mathbf{i}}(z)dF_Z(z), \quad \forall|\mathbf{i}| \leq N.$$

by use of the multivariate expectation introduced previously.

Such a system can be expressed in vectorized form by introducing a Vandermonde like matrix $\mathbf{V} \in \mathbb{R}^{P\times M}$ where M is the number of samples and P is the cardinality of the space spanned by the orthogonal basis polynomials $\Phi_{\mathbf{i}}(Z)$. The matrix $\mathbf{V}$ consists of $\mathbf{V}_{n,m} = \Phi_{\mathbf{i}_n}(Z^m)$ for $n = 1,\ldots,P$ and $m = 1,\ldots,M$. This means that the vectorized representation is on the form

$$\mathbf{Vc} = \mathbf{u},$$

where $\mathbf{c} \in \mathbb{R}^P$ is a coefficient vector consisting of the coefficients $\hat{u}_{\mathbf{i}}$ and $\mathbf{u} \in \mathbb{R}^M$ is a vector containing the solutions $u(x,t,Z^m)$ for each of the M samples of the random variables.
The main idea in the compressive sampling approach is to solve an underdetermined system by use minimization to reduce the number of samples M. This means that the number of samples M is reduced to be far smaller than the cardinality P of the expansion, i.e. $M \ll P$. The minimization acts as a regularization in order to achieve a well-posed solution to the underdetermined system [6] and takes the form

$$\min \|\mathbf{c}\|_k \quad \text{subject to} \quad \mathbf{Vc} = \mathbf{u} \tag{10.2}$$

where $k = 0, 1$. The norms used for the minimizations can be formulated as

$$\|\mathbf{c}\|_0 = \#\{\mathbf{i} : \hat{u}_{\mathbf{i}} \neq 0\} \quad \text{and} \quad \|\mathbf{c}\|_1 = \sum_{|\mathbf{i}|\leq N} |\hat{u}_{\mathbf{i}}|$$

The minimization using $k = 0$ generally leads to a global minimum solution which is not unique and where the cost of a global search is exponential in P [6]. The ℓ_1-minimization is a relaxation of the ℓ_0-minimization ([6], [21]) and is usually the one pursued in practice.
Instead of using an interpolation strategy it can sometimes be useful to relax the constraints in the minimizations in the sense that some error is allowed

$$\begin{aligned} &\min \|\mathbf{c}\|_0 \quad &&\text{subject to} \quad &&\|\mathbf{Vc} - \mathbf{u}\|_2 \leq \delta, \\ &\min \|\mathbf{c}\|_1 \quad &&\text{subject to} \quad &&\|\mathbf{Vc} - \mathbf{u}\|_2 \leq \delta. \end{aligned}$$

This relaxation can be used for several reasons and one of them is that the truncated expansion (10.1) often is not an exact representation of the solution $u(x,t,Z)$ due to truncation errors and this is reflected in the minimizations by introducing a non-zero residual [6].

10.3.1.1 Weighting of the ℓ_1-minimization

It is worth to note that the ℓ_1-minimization is sometimes modified by a diagonal weight matrix $\mathbf{W}$ where the n'th diagonal element is the ℓ_2-norm of the n'th column of $\mathbf{V}$ [6]. This means that the modified ℓ_1-minimization can be formulated as

$$\min \|\mathbf{Wc}\|_1 \quad \text{subject to} \quad \mathbf{Vc} = \mathbf{u},$$

or in relaxed form

$$\min \|\mathbf{Wc}\|_1 \quad \text{subject to} \quad \|\mathbf{Vc} - \mathbf{u}\| \leq \delta.$$

The introduction of the weight $\mathbf{W}$ is to make sure that the optimization does not lead to a solution that is biased towards the $\mathbf{c}$ which are non-zero and whose corresponding columns in $\mathbf{V}$ have a large norm [6].

10.3.2 Recoverability

This is only meant as a brief introduction and therefore most of the theoretical background is not included here. Readers who are interested in the theoretical background and the stability of the relaxed minimizations can read more in [6]. One of the main results in [21] is the recoverability in the high dimensional case $d \gg 1$ and $d \geq N$ where N is the highest polynomial order in one dimension of the polynomial basis. In order to introduce the result the following definition of s-sparse is necessary [21].

Definition 10.1 (s-**sparse**) The vector $\mathbf{v}$ is called s-sparse if

$$\|\mathbf{v}\|_0 = \{\#m : v_m \neq 0 \leq s\},$$

which means that if there is no more than s non-zero elements in the vector $\mathbf{v}$ then $\mathbf{v}$ is s-sparse.

The article [21] is based on a Legendre polynomial expansion which means that the main result cited in theorem (10.2) is based on Legendre polynomials but it could be expanded to cover other polynomial bases as well [21].

Theorem 10.2 (Recoverability for ℓ_1-minimization) *For $d \geq N$ where d is the number of i.i.d. stochastic variables and N is the highest polynomial order in one dimension. Let $\boldsymbol{Z}$ be a sample of M independent draws from the uniform distribution $[-1,1]^d$. The number of samples M fulfill the boundary*

$$M \gtrsim 3^P s \log^3(s) log(P)$$

where P is the cardinality of the polynomial space $\mathbb{P}_N^d$ introduced in chapter 4 with $d \geq N$ and s is the sparsity level of a vector $\hat{\boldsymbol{c}} \in \mathbb{R}^M$. The Legendre polynomials are chosen as basis polynomials for a polynomial approximation $w \in \mathbb{P}_N^d$ of the type 10.1 which yields

$$w(Z) = \sum_{|\boldsymbol{i}| \leq N} \hat{u}_{\boldsymbol{i}} \Phi_{\boldsymbol{i}}(Z),$$

where $\mathbf{c}$ is the coefficient vector. This system is solved with regard to $\mathbf{c}$ by use of ℓ_1-minimization of the type (10.2) *and the data vector $\boldsymbol{u} = \boldsymbol{V}\hat{\mathbf{c}}$. For a universal constant γ the introduced settings leads to $\hat{\mathbf{c}}$ being recoverable with probability $1 - P^{-\gamma \log^3(s)}$ to within a factor of its best s-term approximation, i.e.*

$$\|\boldsymbol{c} - \hat{\boldsymbol{c}}\|_2 \lessapprox \frac{\delta_{s,1}(\hat{\boldsymbol{c}})}{\sqrt{s}} \tag{10.3}$$

where $\delta_{s,p}(\boldsymbol{v}) = \inf_{\|\boldsymbol{y}\|_0 \leq s} \|\boldsymbol{y} - \boldsymbol{v}\|_p$ is the the error of the best s-term approximation of a vector $\boldsymbol{v} \in \mathbb{R}^P$ in ℓ_p-norm.

The proof and theory behind theorem 10.2 can be found in [21].

10.3.3 Further reading

The outline of the method is given here and readers who are interested are recommended to read more in [6] and [21]. The interested reader can besides numerical results and a more thorough introduction of the theoretical background be introduced to Chebyshev Preconditioning of the ℓ_1-minimization in [21] and how to choose the truncation error tolerance δ in [6].
The choice of δ is naturally an interesting topic when using the relaxed minimization and the Chebyshev preconditioning yields some interesting results as well. For $d \geq P$ the preconditioning of the ℓ_1-minimization yields a number of points that scales with 2^d where the direct ℓ_1-minimization scales with 3^P. This means that for large dimensions d and moderate polynomial order P the direct minimization is the most efficient while in e.g. one dimension, $d = 1$, the preconditioned ℓ_1-minimization is the most efficient.

Chapter 11

Multivariate Collocation Method

In this chapter the theory for the stochastic collocation method will be expanded and the theoretical background for using a multivariate SCM is given. The introduced theory will be based on the interpolation approach of the collocation method.

11.1 Tensor Product Collocation

With the theory introduced so far the a straight forward approach for applying SCM in multiple dimensions is to apply the univariate expansions in each of the stochastic dimensions and then use tensor products to combine these to a multivariate SCM.
First a set of $d \geq 1$ independent variables is introduced as $Z = \{Z_1, \ldots, Z_d\}$ and for each variable a set of m_i collocation nodes is established as $\mathbf{Z}_i = Z_i^1, \ldots, Z_i^{m_i}$. By use of Kronecker products this yields a total set of nodes $\mathbf{Z}$ defined as

$$\mathbf{Z} = \mathbf{Z}_M = \mathbf{Z}_1 \times \cdots \times \mathbf{Z}_d$$

where M is the total number of nodes in the set and can be computed as $M = m_1 \times \cdots \times m_d$. The interpolating polynomial operator $\mathcal{I}_{m_i}$ in one variable is introduced as

$$\mathcal{I}_{m_i}[f] = \prod_{m_i} f(Z_i) \in \mathbb{P}(Z_i). \tag{11.1}$$

for $1 \leq i \leq d$ such that $\mathcal{I}_{m_i}[f]$ is an interpolating polynomial of order m_i in the i'th Z-variable equivalently to the univariate definition (5.2). This operator can be used to construct the multivariate interpolation operator by tensor product

$$\mathcal{I}_M = \mathcal{I}_{m_1} \otimes \cdots \otimes \mathcal{I}_{m_d}$$

By using tensor products to expand the dimensionality of the space some of the properties from one dimension is valid for the multivariate case as well. This is true for the convergence rate as well. For simplicity but without loss of generality it is here assumed that $m_1 = \cdots = m_d = m$. For the i'th dimension the one dimensional convergence rate is

$$(I - \mathcal{I}_{m_i})[f] \propto m^{-\alpha},$$

where α is constant that depends on the smoothness of the function f - the smoother a function the faster convergence, see e.g. [19]. This property still holds in the multivariate case which yields the convergence rate

$$(I - \mathcal{I}_M)[f] \propto m^{-\alpha}.$$

It is a nice feature that this property holds in the multivariate case but applying $M = m^d$ and expressing the convergence rate in terms of M results in a convergence rate of

$$(I - \mathcal{I}_M)[f] \propto M^{-\frac{\alpha}{d}},$$

This means that the convergence can be very slow if there is many stochastic variables d. Furthermore the total number of points grows exponentially with d, i.e. $M = m^d$ which means that the computational effort becomes very large for large d. This is due to the fact that for each node a deterministic system is solved, which could be very problematic in higher dimensions and is often referred to as the curse of dimensionality [19].

11.2 Multivariate expansions and statistics

The multivariate SCM can be formulated in much the same way as the univariate SCM as will be outlined here. The interpolation of a solution u to a multivariate PDE with d stochastic variables can be formulated as

$$\hat{u}(Z) \equiv \mathcal{I}_M(u) = (\mathcal{I}_{m_1} \otimes \cdots \otimes \mathcal{I}_{m_d})(u) = \sum_{j_1}^{m_1} \cdots \sum_{j_d}^{m_d} u(Z_1^{j_1}, \ldots, Z_d^{j_d})(h_1(Z_1^{j_1}) \ldots h_d(Z_d^{j_d})),$$

where h_i refers to the Lagrange polynomial in i'th stochastic dimension. This notation is widely adopted and is a bit more intuitive than the notation using multi-index. The multi-index and the corresponding single index does however allow a more compact notation, i.e.

$$\mathcal{I}_M(u) = \sum_{|\mathbf{i}| \leq N} u(\mathbf{Z}_{\mathbf{i}}) h_{\mathbf{i}}(Z),$$

where $h_{\mathbf{i}}(Z)$ is the multidimensional Lagrange Polynomial in Z. By the definition of the multivariate Lagrange polynomials they have the same qualities as the univariate polynomials with regard to attaining zeros and ones in the interpolation points.

The connection between the univariate SCM and the multivariate SCM is easier seen when using the single index since the multidimensional interpolation can be expressed as

$$\mathcal{I}_M(u) = \sum_{k=1}^{M} u(\mathbf{Z}_k) h_k(Z), \tag{11.2}$$

where the correspondence between the multi-index and the single index is as outlined in Table 4.1. The computation of the statistics, e.g. the expectation and the variance, can be conducted in much the same way as in the univariate case. The interested reader can read more about the application of the multidimensional cubature rules in [16], which leads to the following expression

$$Q(u) = (Q_1 \otimes \cdots \otimes Q_d)(u) = \sum_{j_1}^{m_1} \cdots \sum_{j_d}^{m_d} u(Z_1^{j_1}, \ldots, Z_d^{j_d})(\rho_1 w_1^{j_1} \cdots \rho_d w_d^{j_d}),$$

where Q_i refers to the 1D quadrature applied in the i'th dimension, ρ_i is the integration weight of the integral in the i'th dimension, i.e. the PDF of the i'th variable Z_i, and w_i^j refers to the j'th quadrature weight of the i'th stochastic variable.

By use of the tensor product the notation can be simplified when an appropriate index is used. The tensor product of the variables, $\mathbf{Z}$, has already been introduced and likewise the weights can be expressed by use of a tensor product. First the weights $\tilde{w}_i^j = \rho_i w_i^j$ are introduced to simplify notation and then the tensor product is computed as

$$\mathbf{W} = W_1 \otimes \cdots \otimes W_d$$

where $W_i = \{\tilde{w}_i^1, \ldots, \tilde{w}_i^d\}$. This means that the numerical integration by use of quadrature can be expressed as

$$Q(\tilde{u}) = \sum_{k=1}^{M} u(\mathbf{Z}_k)\mathbf{W}_k, \tag{11.3}$$

where the index used is the appropriately chosen single index. Now the mean and variance can be computed since they are based on integration of the interpolation $\tilde{u}$. The mean can be computed as

$$\mathbb{E}[\tilde{u}] = \sum_{k=1}^{M} u(\mathbf{Z}_k)\mathbf{W}_k,$$

and the variance of the multivariate interpolation can be computed as

$$\text{var}[\tilde{u}] = \sum_{k=1}^{M} \left(u(\mathbf{Z}_k) - \mathbb{E}[\tilde{u}]\right)^2 \mathbf{W}_k.$$

It is seen that there is a correspondence between computing the statistics of the univariate SCM and the multivariate SCM. The computations of the statistics have been implemented in MATLAB in the function `ColStat` which can be seen in appendix B.

11.3 Smolyak Sparse Grid Collocation

There exists many methods to reduce the impact of the curse of dimensionality. Here one approach will be outlined and in the literature study some other approaches will be introduced. The introduction given here is only an outline of the method and for further information see e.g. [19]

The approach introduced here is also based on tensor products but the trick is to only use a subset of the full tensor grid. This naturally leads to fewer nodes and hence less computational work since the deterministic system is solved fewer times.

The Smolyak's method is introduced in [5] as

$$Q_N = \sum_{\mathbf{i}\in I} c(\mathbf{i}) \cdot (Q_{i_1} \otimes \cdots \otimes Q_{i_d}),$$

where I is the set of admissible multi-indices and $c(\mathbf{i})$ is the corresponding coefficients. The set I is defined as [5]

$$I = \{\mathbf{i} \in \mathbb{N}^d \,|\, l+1 \leq |\mathbf{i}| \leq l+d\},$$

where l is an integer called the level parameter. The coefficients can be defined as

$$c(\mathbf{i}) = (-1)^{l+d-|\mathbf{i}|} \begin{pmatrix} d-1 \\ l+d-|\mathbf{i}| \end{pmatrix}.$$

By using the set I the appropriate quadrature nodes and weights can be chosen from the full tensor grid computed previously. Another definition is given in [19] where the Smolyak construction is expressed as

$$Q_N = \sum_{N-d+1 \le |\mathbf{i}| \le N} (-1)^{N-|\mathbf{i}|} \begin{pmatrix} d-1 \\ N-|\mathbf{i}| \end{pmatrix} \cdot (Q_{i_1} \otimes \cdots \otimes Q_{i_d}),$$

where the integer $N \geq d$ denotes the level of the construction. The Smolyak sparse grid can be computed by the following union of the subsets of the full tensor product

$$\mathbf{Z}_M = \bigcup_{N-d+1 \le |\mathbf{i}| \le N} (\mathbf{Z}_1^{i_1} \times \cdots \times \mathbf{Z}_d^{i_d}). \tag{11.4}$$

In contrast to applying the full tensor product there is no general expression or formula to compute the total number of nodes M in terms of d and N [19].

Different approaches for choosing I and $c(\mathbf{i})$ can be applied depending on the given multivariate problem. Furthermore adaptive methods might be very useful in some cases. But in general the Smolyak Sparse Grids can be described as a linear combination of tensor products that seek to ensure that the number of nodes does not grow too much [5].
The approach outlined here can be even more effective when the chosen nodes are nested which means that the one-dimensional nodal sets satisfies [19]

$$\mathbf{Z}_i^j \subset \mathbf{Z}_i^k \quad j \leq k. \tag{11.5}$$

More information about nested nodal sets can be found in e.g. [5] and [19].

11.3.1 Clenshaw-Curtis: Nested sparse grid

The Clenshaw-Curtis nodes are a popular choice when computing a nested sparse grid. The nodes are computed by finding the extrema of the Chebyshev polynomials which are a sub-class of the Jacobi polynomials. The zeros of the Chebyshev polynomials can be computed as [19]

$$Z_i^j = -\cos\left(\frac{\pi(j-1)}{m_i^k - 1}\right), \quad j = 1, \ldots, m_i^k, \tag{11.6}$$

where $1 \leq i \leq d$, k is referred to as the level and m_i^k often is chosen as $m_i^k = 2^{k-1} + 1$. The grid that follows from this definition will be a nested grid [19] and the level k defines how many points there are in the grid, i.e. the higher k the more nodes.

Chapter 12

Numerical tests for multivariate stochastic PDEs

This chapter will investigate the effects of having multiple random variables in the two problems previously investigated. The tests conducted in this section will be based on the stochastic Collocation method and will not involve tests with the Monte Carlo Sampling and the stochastic Galerkin method.

12.1 Test Equation with two random variables

In this section a brief introduction will be given to how to solve the multivariate Test equation and how to compute the statistics.
The approach is similar to the one described in Algorithm 5 and is outlined in Algorithm 9.

Algorithm 9 Pseudocode for SCM applied to the Test Equation

```
1: Determine the distributions of $\alpha$ and $\beta$ and the number of quadrature nodes to be used in each variable, $N_\alpha + 1$ and $N_\beta + 1$.
2: Compute the quadrature nodes $[z_\alpha]_{i=0}^{N_\alpha}$ and $[z_\beta]_{i=0}^{N_\beta}$ and the weights $[w_\alpha]_{i=0}^{N_\alpha}$ and $[w_\beta]_{i=0}^{N_\beta}$.
3: Compute the Kronecker product of the random variables and of the corresponding weights.
4: Compute the dot product of the $N = (N_\alpha+1)(N_\beta+1)$ weights $W = W_\alpha.*W_\beta$
5: Solve the deterministic system to obtain the $N$ deterministic solutions $u$.
6: Compute mean $\mathbb{E}[u] = \sum_{i=1}^{N} u_i W_i$
7: Compute variance $\mathrm{var}(u) = \sum_{i=0}^{N} \left(u_i - \mathbb{E}[u]\right)^2 w_i$.
```

As outlined in the theory the choice of quadrature rules depends on the distributions of the stochastic variables and the deterministic system has been solved by use of the function `ERK` implemented in MATLAB.

12.1.1 Implementation

The implementation of Algorithm 9 is seen in the appendix and some of the details are outlined here. The tensor product of the nodes and weights are computed by use of the MATLAB function `meshgrid`.
The right hand side function is implemented in the function `rhsSCMtest` and looks like this

```matlab
function un = rhsSCMtest(t,U,alpha)

un = alpha.*U;
```

The statistics of the solution are computed as described in the theory and is implemented in `TestEqStat` which yields

```matlab
function [Umean, Uvar] = TestEqStat(U,zW,zN,tspan)

% Computing mean:
Um = zeros(zN,length(tspan));
for zi = 1:zN
    Um(zi,:) = U(zi,:)*zW(zi);
end
Umean = sum(Um,1);
```

```

% Computing variance:
Uv=zeros(length(tspan),zN);
for zi = 1:zN
    Uv(:,zi) = (U(zi,:)-Umean(end,:)).^2*zW(zi);
end
Uvar = sum(Uv,2); Uvar = Uvar';
```

The computation of the exact mean and variance have been outlined in the theory and the implementation `TestEqEx` can be seen in appendix B.

12.1.2 Gaussian distributed α-parameter and initial condition β

The multivariate Test equation is solved for the case where α and β are both Gaussian distributed. The α-parameter is chosen to be standard normal distributed, i.e. $\alpha \sim \mathcal{N}(0,1)$, and the initial condition is distributed as $\beta \sim \mathcal{N}(1,1)$. Due to the distribution of the stochastic parameters the Gauss Hermite quadrature is chosen and six points are chosen in each dimension which leads to 36 deterministic solutions due to the tensor product of the grid points.
In figure 12.1 the 36 deterministic solutions are plotted.

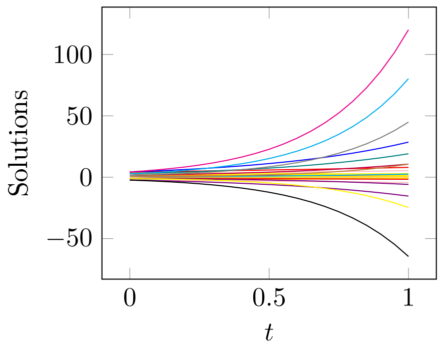

Figure 12.1: The $N = 36$ deterministic solutions.

It is seen that there is big variations in the 36 solutions. A difference from the univariate case is that now there are negative solutions which was not obtained previously and this means that a single deterministic solution can differ a lot from the computed mean since it could be different in sign and very different in size. The negative solutions are obtained since some of the computed initial

conditions are negative which leads to negative solutions.
The 36 deterministic solutions have been used to compute the statistics. The estimated mean $\bar{u}$ can be seen in figure 12.2 with an error bound computed by $\bar{u} \pm \bar{\sigma}$ as well as the exact mean.

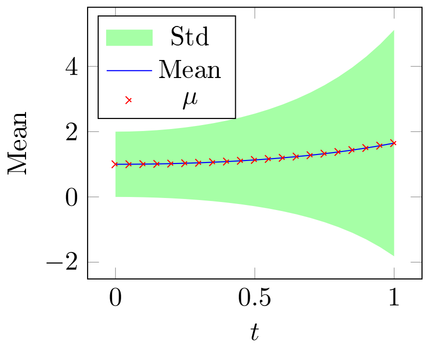

Figure 12.2: The estimated mean with the computed standard deviation and the exact mean.

From figure 12.2 it is seen that the mean has not changed a lot in comparison with the mean in figure 7.8.
In figure 12.2 it is seen that the stochastic initial condition introduces a lot more uncertainty in the solution in the first half of the time domain compared to 7.8. With time the effects of the stochastic initial condition becomes less dominant since the variance introduced by the α-parameter increases exponential with time.
The errors in mean and variance have been computed for different M and in figure 12.3 the errors have been plotted as function of M, where $m_\alpha = m_\beta = m$ which means that $M = m^2$.

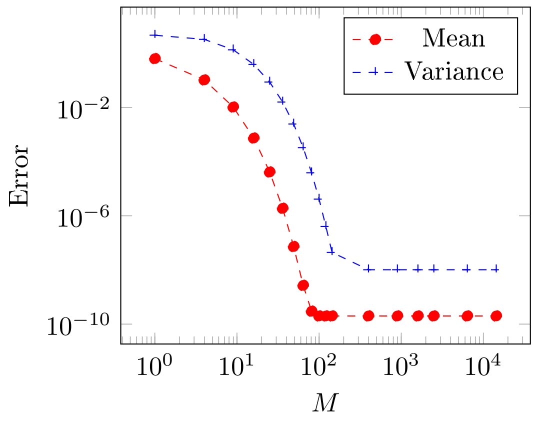

Figure 12.3: Error on the mean and variance for increasing M at time $t = 1$.

From the errors in figure 12.3 it is seen that the errors converge like it did in the univariate case except that now it is a function of M instead of just m_1.
Unless the stochastic β reduces the effects of the stochastic α it is expected that at least $M = 81$ is needed for achieving the smallest error in the mean since $M = m^2$ and $m_\alpha = 9$ was needed to minimize the error in α. Apparently $M = m^2 = 81$ is the optimal choice with regard to the error in the mean. This indicates that either is the effects of the parameter α reduced or else it holds that the best approximation in the mean can be obtained with $m_\beta \le m_\alpha$.
A simple test of this assumption is to maintain $m_\alpha = 2$ while increasing m_β and vice versa. This results in the error-plots in figure 12.4.

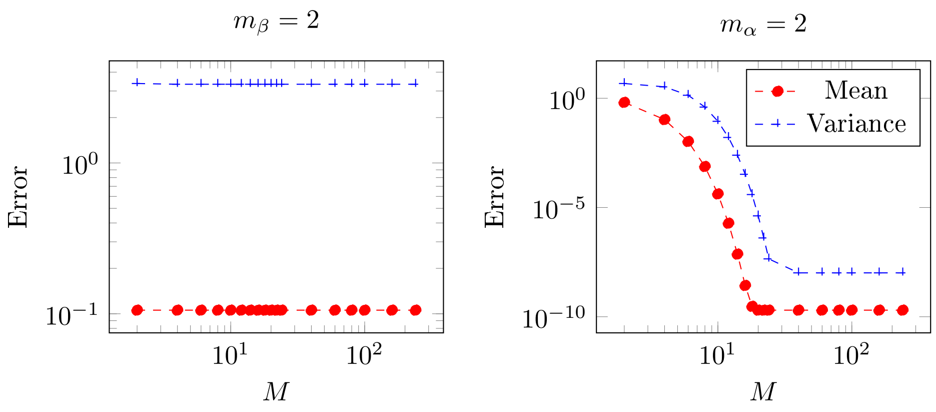

Figure 12.4: Error on the mean and variance at time $t = 1$ for increasing M. Left: $m_\beta = 2$ and increasing m_α. Right: $m_\alpha = 2$ and increasing m_β.

From the two plots it is seen that it is the number of quadrature points for representing α that makes a difference with regard to the convergence of the estimated mean and variance. The number of quadrature points used to represent the β-parameter does not have a great impact on the quality of the estimates. This means that instead of using $M = 9 \cdot 9 = 81$ deterministic solutions it is sufficient to compute $M = 9 \cdot 2 = 18$ deterministic solutions as long as $m_\alpha = 9$ and $m_\beta = 2$ and not the other way around.
This means that the α-parameter has much more influence on accuracy of the approximations of the statistics than the β-parameter. This is a good example on how a smart choice of quadrature nodes in each stochastic dimension can lead to much more efficient computations without compromising the quality of the approximations.

12.1.3 Gaussian distributed α and uniformly distributed initial condition β.

In this section it has been investigated how it affects the statistics of the solution to have two stochastic variables with different distributions namely $\alpha \sim \mathcal{N}(0, 1)$ and $\beta \sim \mathcal{U}(0, 2)$. Since β is uniformly distributed the quadrature chosen for this variable is Gauss Legendre quadrature. There are again 6 quadrature nodes and weights for both α and β which by use of tensor product leads to 36 deterministic solutions. Like in the previous test case the deterministic solutions are plotted and can be seen in figure 12.5.

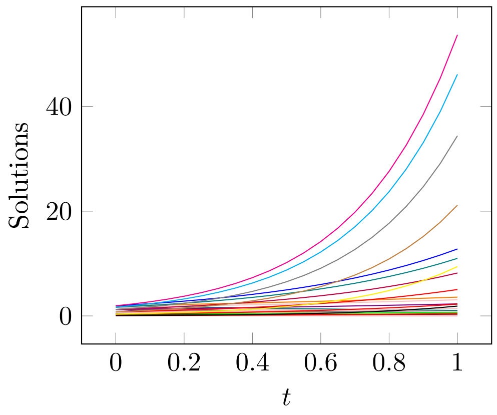

Figure 12.5: 12 of the $N = 36$ deterministic solutions.

The deterministic solutions in figure 12.5 are different from the ones in figure 12.1 since none of them are negative and they do not grow as much as when β was Gaussian distributed. This is because the quadrature nodes are strictly limited to be in the interval $[0, 2]$ in contrast to the other test case where β was not limited in this way.
A comparison of the deterministic solutions in figure 12.5 with the ones in figure 12.1 yields that it can have a great impact on the individual deterministic solution whether β is Gaussian or uniformly distributed. This difference is however not so visible when looking at the statistics as illustrated in figure 12.6 where the estimated mean, the std-bound and the exact mean are plotted.

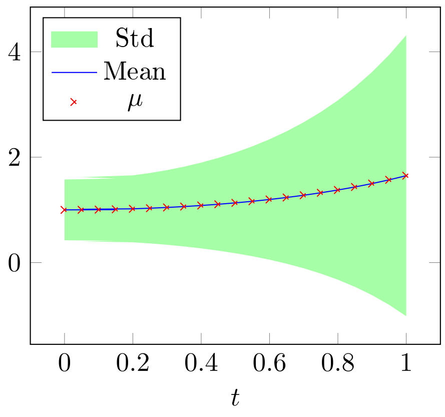

Figure 12.6: The estimated mean with the computed standard deviation as well as the exact mean.

It is seen that the mean is not greatly affected by the distribution of β but the standard deviation is different than the one seen in figure 12.2. It is smaller when β is uniformly distributed than when β is Gaussian distributed. Especially in the first half of the time domain the difference in the standard deviation is visible.
For this choice of parameters the error has been plotted in figure 12.7 as function of $M = m^2$ where $m = m_\alpha = m_\beta$.

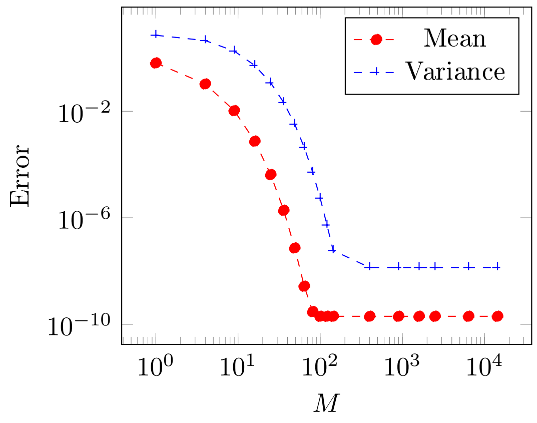

Figure 12.7: Error on the mean and variance for increasing M at time $t = 1$.

The tendency in the error in figure 12.7 is the same as was seen in figure 12.3. Again the optimal choice seems to be $M = 81$ which could indicate that the error is not greatly affected by the choice of distribution of β. But it worth to remember that the β-parameter apparently has much less influence on the accuracy than the α-parameter which means that with 9 quadrature points to represent β the shift in distribution should not change a lot.
To investigate this further the same simple test as was conducted in the previous section is conducted here. This means that $m_\alpha = 2$ is maintained while m_β is increased and vice versa. This resulted in the error-plots visualized in figure 12.8.

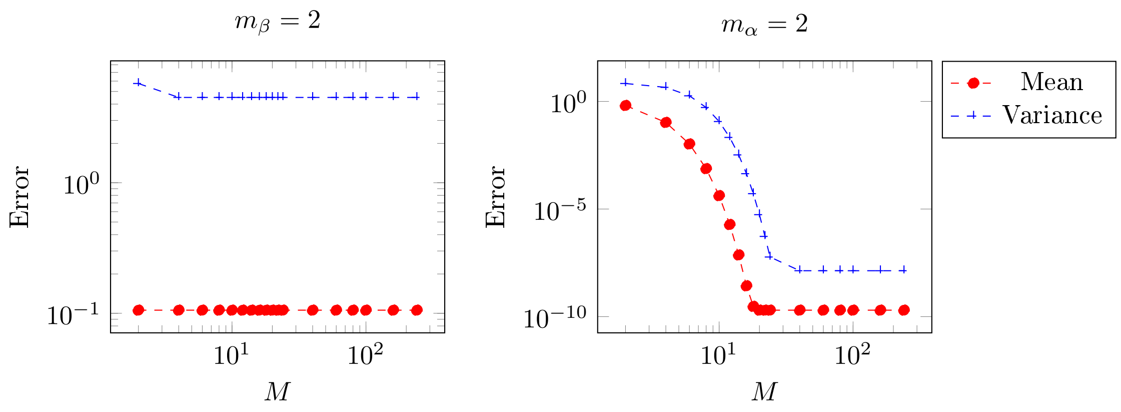

Figure 12.8: Error on the mean and variance at time $t = 1$ for increasing M. Left: $m_\beta = 2$ and increasing m_α. Right: $m_\alpha = 2$ and increasing m_β.

From the error plots it seems that the α-parameter is still the most dominant parameter when computing the statistics of the multivariate Test equation. Furthermore the change in distribution of β does not seem to have changed the optimal number of quadrature points to represent α and β which means that $m_\alpha = 9$ and $m_\beta = 2$, and 18 deterministic solutions are sufficient to obtain a reasonable approximations of the mean.

12.2 Multivariate Burgers' Equation

In this chapter it will be investigated how the statistics of the stochastic Burgers' equation will be affected by introducing uncertainty in both boundary condition and in the ν-parameter. The uncertainty in the BC's will be assumed to be uniformly distributed and therefore Gauss Legendre quadrature will be used to compute the quadrature nodes.

12.2.1 Burgers' Equation with stochastic boundary conditions

The 2-variate stochastic Burgers' equation is solved in this section and the effects of having stochastic BC's are investigated. First the outline of how the SCM is used on the multivariate Burgers' equation is outlined in Algorithm 10.

Algorithm 10 Pseudocode for SCM applied to the 2-variate stochastic Burgers' equation

```
1: Compute the spatial collocation points $x$.
2: Compute the differential matrices $\mathcal{D}$ and $\mathcal{D}^2$.
3: Compute the quadrature nodes $z_{b_1}$ and $z_{b_2}$ and the corresponding weights
   $w_{b_1}$ and $w_{b_2}$.
4: Compute the tensor product of the weights and nodes to obtain $Z_{b_1}$, $Z_{b_2}$
   and $\mathbf{W}$.
5: Compute the initial condition $IC$.
6: Use a deterministic solver to solve the system in time (reach steady state).
7: Compute the mean $\mathbb{E}[u] = \sum_{i=0}^{M} u_i W_i$
8: Compute the variance $\mathrm{var}(u) = \sum_{i=0}^{M} \left(u_i - \mathbb{E}[u]\right)^2 W_i$.
```

In general the approach is very similar to the approach used for the univariate Burgers' equation. The main difference is the computation of the stochastic grids and weights as well as the initial condition. Therefore the implementation of this is outlined in section 12.2.1.1.

12.2.1.1 Implementation

In this section the implementations that are significantly different than the implementations in univariate case is outlined. The rest of the code can be found in appendix B.
The interesting part is implemented in the MATLAB function `InitBurg2D` which looks like this

```
function [InitCond,U,Z,ZW] = InitBurg2D(zN,alpha,beta,nu,
    x,dStart,dEnd)

ck=0; xL = length(x);

Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

[z1,z1W] = JacobiGQ(alpha,beta,zN-1);
[z2,z2W] = JacobiGQ(alpha,beta,zN-1);

% Scaling the weights
z1W = z1W/2;
z2W = z2W/2;

% Scaling the disturbances.
z1 = ((z1+1)/2)*(dEnd(1)-dStart(1))+dStart(1);
z2 = ((z2+1)/2)*(dEnd(2)-dStart(2))+dStart(2);

[Z1,Z2] = ndgrid(z1,z2);
[Z1W,Z2W] = ndgrid(z1W,z2W);

Z1 = Z1(:); Z2 = Z2(:);
Z = [Z1 Z2];

Z1W = Z1W(:); Z2W = Z2W(:);
ZW = Z1W.*Z2W;

% Initialization and pre-allocation.
delta1 = Z1;
delta2 = Z2;
ZN = length(Z1);
InitCond = Uexact(x,0,nu);
```

```
U = zeros(xL*ZN,1);

for i = 1:ZN
    InitTemp = InitCond;
    InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta1
        (i));
    InitTemp(InitTemp<0) = InitTemp(InitTemp<0)*(1+delta2
        (i));
    U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

The BC's are uniformly distributed which means that Gauss Legendre quadrature is used.
The function `ndgrid` is used to compute all combinations of the nodes and of the weights. Then the weights are multiplied to attain the tensor product.
The initial condition is computed by using the exact solution and then change the values that are higher or lower than zero according to the grid of stochastic BC's. This is not strictly necessary in order to obtain good results but it speeds up the iteration process.

12.2.1.2 Tests

In this test a space grid of 31 points have been used and boundary conditions are distributed with $u(-1) \sim \mathcal{U}(1, 1.1)$ and $u(1) \sim \mathcal{U}(-1, -1.1)$.
The tests have been conducted with 4 quadrature points in each stochastic dimension which means that 16 deterministic solutions have been computed.
The one dimensional differentiation matrix $\mathcal{D}$ and its square $\mathcal{D}^2$ are of dimension 31×31 which means that the differentiation matrix for all the deterministic solutions is a $(18 \cdot 31) \times (18 \cdot 31)$ matrix. In this matrix $\frac{1}{18}$'th of the elements are non-zero which means that the sparse structure of the matrix is definitely worth to utilize. If there had been used more quadrature points the system would have been even more sparse which is worth to note when solving the deterministic system.
The 18 deterministic solutions are plotted in figure 12.9 and have some interesting characteristics.

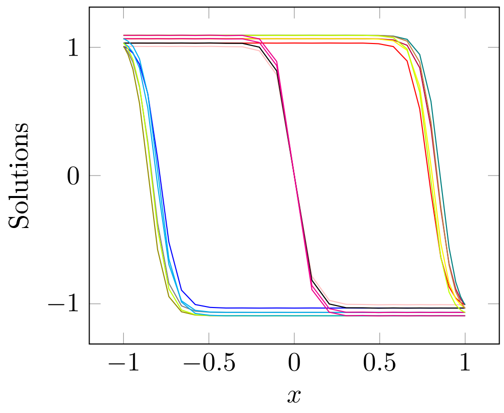

Figure 12.9: The $M = 18$ deterministic solutions for Burgers' Equation.

From figure 12.9 it is seen that the deterministic solutions are characterized by three different behaviours. The first kind of deterministic solutions are the ones recognized from the previous univariate test case where the transition layer is shifted to the far right in the plot due to a positive disturbance in the left boundary. This was the tendency that was observed in the univariate case and it is not surprisingly also present in this multivariate test since some of the deterministic solutions are characterized by disturbances in the left boundary but (almost) none in the right boundary.
The second type of deterministic solutions are when there are disturbances in both BC's and the transition layer is located in at $x = 0$. This means that some of the deterministic solutions have the same characteristics in terms of the transition layer as the unperturbed deterministic solution but with shifted boundary values such the values in $u(-1)$ are a bit higher than the original deterministic solution and the values in $u(1)$ are a bit lower.
The last kind of deterministic solutions are the opposite of what was observed in the univariate test case, namely that there is disturbance in the right boundary and (almost) none in the left boundary which means that the transition layer is shifted to the far left.
The multivariate Burgers' equation yields deterministic solutions that are much different from the ones obtained in the univariate case since some of the solutions are very similar to the unperturbed deterministic solution and since there is solutions where the transition layer is shifted both to the left and to the right.
The statistics of the multivariate Burgers' equation have also been computed and in figure 12.10 the mean and variance can be seen.

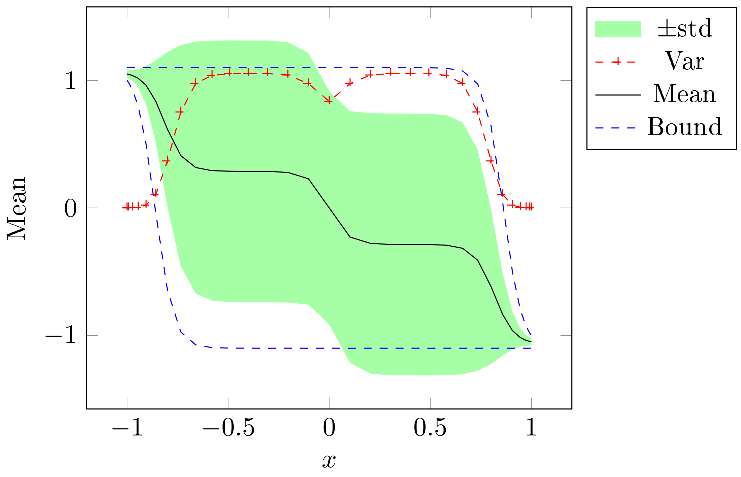

Figure 12.10: The estimated mean and variance and bounds on the solutions.

In figure 12.10 it is seen that the behaviour of the mean is very different from the mean computed for the univariate Burgers' equation and the solution of the deterministic Burgers' equation. Instead of having one transition layer where the mean goes from 1 to -1 it is divided into three.
It is also seen that there is a large variance and the uncertainty of this solutions is greater than seen for the other test cases.

12.2.2 3-variate Burgers' equation

Now the ν-parameter is chosen to be stochastic as well as the BC's. This means that it is now a three dimensional stochastic system and due to the tensor product of the stochastic variables this results in a potentially very large system. The overall approach is very similar to the 2-variate case and is outlined in Algorithm 11.

Algorithm 11 Pseudocode for SCM applied to the 3-variate stochastic Burgers' Equation

```
1: Compute the spatial collocation points $x$.
2: Compute the differential matrices $\mathcal{D}$ and $\mathcal{D}^2$.
3: Compute the quadrature nodes $z_{b_1}$, $z_{b_2}$ and $z_\nu$ and the corresponding weights $w_{b_1}$, $w_{b_2}$ and $w_\nu$.
4: Compute the tensor product of the weights and nodes to obtain $Z_{b_1}$, $Z_{b_2}$, $Z_\nu$ and $\mathbf{W}$.
5: Compute the initial condition $IC$.
6: Use a deterministic solver to solve the system in time (reach steady state).
7: Compute the mean $\mathbb{E}[u] = \sum_{i=0}^{M} u_i W_i$
8: Compute the variance $\mathrm{var}(u) = \sum_{i=0}^{M} \big(u_i - \mathbb{E}[u]\big)^2 W_i$.
```

By comparison of the pseudocodes in algorithm 10 and 11 it is seen that most of the steps are the same and some of the implementations can be reused. In the tests presented here the BC's are again chosen to be uniformly distributed as well as the ν-parameter.
The ν-parameter is very interesting since it is the viscosity of the Burger's equation and determines the smoothness of the PDE. If $\nu = 0$ there would be a shock discontinuity instead of a transition layer [19].
The polynomial representations used in the spectral methods are not well suited for describing discontinuities and $\nu = 0$ would therefore pose a serious problem. During the numerical tests a small ν was chosen which could not be solved with the current settings. This problem might be dealt with by using some other representations than the orthogonal polynomials but it is outside the scope of this thesis to investigate that.
If the ν-parameter is increased the solution of Burgers' equation becomes smoother and the transition layer would not be a region of such rapid change. It would in other words result in a very different solution which will be seen in the numerical experiments.

12.2.2.1 Implementation

The interesting part of the implementations compared to the earlier implementations is where the quadratures are used and the initial condition is computed. This is done in the MATLAB function `InitBurg3D` that is included here while the rest of the implementation is in appendix B.

```
function [InitCond,U,Z,ZW] = InitBurg3D(zN,alpha,beta,nu,
    x,dStart,dEnd)
```

```

ck=0; xL = length(x);

Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

% Computing the quadrature weights and nodes for the
    random BC's
[z1,z1W] = legendrequad(zN);
[z2,z2W] = legendrequad(zN);

% Scaling the weights
z1W = z1W/2;
z2W = z2W/2;

% Scaling the disturbances.
z1 = ((z1+1)/2)*(dEnd(1)-dStart(1))+dStart(1);
z2 = ((z2+1)/2)*(dEnd(2)-dStart(2))+dStart(2);

% Computing the nodes and weights for the nu paremeter
if strcmp(nu.t,'uniform')
    [z3,z3W] = JacobiGQ(alpha,beta,zN-1);

    % Scaling the weights and nu
    z3W = z3W/2;
    z3 = ((z3+1)/2)*(nu.par(2)-nu.par(1))+nu.par(1);

elseif strcmp(nu.t,'normal')
    [z3,z3W] = HermiteQuadN(zN);

    % Scaling nu
    z3 = nu.par(1)+nu.par(2)*z3;
end
%z3 = 0.05; z3W = 1;
[Z1,Z2,Z3] = ndgrid(z1,z2,z3);
[Z1W,Z2W,Z3W] = ndgrid(z1W,z2W,z3W);

Z1 = Z1(:); Z2 = Z2(:); Z3 = Z3(:);
Z = [Z1 Z2 Z3];

Z1W = Z1W(:); Z2W = Z2W(:); Z3W = Z3W(:);
ZW = Z1W.*Z2W.*Z3W;

% Initialization and pre-allocation.
delta1 = Z1; delta2 = Z2; ZN = length(Z1);
```

```
nuEst = sum(z3)/zN;
InitCond = Uexact(x,0,nuEst);
U = zeros(xL*ZN,1);

for i = 1:ZN
    InitTemp = InitCond;
    InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta1
        (i));
    InitTemp(InitTemp<0) = InitTemp(InitTemp<0)*(1+delta2
        (i));
    U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

The implementation is very similar to the code used in the two dimensional case and the interesting part is the computation of the tensor grids.
It might seem like an innocent extension of the code used previously but it leads to a very increased computational effort. This is due to the fact that $M = m^3$ if $m_{BC_1} = m_{BC_2} = m_\nu = m$ which means that the differentiation matrices are increased very much in size which makes the deterministic system much more costly to solve.

In the tests there are used 61 space points and $m = 4$ for each stochastic dimension which means that the differentiation matrices are of the size $(61 \cdot 4^3) \times (61 \cdot 4^3) = 3904 \times 3904$. This means that the two matrices contains 15,241,216 elements. Only a fraction of $\frac{1}{64}$ of these elements are non-zero which is about 1.5 % but it is still a large system that potentially has to be solved many times - depending on the solver - in order to reach steady state.
Assuming that $m = 5$ quadrature points are needed in each dimension to obtain the desired accuracy it will result in a much larger to system. Instead of having 64 deterministic solutions there is now 125 and the differentiation matrices are now of the size $(61 \cdot 5^3) \times (61 \cdot 5^3) = 7625 \times 7625$ and contains 58,140,625 elements. Even though "only" 465,125 elements are non-zero it is still a large system and it shows how important it is not to use too many quadrature points if it is not needed. It also demonstrates that when the dimensionality of a problem is increased the computational effort can be increased very much.

This example is a three dimensional stochastic problem and many of the stochastic PDE's to be solved have 50 or 100 which illustrates that the growth in the computational effort is a real problem in practise.
It is clear that sparsity has to be utilized when it is possible - both when solving the deterministic system but also - if possible - when computing the quadrature

nodes since fewer quadrature nodes in each dimension can significantly reduce the number of deterministic solutions.

12.2.2.2 Tests

The BC's have been chosen to be the same as those in the test of the 2-variate Burgers' equation and the ν-parameter is chosen to be univariate as well with the distribution $\nu \sim \mathcal{U}(0.05, 0.5)$. This means that there is a large uncertainty in the ν-parameter but it results in some interesting results and demonstrates the effects of the viscosity of the Burgers' Equation.
There are $m = 4$ quadrature points and weights to represent each stochastic variable which means that $M = 4^3 = 64$ deterministic solutions are computed. The 64 solutions have been plotted in figure 12.11.

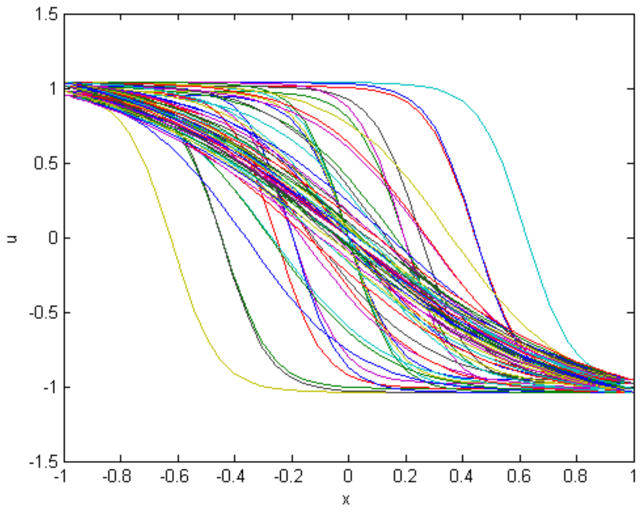

Figure 12.11: The deterministic solutions for the multivariate Burgers' Equation.

It is seen that the ν-parameter has great influence on the shape of the solutions and the transition layer. Theoretically it would require less quadrature points to represent the solution for increasing ν since the solution becomes smoother. Besides a smoothing effect it also effects the significance of the perturbed BC's. When ν is high the realizations of the stochastic Burgers' equation are less effected by the stochastic BC's. The stochastic ν is therefore a significant pa-

rameter when conducting UQ for Burgers' equation.
The effect on the statistics of having three stochastic variables have also been investigated. In figure 12.12 the mean and variance are plotted.

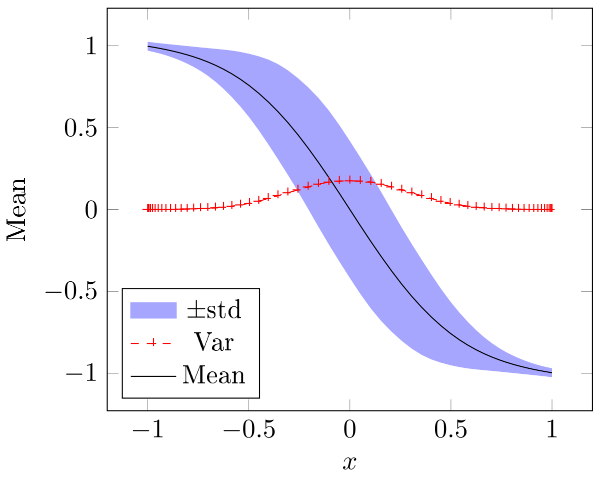

Figure 12.12: The mean, variance and bounds by standard deviation for the computed solution.

It is seen that the variance is relatively small compared to the one obtained in the 2-variate case and the transition layer in the mean is much smoother than previously which means that the stochastic ν-parameter has a smoothing effect on the mean and dampens the variance.
It seems like the stochastic ν-parameter decreases the variance of the solution but as seen from figure 12.11 there are larger variations in the deterministic solutions. Hence the smaller variance is not a result of very similar solutions but of the fact that the steep solutions have less influence on the statistics than the smooth solutions.
It is worth to note that the transition layer of the estimated mean is centred around $x = 0$ as it is for the unperturbed solution. The transition layer in the estimated mean does not give an exact representation of the solution to the unperturbed problem but it gives a better impression of the location of the transition layer than e.g. the estimated mean for the univariate Burgers' equation.

12.2.2.3 Test with a different right BC

In this section the right BC have been changed such that it is distributed as $u(1) \sim \mathcal{U}(-0.9, -1)$. The rest of the settings are the same as before and $m = 4$ for each stochastic variable which implies that 64 deterministic solutions have been computed. These solutions have been plotted in figure 12.13.

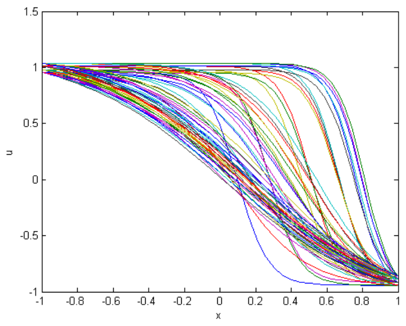

Figure 12.13: The deterministic solutions for the multivariate Burgers' Equation.

In figure 12.13 it seen that that there are no deterministic solutions which have a transition layer shifted to the left and that there are more solutions which are shifted to the right than previously. This will naturally affect the statistics as seen in figure 12.14 where the estimated mean and variance are plotted.

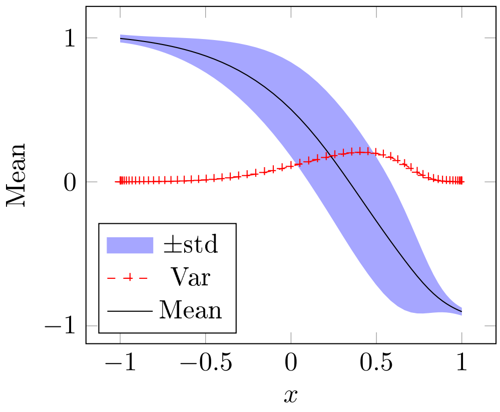

Figure 12.14: The estimated mean and variance.

The estimated mean and variance are skewed to the right in figure 12.14 compared to the results plotted in figure 12.12 and the transition layer is steeper.

Based on the investigations of the multivariate problems it safe to say that the stochastic parameters have a great influence on the solutions of the problems. Furthermore the effects of a stochastic parameter might be lessened or increased if another stochastic parameter in introduced and UQ is definitely a useful tool when working with e.g. stochastic PDEs. But as outlined in this chapter there are some difficulties that need to be solved and one of the main problems is the growth in computational effort when the number of stochastic variables is increased.

Chapter 13

Tests with Smolyak sparse grids

From the theory outlined previously it follows that each collocation node implies to compute a deterministic solution to the system at hand. This means that the sparse grids potentially implies a significantly reduction in the computational work and it is therefore lucrative to use sparse grids instead of the full tensor grids.
It is furthermore to be noted that the use of sparse grids becomes increasingly beneficial for increasing dimensions. The decreased computational effort and time consumption motivates the use of the sparse grids even though it should be noted that the sparse grids are not a cure to the curse of dimensionality but only a remedy to ensure a lessened impact.

The implementations made by John Burkardt [2] are used to conduct tests with the sparse Gauss Legendre (SGL) grid and the sparse Clenshaw-Curtis (CC) grid.
Both types of sparse grids are often used to represent uniformly distributed stochastic variables. The relationship between the grids and the distribution of the variables is similar to the relationship between the orthogonal polynomials and the distributions outlined in Table 4.1. This means that the two types of grids should be optimal when used to represent uniformly distributed variables

but they can be used for stochastic variables with other distributions.
It should be noted that the Smolyak grids in general are not restricted to represent uniformly distributed variables and there are many types of Smolyak sparse grids that can be applied.

13.1 Introduction of the sparse grids

The two sparse grids will be introduced further in this section and in the following sections they will be used to estimate the statistics of the stochastic solutions of the Test equation and the stochastic Burgers' equation.

13.1.1 Sparse Gauss Legendre grid

The sparse Gauss Legendre (SGL) grid is based on 1D Gauss Legendre quadrature rules which are combined, as introduced in section 11.3.
According to [2] the total number of nodes M in the sparse Gauss Legendre grid is coupled to the level and the stochastic dimension d as illustrated in table 13.1.

Level / d	1	2	3
0	1	1	1
1	3	5	7
2	9	21	37
3	23	73	159
4	53	225	597
5	115	637	2031
6	241	1693	6405

Table 13.1: Total number of points in the sparse Gauss Legendre grid.

The level here refers to the maximum attained level in the sparse grid which is represented by N in (11.4) and (11.5).
It should be noted that according to [2] the sparse Gauss Legendre grid is weakly nested in the sense that all the odd levels contain the value $X = 0.0$.
This is reflected in the outcome of the implementations since this point is repeated once for each odd level and once for level 0. This means that the two-dimensional SGL grid with level 2 contains 22 points where 21 are unique and with level $l = 3$ there are 75 points in the returned grid and 73 of these are unique. The grid points for level $l = 2$ which contains 21 unique points is plot-

ted in figure 13.1. The SGL grid has been plotted as well as a full tensor grid based on Gauss Legendre quadrature with 7 nodes in each stochastic dimension. The numerical tests for the Test equation includes a comparison of the accuracy of the estimates based on these two grids.

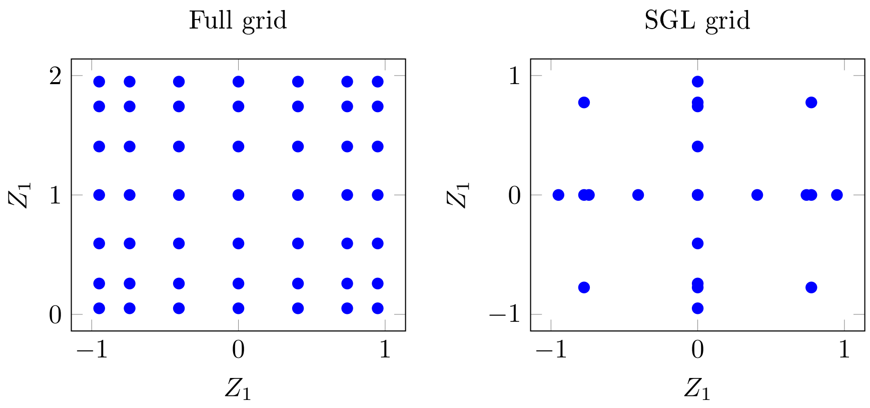

Figure 13.1: Left: A full tensor grid with 7 Gauss Legendre quadrature points in each dimension. Left: The SGL grid with level $l = 2$ and 21 unique nodes.

13.2 Sparse Clenshaw-Curtis grid

A major difference between the Clenshaw-Curtis (CC) grid and the SGL grid is that the Clenshaw-Curtis grid is nested. This means that the grid at a certain level is contained in a grid of a higher level. Another important difference between the two types of grids is that CC contains nodes on the boundary where the SGL only contains interior points. This can make a big difference can be seen in the numerical tests for Burgers' equation.
The CC grids does not follow the same correspondence between total number of nodes in the grid and the level l and dimension d, as was introduced in Table 13.1. In [4] this relationship is outlined for the CC grid and it can be seen in Table 13.2.

level / d	1	2	3
0	1	1	1
1	3	5	7
2	5	13	25
3	9	29	69
4	17	65	177
5	33	145	441

Table 13.2: Total number of nodes in the sparse Clenshaw-Curtis grid.

It is seen that for the same level the CC grid contains fewer points than the SGL grid. But the SGL grid has higher accuracy if the same level is applied. The accuracy of the two grids for specific levels will be investigated in the numerical tests for the Test equation.
It should be noted that the higher the dimension the larger is the difference in the number of points. This means that even though the SGL grid should be more efficient than CC for a certain level, the CC grid might be worth to use in high dimensions since the computational effort is much smaller.

A plot of the 65 grid points for a level 4 sparse CC grid is plotted in figure 13.2 as well as a full tensor grid of 17 points in each dimension.

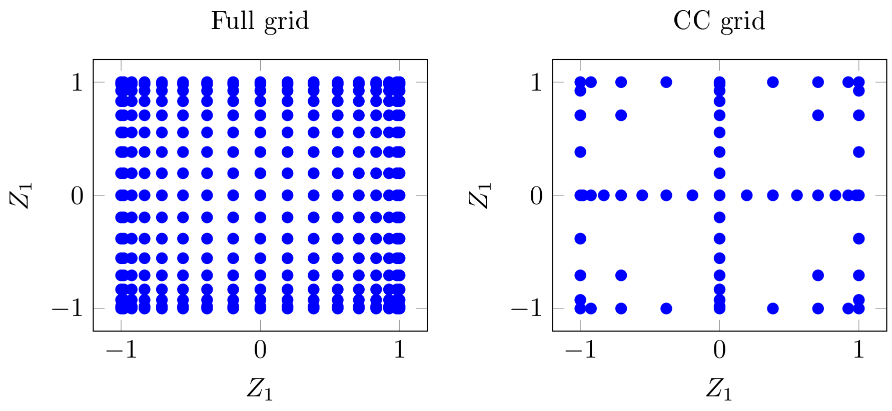

Figure 13.2: In this figure a full tensor grid is illustrated to left the and to the right a sparse CC grid is plotted.

In figure 13.2 it is seen that the sparse CC grid corresponds to a subset of the full tensor grid. The application of the sparse CC grid can be seen in the following sections.

13.3 Smolyak sparse grid applied

In this section the results of the numerical tests with the two sparse grids are presented. The numerical tests involve the 2-variate Test equation and the 2-variate Burgers' equation.
The implementations for this section are very similar to the ones used in the previous multivariate tests with the only difference that the tensor grids of quadrature points have been replaced with the sparse grids. This means that the rhs-function, the deterministic solver etc. are the same as previously. Because of this the implementations have not been included here but the test scripts can be found in appendix B. For the implementations of the sparse grid the reader is referred to [2].

13.3.1 The 2-variate Test Equation

The stochastic Test equation with uniform α and β has been investigated using the two sparse grids. The two variables are distributed with $\alpha \sim \mathcal{U}(-1,1)$ and $\beta \sim \mathcal{U}(0,2)$.
The SGL grid and CC grid with level 3 have been used to estimate the statistics for the stochastic Test equation and the results can be seen in the following sections. Since the estimates of the statistics are very similar to the ones obtained with the full tensor grids the numerical tests in this section will be an investigation of the errors. Furthermore the accuracy of the the grids have been compared with the accuracy obtained with the full tensor grid in section 13.3.1.3.

13.3.1.1 Sparse Gauss Legendre grid

In this section the SGL grid is applied and the error of the obtained estimates have been computed. The errors have been plotted as functions of the total number of grid points, M. The correspondence between number of points and the levels can be seen in table 13.1. The grids have been computed for the levels $l = 0, 1, \ldots, 6$ and the errors for the corresponding grid sizes have been plotted in figure 13.3.

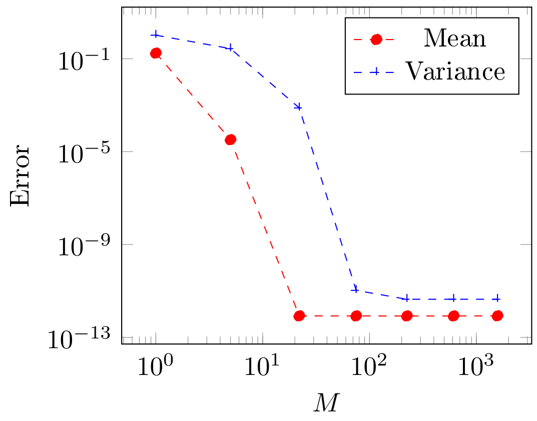

Figure 13.3: Error of the estimated mean and variance for increasing M at time $t = 1$.

It is seen that the error drops rather quickly and the error reaches a minimum which is equivalent to the minimum error obtained in the previous tests of the multivariate Test equation at level $l = 3$ where the grid contains 21 unique points.

13.3.1.2 Sparse Clenshaw-Curtis grid

The sparse Clenshaw-Curtis grid has been use to estimate the statistics for the multivariate Test equation. The errors using the CC grid have been plotted in figure 13.4 as function of the total number of grid points, M.

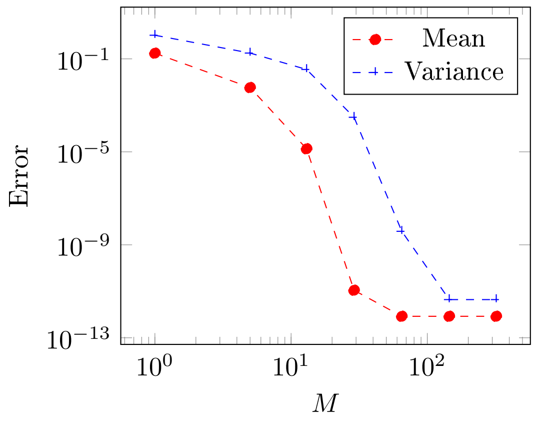

Figure 13.4: Error of the estimated mean and variance for increasing M at time $t = 1$.

It is seen that the convergence of the errors when using the CC grid is not impressive compared to the convergence obtained previously using full tensor grids. This means that the CC grid might not be suited for estimating the statistics of this problem.

13.3.1.3 Comparison of the errors on the estimated statistics

As illustrated in figure 13.3 and figure 13.4 the two sparse grids can be used as collocation points for the stochastic Collocation method which leads to fair results when the stochastic variables are uniformly distributed. In this section the accuracy of the SCM used with SGL and CC grids is investigated further. As a comparison the SCM has been used with full tensor grid on the same problem and the errors are included in Table 13.3.
The full tensor grids are computed from 1D Gauss Legendre quadrature and the same number of quadrature nodes has been used in each dimension, i.e. the total number of nodes is $M = m_1 \times m_2 = m^2$ where m is the number of quadrature nodes in each dimension.

M	Error in the mean	Error in the variance
1	$1.75 \cdot 10^{-1}$	1.04
4	$3.85 \cdot 10^{-3}$	$8.34 \cdot 10^{-2}$
9	$3.28 \cdot 10^{-5}$	$2.98 \cdot 10^{-3}$
16	$1.48 \cdot 10^{-6}$	$5.39 \cdot 10^{-5}$
25	$4.13 \cdot 10^{-10}$	$5.98 \cdot 10^{-7}$
36	$1.63 \cdot 10^{-12}$	$4.52 \cdot 10^{-9}$
49	$8.43 \cdot 10^{-13}$	$2.91 \cdot 10^{-10}$
64	$8.41 \cdot 10^{-13}$	$4.50 \cdot 10^{-12}$
81	$8.42 \cdot 10^{-13}$	$4.49 \cdot 10^{-12}$
100	$8.41 \cdot 10^{-13}$	$4.49 \cdot 10^{-12}$

Table 13.3: The errors in the estimated statistics and the corresponding number of nodes in the full tensor grid.

From table 13.3 it is seen that the minimum error in the mean is obtained for 49 grid points and 64 grid points for the variance. This corresponds very well with the investigations made previously for the univariate Test equation with uniformly distributed α. In this case 7 quadrature nodes was needed to obtain the minimum error in the estimated mean and 8 quadrature nodes to obtain the minimum error in the estimated variance.
Furthermore the previous numerical tests with the 2-variate stochastic Test equation indicated that the α-parameter was more dominant than β in terms of the error in the estimated statistics. This corresponds very well with the optimal number of tensor grid points for the full tensor grid since it corresponds to chose the optimal choice of quadrature nodes for α in each dimension.

The SGL grid is used as collocation points for the SCM for the stochastic Test equation. The grids with level $l = 0, 1, \ldots, 6$ are used and the errors of the estimated statistics are outlined in table 13.4. A comparison of Table 13.4 efficiency of the SCM is higher when using the SGL grids than when using full tensor grids in the sense that the same accuracy is obtained using SGL grids but with fewer grid points.

Level	M	Error in the mean	Error in the variance
0	1	$1.75 \cdot 10^{-1}$	1.037
1	5	$3.27 \cdot 10^{-5}$	$2.73 \cdot 10^{-1}$
2	22	$8.42 \cdot 10^{-13}$	$7.65 \cdot 10^{-4}$
3	75	$8.42 \cdot 10^{-13}$	$1.06 \cdot 10^{-11}$
4	224	$8.40 \cdot 10^{-13}$	$4.39 \cdot 10^{-12}$
5	613	$8.40 \cdot 10^{-13}$	$4.39 \cdot 10^{-12}$
6	1578	$8.48 \cdot 10^{-13}$	$4.39 \cdot 10^{-12}$

Table 13.4: The errors in the estimated statistics and the corresponding levels and total number of nodes in the SGL grid.

Numerical tests are conducted using the CC grids as well and this resulted in computation of the errors of the estimated mean and variance which can be seen in Table 13.5.

Level	M	Error in the mean	Error in the variance
0	1	$1.75 \cdot 10^{-1}$	1.04
1	5	$5.83 \cdot 10^{-3}$	$1.78 \cdot 10^{-1}$
2	13	$1.35 \cdot 10^{-5}$	$3.49 \cdot 10^{-2}$
3	29	$1.11 \cdot 10^{-11}$	$3.06 \cdot 10^{-4}$
4	65	$8.39 \cdot 10^{-13}$	$3.81 \cdot 10^{-9}$
5	145	$8.39 \cdot 10^{-13}$	$4.39 \cdot 10^{-12}$
6	321	$8.40 \cdot 10^{-13}$	$4.39 \cdot 10^{-12}$

Table 13.5: The errors of the estimated statistics and the corresponding levels and number of nodes in the CC grid.

The accuracy of the CC grids is lower than the accuracy obtained using the SGL grids on this particular test problem. This is not necessarily true in general but it still yields an interesting result. It seems like fewer grid points are needed when using the full tensor grid than when using the CC grid to obtain the minimum error. But this is might just be due to the fact that the CC grid has 29 grid points in the level 3 grid and 65 grid points for level 4. This means that because the optimal choice of level is not level 3 at least 65 nodes are used to obtain the best estimates which is more than in the full grid.
It is furthermore to be noted that the CC grid might be much more useful in higher dimensions than 2 since the sparse grids are introduced to reduce the effects of the curse of dimensionality in high dimensions.

13.3.1.4 Discussion

The sparse grids seems to be very useful and the use of the SGL grid definitely demonstrates that a high accuracy can be obtained with fewer nodes in comparison to the full tensor grid.
It should be noted that for the stochastic Test equation it seems like α is more dominant in terms of influence on the error than β. Therefore a full tensor grid might involve fewer quadrature nodes to represent β and thereby lead to a smaller number of total nodes, while maintaining a high accuracy. Therefore the sparse grids are more efficient than the full tensor grids investigated in this section but smarter grids might be applied which are full tensor grids but still involves relatively few grid points.
This is one of the reasons why a method like ANOVA can be very useful since ANOVA can be used to investigate which of the stochastic variables that have a great influence on the statistics and which of them are less important.

13.3.2 The 2-variate Burgers' Equation

The SCM is tested for the stochastic Burger' equation using the two Smolyak sparse grids introduced previously. The tests are conducted for the case where the BC's are stochastic and distributed as $u(-1,t) \sim \mathcal{U}(1, 1.1)$ and $u(1,t) \sim \mathcal{U}(-1, -0.9)$.

13.3.2.1 Sparse Gauss Legendre grid

In this section SGL is used as collacation points when computing estimates of the statistics for the Burgers' equation with stochastic BCs.
The SGL grid with level $l = 2$ and a total of 21 unique nodes results in the estimates of the statistics plotted in figure 13.5.

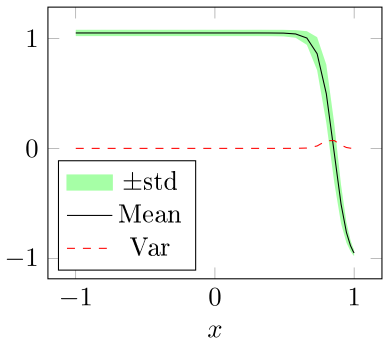

Figure 13.5: The estimated mean with the computed standard deviation as well as the variance.

The estimates seem to correspond very well to the results obtained previously where the full tensor grid was used. A comparison with figure 12.10 shows that the estimated statistics are much alike. The difference between the case where a full tensor grid with 9 quadrature points in each stochastic dimension, i.e. 81 grid points, and the case with a level $l = 2$ SGL grid has been plotted in figure 13.6.

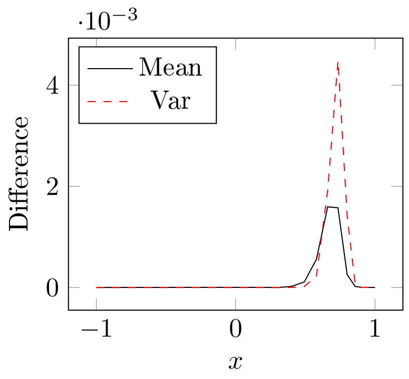

Figure 13.6: The difference between the estimated statistics.

It is seen that SCM used on the SGL grid with level $l = 2$ does not reproduce the same results as the SCM used with the full tensor grid with 81 nodes. But they are very similar.
Since the differences in figure 13.6 are the differences between two sets of estimates of the statistics it does not say anything about the accuracy of the estimates - only that they differ. In this context it does not matter that much

which of the two grids that produces the most accurate estimates since the estimates are so close and it demonstrates that the sparse grids could be used instead of a full tensor grid and it could reduce the computational effort.

13.3.2.2 Sparse Clenshaw-Curtis grid

In this section the CC grid is used as collocation points and the level of the CC grid is chosen to be 2 which means that 13 grid point are used. The 13 collocation points implies the computation of 13 deterministic solutions, which are plotted in figure 13.7.

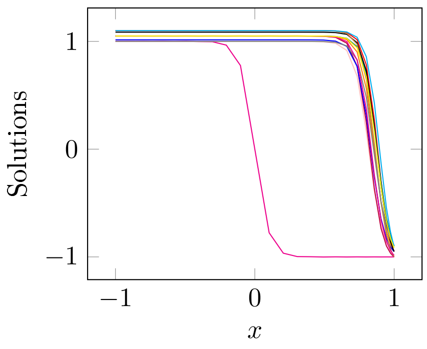

Figure 13.7: The 13 deterministic solutions for a level 2 CC grid.

It is seen that since the CC grids include boundary points the solution with the unperturbed BC's, i.e. $u(-1) = 1$ and $u(1) = -1$, is among the deterministic solutions. This has not been the case when using the SGL grid and the full tensor grid and might have a great effect on the estimated statistics - especially since the estimates are based on only 13 deterministic solutions.
The approximated statistics can be seen in figure 13.8 where it is seen that the statistics are very different from the estimates based on the full tensor grid and the SGL grid as seen in figure 13.5.

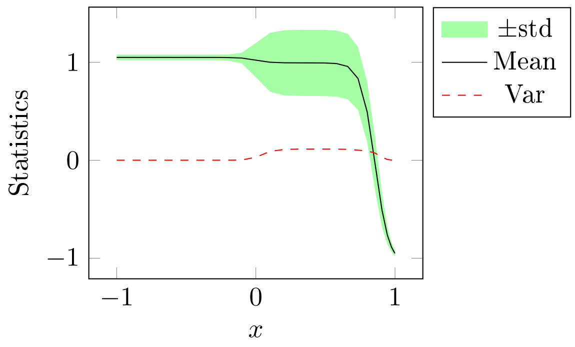

Figure 13.8: The estimated statistics computed by using a CC grid with level 2.

It is interesting to see how the estimates have changed due to the inclusion of the unperturbed solution and it demonstrates that it can make a great difference which collocation points are used when computing the estimates.
This illustrates that the CC grids has some different qualities than the SGL grids and the choice of grid is therefore not only a choice based on accuracy but also based on the special characteristics of the grids.

13.4 Conclusion

The tests in this chapter have illustrated that the sparse grids can be used to obtain high accuracy results and that they can decrease the number of nodes needed to obtain these results. Fewer nodes means less deterministic solutions and thereby less computational effort.
The tests with the sparse grids yields some interesting results, but the sparse grids are designed to be used in high dimensions. This means that the tests of 2-variate differential equations might not illustrate all the benefits of using the sparse grids.

As outlined in the beginning of the chapter the two sparse grids have different properties, which has been confirmed in the tests. In order to choose which sparse grid that is best suited to a given problem the user has to consider both accuracy and the properties of the sparse grids.
Furthermore the user has to consider whether the use of sparse grids is the optimal choice of method to reduce the computational effort. Other methods might be more efficient in some cases e.g. ANOVA. There are many aspects to

consider when handling UQ and the sparse grids is a useful tool that should be considered when solving high dimensional problems.

Chapter 14

Discussion

The focus in this thesis has been to investigate spectral methods for Uncertainty Quantification and in particular an investigation of the stochastic Collocation method. As a reference the Monte Carlo sampling has been introduced and applied on the Test equation.
The test cases in this thesis has been low dimensional problems which have relatively smooth solutions and under these conditions the spectral methods have proved efficient and accurate.
The numerical tests and the outlined theory has definitely motivated the use of spectral methods which have a much higher convergence rate than the Monte Carlo sampling. But it has also been discovered that the spectral methods has some weaknesses. One of them is the computational work of especially the Collocation method when applied to problems of higher dimensions. The Galerkin is more efficient than the Collocation method but it comes at the cost of much more work in terms of derivations and implementations.
Another perspective of the two methods is that the stochastic Galerkin method usually is based on solving a system of coupled equations whereas the solutions computed in the stochastic Collocation method are de-coupled. The de-coupling of the solutions could be utilized by introducing parallel programming. In this way the computation time could be greatly reduced.

The computational work of the spectral methods can be lessened by use of e.g. Smolyak sparse grids. The introduction sparse grids is a useful tool that even for 2-variate problems can lead to high accuracy and relatively low computational effort. In higher dimensions the gain in terms of less computational effort is even greater. Other methods to achieve the spectral convergence of the spectral methods and decrease the computational effort can be applied like the compressive sampling method. This means that the spectral methods can be a useful tool not only in low dimensions but potentially in very high dimensions.

The thesis does not include numerical tests in high dimensions and the efficiency of the spectral methods combined with e.g. sparse grids has not been established for high dimensional problems. This means that even though the sparse grids decrease the computational effort the Monte Carlo sampling could potentially still be more efficient in high dimensions than the spectral methods.
Another strength of the Monte Carlo sampling is that the convergence and accuracy of the method is independent of the characteristics of the solutions. The spectral methods introduced in this thesis are based on polynomial representations which means that the solutions should be smooth in order to obtain good representations. This could pose a considerable problem and for example the solution of the stochastic Test equation could be difficult to represent by use of spectral methods if the ν-parameter is too low, i.e. the solutions become non-smooth.

14.1 Future work

The thesis has focused on one type of method to reduce the effects of the curse of dimensionality which is the use of Smolyak sparse grids. The next step would be to take these investigations further by numerical tests in higher dimensions. These tests could reveal the full potential of the sparse grids and might demonstrate unforeseen qualities or flaws of the sparse grids.
Another natural step would be to do some numerical tests with the ANOVA, the non-adapted sparse method and the sparse pseudospectral approximation method. These tests could be conducted as well as the sparse grid tests in high dimensions and a comparison of the accuracy and efficiency of the methods could be done. This would be a very comprehensive task if the tests where to be conducted on several advanced problems in high dimension but it could reveal some interesting guidelines to when to use which methods.
Another interesting perspective could be to utilize that the stochastic Collocation method is based on uncorrelated deterministic solutions by computing the deterministic solutions by use of parallel programming.
Finally it could be interesting to investigate if it is possible to estimate the

statistics of non-smooth solutions efficiently and accurately. Potentially the investigations could involve high dimensional problems with non-smooth solutions.

Chapter 15

Conclusion

This thesis introduce and tests three different methods for Uncertainty Quantification. The Monte Carlo Sampling proved to have slower convergence for the univariate Test equation but the method is relatively easy to apply and the convergence is independent of the dimensionality of the problem at hand. The stochastic Galerkin method and the stochastic Collocation method have greater convergence rate for the univariate Test equation but both methods requires more computations when the dimensionality is increased.
Furthermore it is to be noted that the Monte Carlo sampling is independent of the structure of the solutions - it basicly just requires that enough solutions can be computed. This is a strength compared to the spectral methods due to the computational effort in high dimensions for these methods and the fact that they are based on approximations with polynomials implies that they can have difficulties with representing e.g. non-smooth solutions.
The stochastic Galerkin method is in general more accurate than the stochastic Collocation method but requires a lot more work in terms of derivations and implementation. The stochastic Collocation method is easier to apply but suffers under the curse of dimensionality.

Due to the relative ease of use and the spectral convergence the stochastic Collocation method is a popular choice of UQ method. The curse of dimensionality has led to a lot of research to minimize the effects. This includes the studies of

Smolyak sparse grids and the application of these.
The sparse grids can assure high accuracy with a relatively low number of nodes which means that the computational effort of the stochastic Collocation method is greatly reduced since the computational effort of the method is related to the number of deterministic solutions which is equal to the number of nodes.

Two different types of grids have been used in this thesis and they have different characteristics. The Clenshaw-Curtis grid is nested and contains the boundary nodes while the sparse Smolyak Gauss Legendre grid is not nested and does not contain the boundary nodes. The sparse Gauss Legendre grid is more accurate for a given level than the Clenshaw-Curtisg grid but the Clenshaw-Curtis grid contains fewer points for each level. This means that in many cases the optimal choice of sparse grid could be problem dependent.

The Uncertainty Quantification is definitely a useful tool which can be applied in many different settings since uncertainty is present whenever measurements are made and parameters are estimated. This thesis has introduced spectral methods for Uncertainty Quantification and tested one method to reduce the effects of the curse of dimensionality but there is without a doubt much more within the field which is worth to study further. Tests in higher dimensions seems like a natural step from here and the application of other methods as well could very likely be lucrative. The ANOVA and the compressive sampling definitely seems like interesting alternatives to the Smolyak grids in order to avoid the effects of the curse of dimensionality.

Appendix A

Supplement to the mathematical background

A.1 Orthogonal Polynomials

Here an introduction to an alternative definition of the Hermite Polynomials is given.

A.1.1 Alternative definition of the Hermite polynomials

The alternative definition of the Hermite Polynomials is denoted $H_n(x)$ and is based on the following weight functions

$$w_H(x) = \frac{1}{\sqrt{\pi}} e^{-x^2}$$

Furthermore the definition of $H_n(x)$ is

$$H_n(x) = \frac{1}{(-1)^n e^{-x^2}} \frac{d^n}{dx^n}[e^{-x^2}] = n! \sum_{k=0}^{[n/2]} (-1)^k \frac{1}{k!(n-2k)!}(2x)^{n-2k}.$$

The three-term recurrence relation is

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x), \tag{A.1}$$

with $H_0(x) = 1$ and $H_1(x) = 2x$. The square norm of the polynomials are given as

$$\gamma_n = \langle H_n(x), H_n(x) \rangle = \frac{1}{\sqrt{\pi}} \int_{-\infty}^{\infty} He_n^2(x) e^{-x^2} dx = 2^n n!.$$

A.2 Probability theory

In this section some basic concepts within probability theory which can be useful in UQ is introduced.

A.3 Random fields and useful spaces

The UQ discussed in this thesis is based on uncertainty in the input to a model which means that the deterministic systems to be solved become stochastic since some or all of the input parameters are stochastic. To cope with this a description of the stochastic parameters is needed.
Generally speaking the stochastic inputs and outputs of a mathematical model can be classified as random fields which are denoted $X(\omega)$. A random field is a mapping from the probability space Ω to the function space V, i.e. $X : \Omega \to V$. In the case where $V = \mathbb{R}$ the $X(\omega)$ is called a random variable and when V is a function space over a space and/or time interval $X(\omega)$ is called a random field or a stochastic process.
The $\omega \in \Omega$ is generally called an outcome and Ω is the outcome space. Furthermore it can be useful to introduced the space of all relevant events denoted Σ, where events are subsets of Ω. An event could be $\{\omega : X(\omega) = c\}$ or $\{\omega : X(\omega) \leq k\}$. The Σ is called a σ-algebra and is in [19] defined as

Definition A.1 A collection of subsets of Ω denoted Σ is a σ-algebra if it satisfies the conditions

- Σ is not empty, i.e. $\emptyset \in \Sigma$ and $\Omega \in \Sigma$.
- If $A \in \Sigma$, then so is the complement set as well, $A^{\mathsf{C}} \in \Sigma$.

- If $\{A_i\} \in \Sigma$ then

$$\bigcup_{i=1}^{\infty} A_i \in \Sigma \quad \text{and} \quad \bigcap_{i=1}^{\infty} A_i \in \Sigma.$$

Two examples of σ-algebras - or σ-fields as they are also called - are

$$\begin{aligned}\Sigma_1 &= \{\emptyset, \Omega\} \\ \Sigma_2 &= 2^{\Omega} \equiv \{A : A \subset \Omega\}\end{aligned}$$

where Σ_1 is the smallest σ-algebra on Ω and Σ_2 contains all subsets of Ω and is thereby is the largest σ-algebra. Σ_2 is often referred to as the power set of Ω [19].

Two other useful concepts are probability measure and probability space. The probability measure used in this context can be defined as

Definition A.2 For a countable event space, Ω, and the σ-algebra on Ω, $\Sigma_2 = 2^{\Omega}$, P is a probability measure if

- $0 \leq P(A) \leq 1, \forall A \in \Sigma.$
- $P(\Omega) = 1.$
- For $\{A_i\} \in \Sigma$ and $A_i \cap A_j = \emptyset$, $\forall i \neq j$, it holds

$$P\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} P(A_i).$$

The probability measure is also referred to as a probability. By using Ω, Σ and P as defined in definition A.2 the probability space can be defined as the triplet $\{\Omega, \Sigma, P\}$.

A.4 Convergence and Central Limit Theorem

When working with stochastic variables it is often useful to use other convergence definitions than for instance the mean square convergence. Therefore a definition of a weak convergence will be given here.
The weak convergence called convergence in distribution can be very useful when a strong convergence can not be established. The definition of the convergence follows [19] and yields

Definition A.3 (Convergence in distribution) Convergence in distribution of a sequence $\{X_n\}$ to a random variable X is denoted $X_n \xrightarrow{d} X$ and is fulfilled if for all bounded and continuous functions, f, it holds that $\mathbb{E}[f(X_n)] \to \mathbb{E}[f(X)]$ for $n \to \infty$.

The convergence in distribution holds if and only if there is convergence in the distribution function, F_X, for all continuous points $x \in I_X$, i.e. $F_{X_n}(x) \to F_X(x)$ for $n \to \infty$.

Another useful definition is the Central Limit Theorem (CLT) which is used in many applications of probability theory and is the base for numerical tools as the Monte Carlo Sampling which will be introduced later on.

Theorem A.4 (Central Limit Theorem) *Let $\{X\}_{i=1}^{N}$ be independent and identically distributed (i.i.d.) random variables with mean μ and variance σ^2. Then the cumulative density function (CDF) of*

$$Y = \frac{1}{N}\sum_{i=1}^{N} X_i, \tag{A.2}$$

will converge to a normal distribution, $\mathcal{N}(\mu, \frac{\sigma}{\sqrt{N}})$, for $N \to \infty$. Furthermore it follows that the CDF of

$$\bar{Y} = \sqrt{N}\left(\frac{Y-\mu}{\sigma}\right), \tag{A.3}$$

converges towards a standard normal distribution, $\mathcal{N}(0,1)$.

For more information and the proof of CLT see e.g. [8], [16] or [19].

A.5 Introduction of strong and weak gPC approximation

In this section both the strong and weak gPC approximations are introduced.

A.5.1 Strong gPC approximation

A strong gPC approximation can be defined as in [19] like

Definition A.5 (Strong gPC approximation) Let Z be a random variable with support I_Z and probability distribution $F_Z(z) = P(Z \leq z)$. Furthermore let $f(Z)$ be a function of Z and $\mathbb{P}_N(Z)$ be the space of polynomials in Z of degree up to $N \geq 0$. A strong gPC approximation is $f_N(Z) \in \mathbb{P}_N(Z)$ such that for a proper norm, $\|\cdot\|$, defined on the support I_Z it holds that $\|f(Z) - f_N(Z)\|$ converges towards 0 for $N \to \infty$.

One type of strong gPC approximation is the gPC projection, where the N'th degree gPC orthogonal projection for a function $f \in L^2_{dF_Z}(I_Z)$ is

$$P_N f(Z) = \sum_{i=0}^{N} \hat{f}_i \Phi_i(Z), \quad \hat{f}_i = \frac{1}{\gamma_i} \mathbb{E}[f(Z)\Phi_i Z],$$

where $\{\gamma_i\}$ are the normalization factors. From approximation theory the existence and convergence follows, see [19], which means that the mean-square convergence, $\|f - P_N(f)\|_{L^2_{dF_Z}} \to 0$ for $N \to \infty$, holds. Furthermore this approximation is the optimal approximation of f with a polynomial of order up to N.

The convergence of the orthogonal projection follows from the theory but the convergence rate depends heavily on the smoothness of f [19]. The smoother f is the faster the convergence.

A.5.2 Weak gPC approximation

The strong gPC approximation is a very useful tool but it requires knowledge of how f depends on Z, i.e. the explicit form of f in terms of Z, which is not always at hand. It is not unusual that in practice it is only the probability distribution of f that is known which leads to a weak gPC approximation instead of a strong gPC approximation. This can be defined as in [19] which defines the weak gPC approximation as

Definition A.6 (Weak gPC approximation) Let Z and Y be a random variables with support I_Z and I_y respectively and probability distribution $F_Z(z) = P(Z \leq z)$ and $F_Y(y) = P(Y \leq y)$, respectively. Furthermore let $\mathbb{P}_N(Z)$ be the space of polynomials in Z of degree up to $N \geq 0$. A weak gPC approximation is $Y_N \in \mathbb{P}_N(Z)$ such that Y_N converges towards Y in a weak sense, e.g. in probability.

It is to be noted that a weak gPC approximation is not unique and that the strong gPC approximation implies weak approximation but not the other way around.
The weak gPC approximation when Y is an arbitrary random variable and only the probability distribution is known generally takes the form

$$Y_N = \sum_{i=0}^{N} a_i \Phi_i(Z), \quad a_i = \frac{1}{\gamma_i}\mathbb{E}[Y\Phi_i(Z)]. \tag{A.4}$$

where $\{\gamma_i\}$ are the normalization factors. This is the general form but since only the probability distribution of Y is known it is not possible to evaluate the integral of the expectation since the dependence between Z and Y is not known. This means that the integral has to be evaluated in an alternative way namely by using a uniformly distributed variable, U, or by utilizing that $Y = F_Y^{-1}(F_Z(Z))$. The details of the inverse probability distribution F^{-1} and how to approximate it can be found in [19].
The basis for the two approaches is that the probability distribution for Z and Y map the variables to a uniform distribution in $[0,1]$, i.e. $F_Y : I_Y \to [0,1]$ and $F_Z : I_Z \to [0,1]$.
The first approach utilizes the fact that Z and Y can be mapped into a uniform parameter, $U = F_Y(Y) = F_Z(Z) \sim \mathcal{U}(0,1)$, and that $Y = F_Y^{-1}(U)$ and $Z = F_Z^{-1}(U)$. This means that the coefficients can be computed as

$$a_i = \frac{1}{\gamma_i}\mathbb{E}_U[F_Y^{-1}(U)\Phi_i(F_Z^{-1}(U))] = \frac{1}{\gamma_i}\int_0^1 F_Y^{-1}(u)\Phi_i(F_Z^{-1}(u))du,$$

where $\mathbb{E}_U$ is the expectation operator for the random variable U.
The second approach is to use the fact that $Y = F_Y^{-1}(F_Z(Z))$ and to computed the coefficient as

$$a_i = \frac{1}{\gamma_i}\mathbb{E}_U[F_Y^{-1}(F_Z(Z))\Phi_i(Z)] = \frac{1}{\gamma_i}\int_0^1 F_Y^{-1}(F_Z(z))\Phi_i(z)dz.$$

The two new expressions for the coefficients are mathematical equivalent and can be approximated by e.g. Gauss quadrature. The results can be formulated in a theorem as in [19].

Theorem A.7 *Let Z and Y be a random variables with probability distribution $F_Z(z) = P(Z \le z)$ and $F_Y(y) = P(Y \le y)$, respectively and $\mathbb{E}[Y^2] < \infty$. Furthermore let $\mathbb{E}[|Z|^{2m}] < \infty$ for $\forall m \in \mathfrak{N}$ such that the gPC basis functions exist and $\mathbb{E}_Z[\Phi_i(Z)\Phi_j(Z)] = \delta_{i,j}\gamma_i$ for $\forall i,j \in \mathfrak{N}$. Let*

$$Y_N = \sum_{i=0}^{N} a_i \Phi_i(Z), \quad a_i = \frac{1}{\gamma_i}\mathbb{E}_U[F_Y^{-1}(F_Z(Z))\Phi_i(Z)]. \tag{A.5}$$

Then the weak approximation Y_N converges in probability, i.e. $Y_N \xrightarrow{P} Y$ for $N \to \infty$, and convergence in distribution, i.e. $Y_N \xrightarrow{d} Y$.

The proof of the theorem can be found in [19].

Appendix B

Matlab

This appendix contains the Matlab implementations used in the thesis. The implementations follows the theory and the algorithms described in the thesis.

B.1 Toolbox

B.1.1 Polynomials and quadratures

B.1.1.1 Implementation of polynomials

The Hermite polynomials are implemented as

```
function [HerPolEx,w] = HermitePol(x,n,type)
%
% Syntax:    [HerPolEx,w] = HermitePol(x,n,type)
%
% Purpose:  Compute the n'th order Hermite polynomial in
   the nodes x.
%
% input:
```

```
%            n        Order of the Hermite polynomial.
%
%            x        The nodes in which the Laguerre
    poynomial is to be
%                     evaluated.
%
%            type     The type of Hermite polynomial to be
    used. type = 1
%                     refers to He(x) which is the most
    relevant for this
%                     thesis.
%
%
% Output
%            HerPolEx    A vector containing Hermite
    polynomial of order n
%                        evaluated in the nodes x.
%

[a b] = size(x);
if a > b
    x = x';
end

if type == 1

    w = 1/sqrt(pi*2)*exp(-x.^2/2);

    m = [0:floor(n/2)]'; nMinus2m = n-2*m;
    fraction = (-1).^m./(factorial(m).*2.^m.*factorial(
        nMinus2m));

    XRep = (repmat(x,length(fraction),1))';

    XRepFixed = zeros(size(XRep));

    for i = 1 : length(fraction)
        XRepFixed(:,i) = XRep(:,i).^nMinus2m(i);
    end

    HerPolEx = factorial(n)*XRepFixed*fraction;

else
    w = 1/sqrt(pi)*exp(-x.^2);
    x2 = 2*x;
    m = [0:floor(n/2)]'; nMinus2m = n-2*m;
    fraction = (-1).^m./(factorial(m).*factorial(nMinus2m
        ));

    XRep = (repmat(x2,length(fraction),1))';

    XRepFixed = zeros(size(XRep));

    for i = 1 : length(fraction)
        XRepFixed(:,i) = XRep(:,i).^nMinus2m(i);
    end

    HerPolEx = factorial(n)*XRepFixed*fraction;

end
```

A recursive implementation have been made that returns Hermite all the Hermite polynomials in a range of polynomial orders. This has been implemented as

```
function [HerPolEx] = HermitePolRec(x,type,From,To)
%
% Syntax:    [HerPolEx] = HermitePolRec(x,type,From,To)
%
% Purpose:   Compute the Hermite polynomial from order
    From to order To in
%            the nodes x.
%
% input:
%            x        The nodes in which the polynomials
    are to be
%                     evaluated.
%
%            type     The type of Hermite polynomial to be
    used. type = 1
%                     refers to He(x) which is the most
    relevant for this
%                     thesis.
%
%            From     The lowest order polynomial to be
    evaluated.
%
```

```
%           To       The highest order polynomial to be
    evaluated.
%
% Output
%           HerPolEx    A matrix containing Hermite
    polynomials
%                       evaluated in the nodes x.
%

% Help variables
Dif = To-From+1;
xL = length(x);

% Initial Polynomial
HerPolEx = zeros(xL,Dif);
if From <=1
    k=1;
    if From < 1
        HerPolEx(:,1) = ones(xL,1);
        k=k+1;
    end
    if Dif>1
        if type ==1
            HerPolEx(:,k) = x;
        else
            HerPolEx(:,k) = 2*x;
        end
    end
else
    n = From+2;
    if Dif>=2
        [w,HerPolEx(:,1)] = HermitePol(x,From,type);
        [w,HerPolEx(:,2)] = HermitePol(x,From+1,type);
    else
        [w,HerPolEx(:,1)] = HermitePol(x,From,type);
    end
end
n = From+1;
if Dif>2
    for k = 3:Dif
        HerPolEx(:,k) = x'.*HerPolEx(:,k-1)-n*HerPolEx(:,
            k-2);
        n = n+1;
    end
```

```
end
```

The Legendre polynomials are implemented as

```
function [Le] = LegendrePol(x,n)
%
% Syntax:   [Le] = LegendrePol(x,n)
%
% Purpose:  Compute the n'th order Legendre polynomial in
    the nodes x.
%
% input:
%           n       Order of the polynomial.
%
%           x       The nodes in which the Legendre
   poynomial is to be
%                   evaluated.
%
% Output
%           Le      A vector containing Legendre
   polynomial of order n
%                   evaluated in the nodes x.
%
n2 = floor(n/2);

Le = zeros(size(x));
for l = 0:n2
    Le = Le + (-1).^l*nchoosek(n,l)*nchoosek((2*n-2*l),n)
       *x.^(n-2*l);
end

Le = 1/(2^n)*Le;
```

The Legendre polynomials can also be computed recursively which is implemented as

```
function [Le] = LegendrePolRec(x,Start,End)
%
% Syntax:   [Le] = LegendrePolRec(x,Start,End)
%
% Purpose:  Compute the Legendre polynomial from order
   From to order To in
%           the nodes x.
%
```

```
% input:
%           x       The nodes in which the polynomials
    are to be
%                   evaluated.
%
%           Start   The lowest order polynomial to be
    evaluated.
%
%           End     The highest order polynomial to be
    evaluated.
%
% Output
%           Le      A matrix containing Legendre
    polynomials
%                   evaluated in the nodes x.
%

Len = abs(End-Start)+1;

Le0 = ones(size(x));
Le1 = x;

Le = zeros(length(x),Len);
if Start == 0
    Le(:,1) = Le0;
    if Len >= 2
        Le(:,2) = Le1;
    end
elseif Start == 1
    Le(:,1) = Le1;
    if Len >= 2
        Le(:,2) = 1/2*(3*x^2-1);
    end
else
    Le(:,1) = LegendrePol(x,Start);

    if Len >=2
        Le(:,2) = LegendrePol(x,Start+1);
    end
end

if Len>=3
    for k = 2:Len-1
        n = Start+k-1;
```

```
            Le(:,k+1)=(2*n+1)/(n+1).*x'.*Le(:,k)-n/(n+1)*Le
                (:,k-1);
        end
end
```

The Laguerre polynomials are implemented as

```
function [Le] = LegendrePol(x,n)
%
% Syntax:    [Le] = LegendrePol(x,n)
%
% Purpose:   Compute the n'th order Legendre polynomial in
    the nodes x.
%
% input:
%            n        Order of the polynomial.
%
%            x        The nodes in which the Legendre
    poynomial is to be
%                     evaluated.
%
% Output
%            Le       A vector containing Legendre
    polynomial of order n
%                     evaluated in the nodes x.
%
n2 = floor(n/2);

Le = zeros(size(x));
for l = 0:n2
    Le = Le + (-1).^l*nchoosek(n,l)*nchoosek((2*n-2*l),n)
        *x.^(n-2*l);
end

Le = 1/(2^n)*Le;
```

A recursive computation of the Laguerre polynomials is implemented as

```
function [Le] = LegendrePolRec(x,Start,End)
%
% Syntax:    [Le] = LegendrePolRec(x,Start,End)
%
% Purpose:   Compute the Legendre polynomial from order
    From to order To in
```

```
%           the nodes x.
%
% input:
%           x       The nodes in which the polynomials
    are to be
%                   evaluated.
%
%           Start   The lowest order polynomial to be
    evaluated.
%
%           End     The highest order polynomial to be
    evaluated.
%
% Output
%           Le      A matrix containing Legendre
    polynomials
%                   evaluated in the nodes x.
%

Len = abs(End-Start)+1;

Le0 = ones(size(x));
Le1 = x;

Le = zeros(length(x),Len);
if Start == 0
    Le(:,1) = Le0;
    if Len >= 2
        Le(:,2) = Le1;
    end
elseif Start == 1
    Le(:,1) = Le1;
    if Len >= 2
        Le(:,2) = 1/2*(3*x^2-1);
    end
else
    Le(:,1) = LegendrePol(x,Start);

    if Len >=2
        Le(:,2) = LegendrePol(x,Start+1);
    end
end

if Len>=3
```

```
    for k = 2:Len-1
        n = Start+k-1;
        Le(:,k+1)=(2*n+1)/(n+1).*x'.*Le(:,k)-n/(n+1)*Le
            (:,k-1);
    end
end
```

The derivatives of the Jacobi polynomials are implemented as

```
function [Pm] = GradJacobiPoln(x,alpha,beta,N)
%
% Syntax:    [Pm] = GradJacobiPoln(x,alpha,beta,N)
%
% Purpose:  Compute the gradiant to the n'th order Jacobi
      polynomial
%           in the nodes x.
%
% input:
%           N        Order of the polynomial.
%
%           x        The nodes in which the Legendre
    poynomial is to be
%                    evaluated.
%
%           alpha    The alpha value used to define the
    Jacobi polynomial
%
%           beta     The beta value used to define the
    Jacobi polynomial
%
% Output
%           Pm       A vector containing gradiant
    polynomial evaluated in
%                    the nodes x.
%

if N==0;
    Pm = zeros(size(x));
else
    [P] = JacobiPolNorm(x,alpha+1,beta+1,N-1);
    Pm = zeros(length(x));
    for ni = 1:N
        Pm(:,ni+1) = sqrt(ni*(ni+alpha+beta+1))*P(:,ni);
    end
```

```
end
```

The normalized Jacobi polynomials are used and the implementation for these are given in

```
function [p] = JacobiPolNorm(x,alpha,beta,N)
%
% Syntax:    [p] = JacobiPolNorm(x,alpha,beta,N)
%
% Purpose:  Compute the the n'th order normalized Jacobi
    polynomial
%           in the nodes x.
%
% input:
%           N       Order of the polynomial.
%
%           x       The nodes in which the Legendre
    polynomial is to be
%                   evaluated.
%
%           alpha   The alpha value used to define the
    Jacobi polynomial
%
%           beta    The beta value used to define the
    Jacobi polynomial
%
% Output
%           p       A vector containing polynomial
    evaluated in
%                   the nodes x.
%

% Ensure that x is a column and not a row vector.
x = x(:);

P = zeros(length(x),N+1);
PN = P;

P(:,1) = 1;

gamma1 = 2^(alpha+beta+1)*factorial(0+alpha)*factorial(0+
    beta);
gamma2 = factorial(0)*(0+alpha+beta+1)*factorial(0+alpha+
    beta);
```

```
gammaInv0 = gamma2/gamma1;

PN(:,1) = sqrt(gammaInv0)*P(:,1);


if N == 0
      p = PN(:,1);
      return
elseif (N==-1)
      p = PN(:,1)*0;
      return
end

P(:,2) = 1/2*(alpha-beta) + 1/2*(alpha+beta+2)*x;

gamma1 = 2^(alpha+beta+1)*factorial(1+alpha)*factorial(1+
    beta);
gamma2 = factorial(1)*(2+alpha+beta+1)*factorial(1+alpha+
    beta);
gammaInv = gamma2/gamma1;

PN(:,2) = sqrt(gammaInv)*P(:,2);


if N ==1
      p = P;
      return;
end

for n = 1:N-1
      re = 2*n+alpha+beta; % Factor computed once and
          reused.
      Pn1 = 2*(n+alpha)*(n+beta)/((re+1)*re)*P(:,n);
      Pn = ((alpha^2-beta^2)/((re+2)*re) + x).*P(:,n+1);
      InvFac = ((re+2)*(re+1))/(2*(n+1)*(re-n+1));

      P(:,n+2) =  InvFac*(Pn - Pn1);

      n2 = n+1;

      gamma1 = 2^(alpha+beta+1)*factorial(n2+alpha)*
          factorial(n2+beta);
      gamma2 = factorial(n2)*(2*n2+alpha+beta+1)*factorial(
          n2+alpha+beta);
    gammaInv = gamma2/gamma1;

    size(P(:,n+2));
    size(PN);

    PN(:,n+2) = sqrt(gammaInv)*P(:,n+2);
end

p = PN;
end
```

B.1.1.2 Implementation of quadratures

A general function that can call the other quadrature functions has been implemented as

```
function [x,w] = Quadrature(n,type)
%
% Syntax:   [x,w] = Quadrature(n,type)
%
% Purpose:  Generate nodes and weights for different
    quadratures
%
% input:
%           n       Order of the polynomial. Which also
    means the
%                   number of nodes and weights.
%
%           type    Defines which kind of quadrature that
     is used and
%                   contains extra parameters if these
    are needed.
%
% Output
%           x       A vector containing the nodes (
    abscissas) of the
%                   Hermite polynomial.
%
%           w       A vector containing the weights
    corresponding to the
%                   computed nodes.
%
```

```

if strcmp(type(1),'hermite')

    [x,w] = HermiteQuad(n);

elseif strcmp(type(1),'laguerre')

    [x,w] = laguerrequad(n);

elseif strcmp(type(1),'legendre')

    [x,w] = legendrequad(n);

elseif strcmp(type(1),'legendreshift')

    [x,w] = legendrequad01(n);

elseif strcmp(type(1),'jacobi')

    [x,w] = JacobiGQ(type(2),type(3),n);
end
```

The Gauss Hermite quadrature is implemented as

```
function [x,w] = HermiteQuad(n)
%
% Syntax:    [x,w] = HermiteQuad(n)
%
% Purpose:   Generate nodes and weights for Gauss Hermite
    quadrature.
%
% input:
%            n       Order of the polynomial. Which also
    means the
%                    number of nodes and weights.
%
% Output
%            x       A vector containing the nodes (
    abscissas) of the
%                    Hermite polynomial.
%
%            w       A vector containing the weights
    corresponding to the
%                    computed nodes.
```

```
%

nV = 1:n-1;
 BetaSqrt = sqrt(nV./2);
J = diag(BetaSqrt,1)+diag(BetaSqrt,-1);
 [V,D] = eig(J);
 [x,ind] = sort(diag(D));
wTemp = sqrt(pi)*V(1,:).^2;
w = wTemp(ind);
```

The computation of the Gauss Legendre quadrature nodes and weights is implemented as

```
 function [x,w] = legendrequad(n)
%
% Syntax:    [x,w] = legendrequad(n)
%
% Purpose:  Generate nodes and weights for Gauss Legendre
     quadrature.
%
% input:
%           n       Order of the polynomial. Which also
    means the
%                   number of nodes and weights.
%
% Output
%           x       A vector containing the nodes (
    abscissas) of the
%                   Hermite polynomial.
%
%           w       A vector containing the weights
    corresponding to the
%                   computed nodes.
%
u = sqrt(1./(4-1./[1:n-1].^2)); % upper diag.
 [V,Lambda] = eig(diag(u,1)+diag(u,-1));
 [x,i] = sort(diag(Lambda));
 Vtop = V(1,:);
 Vtop = Vtop(i);
w = 2*Vtop.^2;
```

The implementation of the shifted Gauss Legendre quadrature nodes and weights is

```
function [x,w] = legendrequad01(n)
%
% Syntax:    [x,w] = legendrequad01(n)
%
% Purpose:   Generate nodes and weights for the shifted
    Gauss Legendre
%            quadrature i.e. in the interval [0,1].
%
% input:
%            n       Order of the polynomial. Which also
   means the
%                    number of nodes and weights.
%
% Output
%            x       A vector containing the nodes (
   abscissas) of the
%                    Hermite polynomial.
%
%            w       A vector containing the weights
   corresponding to the
%                    computed nodes.
%
a = repmat(1/2,1,n); % main diagonal of J
u = sqrt(1./(4*(4-1./[1:n-1].^2)));
[V,Lambda] = eig(diag(u,1)+diag(a)+diag(u,-1));
[x,i] = sort(diag(Lambda));
Vtop = V(1,:);
Vtop = Vtop(i);
w = Vtop.^2;
```

The Gauss Laguerre quadrature rule is implemented as

```
function [x,w] = laguerrequad(n)
%
% Syntax:    [x,w] = laguerrequad(n)
%
% Purpose:   Generate nodes and weights for Gauss Laguerre
     quadrature.
%
% input:
%            n       Order of the polynomial. Which also
   means the
%                    number of nodes and weights.
%
```

```
% Output
%           x       The nodes (abscissas) of the Hermite
    polynomial.
%
%           w       The weights corresponding to the
    computed
%                   nodes.
%

a = 2*[0:n-1]+1; % diagonal of J
u = [1:n-1]; % upper diagonal of J
[V,Lambda] = eig(diag(u,1)+diag(a)+diag(u,-1));
[x,i] = sort(diag(Lambda));
Vtop = V(1,:);
Vtop = Vtop(i);
w = Vtop.^2;
```

The Gauss-Lobatto nodes are implemented as

```
function [x] = JacobiGL(alpha,beta,N)

% function [x] = JacobiGL(alpha,beta,N)
% Purpose: Compute the N'th order Gauss Lobatto
    quadrature
%          points, x, associated with the Jacobi
    polynomial,
%          of type (alpha,beta) > -1 ( <> -0.5).

x = zeros(N+1,1);
if (N==1) x(1)=-1.0; x(2)=1.0; return; end;

[xint,w] = JacobiGQ(alpha+1,beta+1,N-2);
x = [-1, xint', 1]';
return;
```

The implementation of the Gauss-Lobatto nodes relies on the implementation of the Gauss Quadrature points implemented as

```
function [x,w] = JacobiGQ(alpha,beta,N)

% function [x,w] = JacobiGQ(alpha,beta,N)
% Purpose: Compute the N'th order Gauss quadrature points
    , x,
%          and weights, w, associated with the Jacobi
%           polynomial, of type (alpha,beta) > -1 ( <>
    -0.5).

if (N==0) x(1)=(alpha-beta)/(alpha+beta+2); w(1) = 2;
    return; end;

% Form symmetric matrix from recurrence.
J = zeros(N+1);
h1 = 2*(0:N)+alpha+beta;
J = diag(-1/2*(alpha^2-beta^2)./(h1+2)./h1) + ...
    diag(2./(h1(1:N)+2).*sqrt((1:N).*((1:N)+alpha+beta)
         .*...
    ((1:N)+alpha).*((1:N)+beta)./(h1(1:N)+1)./(h1(1:N)+3)
         ),1);
if (alpha+beta<10*eps) J(1,1)=0.0;end;
J = J + J';

%Compute quadrature by eigenvalue solve
[V,D] = eig(J); x = diag(D);
w = (V(1,:)').^2*2^(alpha+beta+1)/(alpha+beta+1)*gamma(
    alpha+1)*...
    gamma(beta+1)/gamma(alpha+beta+1);
return;
```

B.1.2 Vandermonde-like matrices.

The implementation of the Vandermonde-like matrix V is given by

```
function D = JacobiDn(x,alpha,beta)

% Compute the generalized Vandermonde matrices V and Vx
N = length(x);
%V = JacobiPolNorm(x,alpha,beta,N-1);

V = JacobiPolNorm(x,alpha,beta,N-1);
Vx = GradJacobiPoln(x,alpha,beta,N-1);

% The inverse matrix of V is computed
D = Vx/V;
```

B.1.3 ERK

The implementation of the explicit Runge-Kutta method is given here

```
function U = ERK(t,U,fun,dt,param)

G = U;
P = feval(fun,t,U,param);
U = U + 0.5*dt*P;
G = P;
P = feval(fun,t+0.5*dt,U,param);
U = U + 0.5*dt*(P-G);
G = (1/6)*G;
P = feval(fun,t+0.5*dt,U,param) - 0.5*P;
U = U + dt*P;
G = G - P;
P = feval(fun,t+dt,U,param) +2*P;
U = U + dt*(G + (1/6)*P);
```

B.2 Test Equation

B.2.1 Monte Carlo

A test script for the Monte Carlo simulation for the Test Equation has been implemented as

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% Monte Carlo Script
                                                  %%%%%%
%%%%%% Version 1.0

    %%%%%%
%%%%%% Author: Emil Brandt Kaergaard
                                         %%%%%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


```

```
% Parameters: The size of the dataset, the weight of the
    solutions,
% the number of datasets, the type of distribution and
    parameters specific
% for the distribution.

close all; clear all;


% Number of realizations, time-endpoint
N = [100 1000 5000 10000 30000 50000 80000 100000 150000
    300000 500000 800000 1000000];

Ns = 1./sqrt(N);
tend = 1; tN = 45;

% Computing the parameters for generating the random
    variables.
mu = 0; sigma = sqrt(0.25);
Alpha.DistType = 'normal';
Alpha.Dim = [1,1];
Alpha.Par = [mu,sigma];

Beta = 1;
RealSol = 2;
MuEx = exp(tend^2/2);

% Pre-allocation
Mu = zeros(tN,length(N));
Sigma = zeros(tN,length(N));
MuExact = zeros(tN,length(N));
SigmaExact = zeros(tN,length(N));
MuErr = zeros(tN,length(N));
SigErr = zeros(tN,length(N));
Time = zeros(1,length(N));

for k =1:length(N)
    [Mu(:,k),Sigma(:,k),CI,Exact,Err,Time(k)] =
        MCTestEqSolv(Alpha,Beta,tend,tN,N(k),RealSol);
    MuExact(:,k) = Exact.MuExact;
    SigmaExact(:,k) = Exact.SigmaExact;
    MuErr(:,k) = Err.MuErr;
    SigErr(:,k) = Err.SigErr;
    Time(k)
```

```
end

t = linspace(0,tend,tN);

figure(1)
hold on
plot(t,Mu(:,end),'b-*',t,CI.UB,'k-*',t,CI.LB,'k-*')
plot(t,MuExact,'r-o',t,Exact.UBExact,'g-o',t,Exact.
    LBExact,'g-o')
hold off
```

Much of the computations have been implemented in

```
function [Mu,Sigma,CI,Exact,Err,Time,t] = MCTestEqSolv(
    Alpha,Beta,tend,tN,M,RealSol)
%
% Function for using Monte Carlo method on Test Equation.
% Version 1.0
% Author: Emil Brandt Kaergaard
%
% Syntax:
%           [Mu,Sigma,CI,Exact,Err] = MCTestEqSolv(Alpha,
    Beta,tend,tN,M)
%
%

% Computing the parameters alpha and beta.
alpha = RandVar(Alpha.DistType,Alpha.Dim,M,Alpha.Par);
if length(Beta) == 1
    beta = Beta*ones(1,M);
else
    beta = RandVar(Beta.DistType,Beta.Dim,M,Beta.Par);
end

% Computing the tspan and preallocating a matrix U for
    the solutions u for
% the deterministic solution of each realization of the (
    pseudo-)random parameters.
tspan = linspace(0,tend,tN);
dt = abs(tspan(2)-tspan(1));
U = zeros(length(tspan),M);

if RealSol == 1
    t = tspan';
    tic
    for i=1:M
        U(:,i) = exp(-alpha(i).*t);
    end
    Time = toc;
elseif RealSol == 0
    tic
    for i=1:M
        [t,U(:,i)] = ode45(@TestEq,tspan,beta(i),[],alpha
            (i));
    end
    Time = toc;

else
    t = zeros(size(tspan));
    tic
    for i=1:M
        U(1,i)=beta(i);
        for ti = 2:length(tspan)
            U(ti,i) = ERK(tspan(ti),U(ti-1,i),@TestEq,dt,
                alpha(i));
            t(ti)=t(ti-1)+dt;
        end
    end
    Time = toc;
    t = t';
end
% Computing modes: Mean and variance.
Mu = sum(U,2)/M;

Sig = zeros(length(tspan),M);
for k = 1:M
    Sig(:,k) = (U(:,k)-Mu).^2;
end

Sigma = sum(Sig,2)/(M-1);

UB = Mu + 1.96*sqrt(Sigma)/sqrt(M);
LB = Mu - 1.96*sqrt(Sigma)/sqrt(M);

CI.UB = UB;
CI.LB = LB;

% Coumpitng exact values if these are known.
```

```
if strcmp(Alpha.DistType,'normal')
    MuExact = exp(1/2.*(Alpha.Par(2).*t).^2-Alpha.Par(1)
        .*t);
    SigmaExact = exp(2*(Alpha.Par(2).*t).^2-2*Alpha.Par
        (1).*t)-exp((Alpha.Par(2).*t).^2-2*Alpha.Par(1).*t
        );

    UBExact = MuExact + 1.96*sqrt(SigmaExact)/sqrt(M);
    LBExact = MuExact - 1.96*sqrt(SigmaExact)/sqrt(M);

    MuErr = abs(Mu - MuExact);
    SigErr = abs(Sigma - SigmaExact);
elseif strcmp(Alpha.DistType,'uniform')
    MuExact = -(exp(-(Alpha.Par(2).*t))-exp(-(Alpha.Par
        (1).*t)))./(t.*(Alpha.Par(2)-Alpha.Par(1)));
    SigmaExact = -(exp(-2*(Alpha.Par(2).*t))-exp(-2*(
        Alpha.Par(1).*t)))./(2*t.*(Alpha.Par(2)-Alpha.Par
        (1)))-MuExact.^2;

    UBExact = MuExact + 1.96*sqrt(SigmaExact)/sqrt(M);
    LBExact = MuExact - 1.96*sqrt(SigmaExact)/sqrt(M);

    MuErr = abs(Mu - MuExact);
    SigErr = abs(Sigma - SigmaExact);

else

    MuExact = 0;
    SigmaExact = 0;

    UBExact = 0;
    LBExact = 0;

    MuErr = 0;
    SigErr = 0;
end

Exact.MuExact = MuExact;
Exact.SigmaExact = SigmaExact;
Exact.UBExact = UBExact;
Exact.LBExact = LBExact;

Err.MuErr = MuErr;
Err.SigErr = SigErr;
```

The random variables are computed by using the following implementation

```
function X = RandVar(DistType,Dim,M,Par)
%
% Random Variabel Script
% Version 1.0
% Date: December 2012
% Author: Emil Brandt Kaergaard
%
%  RandVar:
%            This function computes M random variables of
   dimension Dim. The
%            type of distribution used to generate the
   variables is defined
%            in DistType and by the parameters in Par.
%
% Syntax:
%            X = RandVar(DistType,Dim,M,Par)
%
% Input:
%            DistType: The type of distribution used to
   generate the
%                      variables.
%            Dim: Dimension of the variables.
%            M: Number of random variables.
%            Par: The parameters of the chosen
   distribution of the variables.
%
% Output:
%            X: A Dim x M matrix containing M (pseudo-)
   random variables of dimension Dim.
%


% Defining row- and column-length.
row = Dim(1);
column = Dim(2);

% Initialize parameters
if length(Par) == 1 % Use default distribution parameters
    .
    if strcmp(DistType,'normal')
        mu = 0;
        sigma = 1;
    elseif strcmp(DistType,'uniform')
        a = -1;
        b = 1;
    end
elseif length(Par) == 2 % Use user-defined parameters
    if strcmp(DistType,'normal')
        mu = Par(1);
        sigma = Par(2);
    elseif strcmp(DistType,'uniform')
        a = Par(1);
        b = Par(2);
    end
end

% Generating random samples

if strcmp(DistType,'normal')
    X = mu + sigma.*randn(row,column,M);
elseif strcmp(DistType,'uniform')
    X = a + (b-a).*rand(row,column,M);
else
    error('Wrong type of distribution')
end
```

The rhs-function has been implemented as

```
function du = TestEq(t,u,alpha)
%
% Test Equation Script.
% Version 1.0
% Date: December 2012
% Author: Emil Brandt Kaergaard
%
%  TestEq:
%
% Syntax:
%           du = TestEq(t,u,alpha)
%
% Input:
%
% Output:
%

du = -alpha.*u;
```

B.2.2 Stochastic Collocation Method

A test script computing following the pseudocode 5 have been implemented as

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Stochastic Collocation method on the Test Equation.      %%%%%%%%
%%%%%%%% Version 1.0                                              %%%%%%%%
%%%%%%%% Date: February 2012                                      %%%%%%%%
%%%%%%%% Author: Emil Brandt Kaergaard                            %%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Constants:
zN = 9;
dt = 0.05;

tBeg = 0; tEnd = 1;
tspan = tBeg:dt:tEnd;

DistType = 'uniform';

if strcmp(DistType,'normal')
    % Gaussian Distribution: Hermite Gauss Quadrature nodes.
    mu = 0; Var = 1; sigma = sqrt(Var);
    par1  = mu; par2 = sigma;

    [z,zW] = HermiteQuadN(zN); % Computing nodes and weights

    z = mu+sigma*z; % Computing the N(mu,sigma) distributed alpha-parameter
    n = 0:zN-1;
 elseif strcmp(DistType,'uniform')
    % Uniform Distribution: Legendre Gauss Quadrature nodes.
    [z,zW] = legendrequad(zN); % Computing nodes and
        weights
    zW = 1/2*zW;
    a = -1; b=1;
    par1  = a; par2 = b;
end

% Pre-allocation and initial condition.
U = zeros(zN,length(tspan));
U(:,1) = ones(zN,1);

% Solving deterministic system.
tic
for ti = 1:length(tspan)-1
    U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,z);
end
time = toc;

figure
plot(tspan,U')

[Umean, Uvar, MuExact, SigmaExact, MuErr, SigErr] =
    TestEqX(DistType,par1,par2,U,zW,zN,tspan);

figure
hold on
plot(tspan,Umean(end,:),'b')
plot(tspan,MuExact,'r-.')
hold off


%%

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Computing errors for different zN for SCM on the
    Test Equation. %%%%%
%%%%% Version 1.0

    %%%%%
%%%%% Date: February 2012
                                                    %%%%%
```

```
%%%%% Author: Emil Brandt Kaergaard                                %%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants:
ZN = 16; % Maximum number of nodes and weights in z.
dt = 0.005;

tBeg = 0; tEnd = 1;
tspan = tBeg:dt:tEnd;

DistType = 'normal'; % Choice of distribution of z.

MuErr = zeros(ZN,length(tspan));
SigErr = MuErr;

for zN = 1:ZN

    if strcmp(DistType,'normal')
        % Gaussian Distribution: Hermite Gauss Quadrature nodes.
        mu = 0; Var = 1; sigma = sqrt(Var);
        par1  = mu; par2 = sigma;

        [z,zW] = HermiteQuadN(zN);

        z = mu+sigma*z;
        n = 0:zN-1;
    elseif strcmp(DistType,'uniform')
        % Uniform Distribution: Legendre Gauss Quadrature nodes.
        alpha = 0; beta = 0;
        [z,zW] = JacobiGQ(alpha,beta,zN-1);

        zW = 1/2*zW;
        a = -1; b=1;
        par1  = a; par2 = b;
    end

    % Pre-allocation and initial condition.
    U = zeros(zN,length(tspan));
    U(:,1) = ones(zN,1);

    % Solving the deterministic system.
    tic
    for ti = 1:length(tspan)-1
        U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,
            z);
    end
    time = toc;

    [Umean, Uvar, MuExact, SigmaExact, MuErr(zN,:),
        SigErr(zN,:)] = TestEqX(DistType,par1,par2,U,zW,zN
        ,tspan);
end

figure
semilogy(1:ZN,MuErr(:,end),'b.',1:ZN,SigErr(:,end),'ro');

figure
semilogy(1:ZN,SigErr(:,end));
```

The rhs-function has been implemented as

```
function un = rhsSCMtest(t,U,alpha)

un = alpha.*U;
```

The approximated and exact statistics are computed in `TestEqX` which is implemented as

```
function [Umean, Uvar, MuExact, SigmaExact, MuErr, SigErr
    ] = TestEqX(DistType,par1,par2,U,zW,zN,tspan)

t = tspan;

% Computing mean:
Uw = zeros(zN,length(tspan));
for zi = 1:zN
    Uw(zi,:) = U(zi,:)*zW(zi);
end
Umean = sum(Uw,1);

% Computing variance:
Uv=zeros(length(tspan),zN);
for zi = 1:zN
```

```
    Uv(:,zi) = (U(zi,:)-Umean(end,:)).^2*zW(zi);
end
Uvar = sum(Uv,2); Uvar = Uvar';

% Computing exact mean and variance
if strcmp(DistType,'normal')
    MuExact = exp(1/2.*(par2.*t).^2-par1.*t);
    SigmaExact = exp(2*(par2.*t).^2-2*par1.*t)-exp((par2.*t).^2-2*par1.*t);

elseif strcmp(DistType,'uniform')
    MuExact = -(exp(-(par2.*t))-exp(-(par1.*t)))./(t.*(par2-par1));
    SigmaExact = -(exp(-2*(par2.*t))-exp(-2*(par1.*t)))./(2*t.*(par2-par1))-MuExact.^2;
end

% Computing errors on mean and variance
MuErr = abs(Umean - MuExact);
SigErr = abs(Uvar - SigmaExact);
```

B.2.3 Stochastic Galerkin method

A test script computing following the pseudocode 5 have been implemented as

```

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% Stochastic Galerkin Method: Test Equation %%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



% Distribution parameters
DistType = 'normal';
mu = 0; var = 1;
sigma = sqrt(var);
par1 = mu; par2 = sigma;

```

```
beta = 1; % Initial condition.


% Initial parameters:

N = 4;
a = zeros(N+1,1);
a(1) = mu;
a(2) = sigma;

b = zeros(N+1,1);
b(1) = beta;

% Parameters (the matrix A) for the solver:
A = zeros(N+1,N+1);
for j = 0:N
    for k=0:N
        [e,gamma] = TestGalPar(j,k,N);
        A(j+1,k+1) = -1/gamma*(a'*e);
    end
end

% tspan and rhs for the solver.
dt = 0.01;
tStart = 0;
tEnd = 1;
tspan = tStart:dt:tEnd;

rhsTestEq = @(t,v) A'*v;

[t,V] = ode45(rhsTestEq,tspan,b);


% Using ERK:

% rhs for ERK
rhsTestEqERK = @(t,v,A) (A'*v')';

% Allocation
U = zeros(length(tspan),length(b));
MuExact = zeros(length(tspan),1);
U(1,:)=b;
tspan2 = zeros(size(tspan));

```

```
tspan2(1) = tStart;
MuExact(1,:) = (exp(1/2.*(sigma.*tspan2(1)).^2-mu.*tspan2(1)))';

for ti = 2:length(tspan)
    U(ti,:) = ERK(tspan2(ti-1),U(ti-1,:),rhsTestEqERK,dt,A);
    tspan2(ti) = tspan2(ti-1)+dt;
    MuExact(ti) = exp(1/2.*(sigma.*tspan2(ti)).^2-mu.*tspan2(ti));

end

[Uvar, MuExact, SigmaExact, MuErr, SigErr] = TestEqX(DistType,par1,par2,U,N,tspan,V(:,1));

figure
hold on
plot(t,MuExact)
plot(t,U(:,1),'rx')
plot(t,V(:,1),'k-.')
hold off


figure
hold on
plot(t,MuExact)
plot(t,U(:,1),'rx')
plot(t,U(:,1)+Uvar,'r--')
plot(t,U(:,1)-Uvar,'r--')
hold off


%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%% Stochastic Galerkin Method: Test Equation %%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
% Distribution parameters
DistType = 'normal';
if strcmp(DistType,'normal')
    mu = 0; var = 1;
    sigma = sqrt(var);
    par1 = mu; par2 = sigma;
elseif strcmp(DistType,'uniform')
    a = -1; b = 1;
    par1 = a; par2 = b;
    mu = 1/2*(a+b); sigma = 1/2*(b-a);
end

beta = 1; % Initial condition.


% tspan and rhs for the solver.
dt = 0.01;
tStart = 0; tEnd = 1;
tspan = tStart:dt:tEnd;

% Initial parameters:

Nmax = 14;
MuErr = zeros(length(tspan),Nmax);
SigErr = MuErr;
for N = 1:Nmax;

    a = zeros(N+1,1);
    a(1) = mu; a(2) = sigma;

    b = zeros(N+1,1);
    b(1) = beta;

    % Parameters (the matrix A) for the solver:
    A = zeros(N+1,N+1);
    for j = 0:N
        for k=0:N
            [e,gamma] = TestGalParDT(j,k,N,DistType);
            A(j+1,k+1) = -1/gamma*(a'*e);
        end
    end

    rhsTestEq = @(t,v) A'*v;
    [t,U] = ode45(rhsTestEq,tspan,b);

    [Uvar, MuExact, SigmaExact, MuErr(:,N), SigErr(:,N)]
        = TestEqX(DistType,par1,par2,U,N,tspan);
end

figure
semilogy(1:Nmax,MuErr(end,:),'b.',1:Nmax,SigErr(end,:),'
    ro');

figure
hold on
plot(t,MuExact)
plot(t,U(:,1),'rx')
hold off

figure
hold on
plot(t,SigmaExact)
plot(t,Uvar,'r--')
hold off

znV = 1:Nmax;


ErrGalMN = [znV',MuErr(end,:)',SigErr(end,:)'];
dlmwrite('ErrGalMN.txt', ErrGalMN, 'delimiter', '\t','
    precision', 10, 'newline', 'pc')
```

The approximated and exact statistics are computed in `TestEqX` which is implemented as

```
function [Uvar, MuExact, SigmaExact, MuErr, SigErr] =
    TestEqX(DistType,par1,par2,U,N,tspan)

t = tspan;
Umean = U(:,1);


% Computing variance:
if strcmp(DistType,'normal')
    Uv = zeros(size(U));
    for vi = 1:N
        gamma = factorial(vi);
        Uv(:,vi+1) = gamma.*U(:,vi+1).^2;
    end
    Uvar = sum(Uv,2);
elseif strcmp(DistType,'uniform')
    Uv = zeros(size(U));
    for vi = 1:N
        gamma = 1/(2*vi+1);
        Uv(:,vi+1) = gamma.*U(:,vi+1).^2;
    end
    Uvar = sum(Uv,2);
end


% Computing exact mean and variance
if strcmp(DistType,'normal')
    MuExact = exp(1/2.*(par2.*t).^2-par1.*t);
    SigmaExact = exp(2*(par2.*t).^2-2*par1.*t)-exp((par2
        .*t).^2-2*par1.*t);

elseif strcmp(DistType,'uniform')
    MuExact = -(exp(-(par2.*t))-exp(-(par1.*t)))./(t.*(
        par2-par1));
    SigmaExact = -(exp(-2*(par2.*t))-exp(-2*(par1.*t)))
        ./(2*t.*(par2-par1))-MuExact.^2;
end

% Computing errors on mean and variance
MuErr = abs(Umean - MuExact');
SigErr = abs(Uvar - SigmaExact');
```

B.3 Burgers' Equation

B.3.1 Stochastic Collocation Method

The following function generates the initial condition used by SCM for Burgers' Equation.

```
function [InitCond,U,z,zW] = InitBurg(zN,nu,x)

ck=0; xL = length(x);
dStart = 0.0; dEnd = 0.1;
```

```

Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

[z,zW] = legendrequad(zN);

% Scale weights and disturbances.
delta = ((z+1)/2)*(dEnd-dStart)+dStart;
zW = zW/2;

InitCond = Uexact(x,0,nu);

U = zeros(xL*zN,1);
for i = 1:zN
     InitTemp = InitCond;
     InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta(
         i));
     U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

This function generates the Gauss-Lobatto nodes in the x-vector and the parameters for the right-hand-side (rhs) function.

```
function [x,param,paramKron] = xParBurg(alpha,beta,xN,zN,
     nu)
%
%
%

[x] = JacobiGL(alpha,beta,xN);
D = JacobiDn(x,alpha,beta);
D2 = D*D;
xL = length(x);

% Saving the parameters in two structs.
param.D = D; param.D2 = D2; param.nu = nu; param.zN = zN;
     param.xL = xL;

kronD = kron(speye(zN),D);
kronD2 = kron(speye(zN),D2);

paramKron.D = kronD; paramKron.D2 = kronD2;
paramKron.nu = nu; paramKron.zN = zN; paramKron.xL = xL;
```

The rhs function is implemented as

```
function [un] = rhsColBurg(t,U,param)

% Renaming input-parameters
nu = param.nu; D = param.D; D2 = param.D2;
xL = param.xL; zN = param.zN;

% Pre-allocation
un = zeros(size(U));
% Compute solutions
for k =0:zN-1
    un(xL*k+1:xL*(k+1),1) = nu*D2*U(xL*k+1:xL*(k+1),1)-U(
        xL*k+1:xL*(k+1),1).*(D*U(xL*k+1:xL*(k+1),1));
    un(xL*k+1,1) = 0;
    un(xL*(k+1),1) = 0;
end
```

The deterministic solution to Burgers' equation is computed in the following implementation

```
function [U,t,time] = BurgDetSolv(U,dt,param,ErrorBar,
    MaxIter)


t = zeros(MaxIter,1);

iter = 0;
diff = 1;

tic
while diff > ErrorBar && iter <= MaxIter
    UTemp = U;
    U = ERK(t(iter+1),U,@rhsColBurg,dt,param);
    t(iter+2) = t(iter+1)+dt;
    diff = max(abs(UTemp-U));
    iter = iter+1;
end
time = toc;
```

The script for solving the Stochastic Burgers' equation is

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%% Stochastic Collocation Method on Burgers' Equation.     %%%%%%%%
%%%%%%%% Author: Emil Brandt Kaergaard                           %%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants:
a = 0.0; nu = 0.05;
alpha = 0; beta = 0;
xN = 45;
zN = 10;
ErrorBar = 0.2e-5;
MaxIter = 5000000;

dt = 0.0005;

% The parameters for rhs and the Gauss-Lobatto nodes.
[x,param,paramKron] = xParBurg(alpha,beta,xN,zN,nu);

% Initialcondition
[InitCond,U,z,zW] = InitBurg(zN,nu,x);

% Solve the deterministic system
[Usol,t,time] = BurgDetSolv(U,dt,param,ErrorBar,MaxIter);

xL=param.xL;

figure
hold on
for zi = 0:zN-1
    plot(x,Usol(zi*xL+1:xL*(zi+1),1),'b-')
end
hold off

xRep = repmat(x,1,zN);
UsRep = reshape(Usol,xL,zN);
figure
plot(xRep,UsRep)

% Computing statistics.
UTemp=zeros(xL,zN);
for zi = 0:zN-1
```

```
    UTemp(:,zi+1) = Usol(zi*xL+1:xL*(zi+1))*zW(zi+1);
end
Umean = sum(UTemp,2);

UvTemp=zeros(xL,zN);
for zi = 0:zN-1
    UvTemp(:,zi+1) = (Usol(zi*xL+1:xL*(zi+1))-Umean).^2*
        zW(zi+1);
end

Uvar = sum(UvTemp,2);

yL = Umean-sqrt(Uvar);
yU = Umean+sqrt(Uvar);
X = [x' fliplr(x')];
Y = [yU' fliplr(yL')];


figure
hold on
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
fill(X,Y,'g')
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
legend('mean','std','bound')

figure
hold on
%fill(X,Y,'g')
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
legend('mean','std','bound')
```

B.3.2 Stochastic Galerkin method

B.4 Burgers' Equation 2D

The following function generates the initial condition used by SCM for Burgers' Equation.

```
function [InitCond,U,Z,ZW] = InitBurg2D(zN,alpha,beta,nu,
    x,dStart,dEnd)

ck=0; xL = length(x);

Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

[z1,z1W] = JacobiGQ(alpha,beta,zN-1);
[z2,z2W] = JacobiGQ(alpha,beta,zN-1);

% Scaling the weights
z1W = z1W/2;
z2W = z2W/2;

% Scaling the disturbances.
z1 = ((z1+1)/2)*(dEnd(1)-dStart(1))+dStart(1);
z2 = ((z2+1)/2)*(dEnd(2)-dStart(2))+dStart(2);

[Z1,Z2] = ndgrid(z1,z2);
[Z1W,Z2W] = ndgrid(z1W,z2W);

Z1 = Z1(:); Z2 = Z2(:);
Z = [Z1 Z2];

Z1W = Z1W(:); Z2W = Z2W(:);
ZW = Z1W.*Z2W;

% Initialization and pre-allocation.
delta1 = Z1;
delta2 = Z2;
ZN = length(Z1);
InitCond = Uexact(x,0,nu);
U = zeros(xL*ZN,1);

```

```
for i = 1:ZN
     InitTemp = InitCond;
     InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta1
         (i));
     InitTemp(InitTemp<0) = InitTemp(InitTemp<0)*(1+delta2
         (i));
     U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

This function generates the Gauss-Lobatto nodes in the x-vector and the parameters for the right-hand-side (rhs) function.

```
function [x,param,paramKron] = xParBurg(alpha,beta,xN,zN,
     nu)

[x] = JacobiGL(alpha,beta,xN);
D = JacobiDn(x,alpha,beta);
D2 = D*D;
xL = length(x);

% Saving parameters in structs.
param.D = D; param.D2 = D2; param.nu = nu; param.zN = zN;
     param.xL = xL;

kronD = kron(speye(zN),D);
kronD2 = kron(speye(zN),D2);

Nu=repmat(param.nu',xL,1); Nu=Nu(:);

paramKron.D = kronD; paramKron.D2 = kronD2; paramKron.nu
    = nu;
paramKron.Nu = Nu; paramKron.zN = zN; paramKron.xL = xL;
```

The rhs function is implemented as

```
function [un] = rhsColBurg(t,U,param)

% Input parameters are renamed
nu = param.nu;
D = param.D;
D2 = param.D2;
xL = param.xL;
zN = param.zN;

```

```
% Pre-allocation
un = zeros(size(U));

% Rhs is evaluated for each random variable
for k =0:zN-1
     un(xL*k+1:xL*(k+1),1) = nu.*D2*U(xL*k+1:xL*(k+1),1)-U
         (xL*k+1:xL*(k+1),1).*(D*U(xL*k+1:xL*(k+1),1));
     un(xL*k+1,1) = 0;
     un(xL*(k+1),1) = 0;
end
```

The deterministic solution to Burgers' equation is computed in the following implementation

```
function [U,t,time] = BurgDetSolv(U,dt,param,ErrorBar,
    MaxIter)


t = zeros(MaxIter,1);

iter = 0;
diff = 1;

tic
while diff > ErrorBar && iter <= MaxIter
     UTemp = U;
     U = ERK(t(iter+1),U,@rhsColBurg,dt,param);
     t(iter+2) = t(iter+1)+dt;
     diff = max(abs(UTemp-U));
     iter = iter+1;
end
time = toc;
```

The script for solving the Stochastic Burgers' equation is

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% Burgers Equation: Collocation method with random
    BC's %%%%%%%%%%%%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```matlab
% Constants:
nu = 0.05;
alpha = 0; beta = 0;
xN = 30;
zN = 3; ZN = zN^2;
ErrorBar = 0.8e-5;
MaxIter = 5000000;

dStart = [0,0];
dEnd = [0.1,0.1];

dt = 0.0001;

% The parameters for rhs and the Gauss-Lobatto nodes.
[x,param,paramKron] = xParBurg(alpha,beta,xN,ZN,nu);

%dt = 0.95*abs(x(2)-x(1))^2/(2*nu);

% Initial condition
[InitCond,U,z,zW] = InitBurg2D(zN,alpha,beta,nu,x,dStart,
    dEnd);

% Deterministic solution
[Usol,t,time] = BurgDetSolv(U,dt,param,ErrorBar,MaxIter);

xL=param.xL;

xRep = repmat(x,1,ZN);
UsRep = reshape(Usol,xL,ZN);
figure
plot(xRep,UsRep)

% Computing statistics.
UTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UTemp(:,zi+1) = Usol(zi*xL+1:xL*(zi+1))*zW(zi+1);
end
Umean = sum(UTemp,2);

UvTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UvTemp(:,zi+1) = (Usol(zi*xL+1:xL*(zi+1))-Umean).^2*
        zW(zi+1);
end
```

```

Uvar = sum(UvTemp,2);

yL = Umean-sqrt(Uvar);
yU = Umean+sqrt(Uvar);
X = [x' fliplr(x')];
Y = [yU' fliplr(yL')];

% Plotting results
figure
hold on
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
fill(X,Y,'g')
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
legend('mean','std','bound')

figure
hold on
%fill(X,Y,'g')
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
plot(x,Ulow,'k-','linewidth',2)
plot(x,Uup,'k-','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
legend('mean','std','bound')
```

B.5 Burgers' Equation 3D

The following function generates the initial condition used by SCM for Burgers' Equation.

```
function [InitCond,U,Z,ZW] = InitBurg3D(zN,alpha,beta,nu,
    x,dStart,dEnd)
```

```

ck=0; xL = length(x);

Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

% Computing the quadrature weights and nodes for the
    random BC's
[z1,z1W] = legendrequad(zN);
[z2,z2W] = legendrequad(zN);

% Scaling the weights
z1W = z1W/2;
z2W = z2W/2;

% Scaling the disturbances.
z1 = ((z1+1)/2)*(dEnd(1)-dStart(1))+dStart(1);
z2 = ((z2+1)/2)*(dEnd(2)-dStart(2))+dStart(2);

% Computing the nodes and weights for the nu paremeter
if strcmp(nu.t,'uniform')
    [z3,z3W] = JacobiGQ(alpha,beta,zN-1);

    % Scaling the weights and nu
    z3W = z3W/2;
    z3 = ((z3+1)/2)*(nu.par(2)-nu.par(1))+nu.par(1);

elseif strcmp(nu.t,'normal')
    [z3,z3W] = HermiteQuadN(zN);

    % Scaling nu
    z3 = nu.par(1)+nu.par(2)*z3;
end
%z3 = 0.05; z3W = 1;
[Z1,Z2,Z3] = ndgrid(z1,z2,z3);
[Z1W,Z2W,Z3W] = ndgrid(z1W,z2W,z3W);

Z1 = Z1(:); Z2 = Z2(:); Z3 = Z3(:);
Z = [Z1 Z2 Z3];

Z1W = Z1W(:); Z2W = Z2W(:); Z3W = Z3W(:);
ZW = Z1W.*Z2W.*Z3W;

% Initialization and pre-allocation.
delta1 = Z1; delta2 = Z2; ZN = length(Z1);
```

```
nuEst = sum(z3)/zN;
InitCond = Uexact(x,0,nuEst);
U = zeros(xL*ZN,1);

for i = 1:ZN
    InitTemp = InitCond;
    InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta1
        (i));
    InitTemp(InitTemp<0) = InitTemp(InitTemp<0)*(1+delta2
        (i));
    U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

This function generates the Gauss-Lobatto nodes in the x-vector and the parameters for the right-hand-side (rhs) function.

```
function [x,param,paramKron] = xParBurg3D(alpha,beta,xN,
    zN)
%nu = 0.05; alpha = 0; beta = 0; N = 50;

[x] = JacobiGL(alpha,beta,xN);
D = JacobiDn(x,alpha,beta);
D2 = D*D;
xL = length(x);

param.D = D; param.D2 = D2; param.zN = zN; param.xL = xL;

kronD = kron(speye(zN),D);
kronD2 = kron(speye(zN),D2);

paramKron.D = kronD; paramKron.D2 = kronD2;
paramKron.nu = nu; paramKron.zN = zN; paramKron.xL = xL;
```

The rhs function is implemented as

```
function [un] = rhsColBurg3Ds(t,U,param)

% Input parameters are renamed
nu = param.nu;
D = param.D;
D2 = param.D2;
xL = param.xL;
zN = param.zN;
NUD2 = param.NUD2;
```

```matlab

% Pre-allocation
%un = zeros(size(U));

% Rhs is evaluated for each random variable
%for k =0:zN-1
%    un(xL*k+1:xL*(k+1),1) = nu(k+1)*D2*U(xL*k+1:xL*(k+1)
    ,1)-U(xL*k+1:xL*(k+1),1).*(D*U(xL*k+1:xL*(k+1),1));
%    un(xL*k+1,1) = 0;
%    un(xL*(k+1),1) = 0;
%end
%uns1 = D2*U;
Nu=repmat(param.nu',xL,1); Nu=Nu(:);

uns2 = U.*(D*U);
uns3 = NUD2*U;
un = uns3-uns2;

for i = 0:zN-1
      un(i*xL+1) = 0;
      un((i+1)*xL) = 0;
end
```

The deterministic solution to Burgers' equation is computed in the following implementation

```matlab
function [U,t,time] = BurgDetSolv3D(U,dt,param,ErrorBar,
    MaxIter)


t = zeros(MaxIter,1);

iter = 0;
diff = 1;

tic
while diff > ErrorBar && iter <= MaxIter
    UTemp = U;
    U = ERK(t(iter+1),U,@rhsColBurg3Ds,dt,param);
    t(iter+2) = t(iter+1)+dt;
    diff = max(abs(UTemp-U));
    iter = iter+1;
end
time = toc;
```

The script for testing the Stochastic Burgers' equation is

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Burgers Equation: Collocation method with random BC's and random nu %%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants:
nu.t = 'uniform'; nu.par = [0.05 0.051];
alpha = 0; beta = 0;
xN = 40;
zN = 3; ZN = zN^3;
ErrorBar = 0.1e-5;
MaxIter = 5000000;

dStart = [0,0];
dEnd = [0.1,0.1];

dt = 0.00002;


% The parameters for rhs and the Gauss-Lobatto nodes.
[x,param,paramKron] = xParBurg(alpha,beta,xN,ZN,nu);

% Initial Condition
[InitCond,U,z,zW] = InitBurg3D(zN,alpha,beta,nu,x,dStart,dEnd);

param.nu = z(:,3); paramKron.nu = z(:,3);

NUdiag = spdiags(param.nu,0,length(param.nu),length(param.nu));
NU = kron(NUdiag,ones(size(param.D)));

NUD2 = NU.*paramKron.D2;
paramKron.NUD2 = NUD2;

param.zN = length(z(:,3));
ZN = length(z(:,3));

```

```
[Usol,t,time] = BurgDetSolv3D(U,dt,paramKron,ErrorBar,
    MaxIter);

xL=param.xL;

% Computing statistics.
UTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UTemp(:,zi+1) = Usol(zi*xL+1:xL*(zi+1))*zW(zi+1);
end
Umean = sum(UTemp,2);

UvTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UvTemp(:,zi+1) = (Usol(zi*xL+1:xL*(zi+1))-Umean).^2*
        zW(zi+1);
end

Uvar = sum(UvTemp,2);

yL = Umean-sqrt(Uvar);
yU = Umean+sqrt(Uvar);
X = [x' fliplr(x')];
Y = [yU' fliplr(yL')];

% Plotting the solutions
xRep = repmat(x,1,ZN);
UsRep = reshape(Usol,xL,ZN);
figure
plot(xRep,UsRep)
xlabel('x')
ylabel('u')


figure
hold on
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
fill(X,Y,'g')
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
```

```
legend('mean','std','bound')

figure
hold on
plot(x,Umean,'r-','linewidth',2)
plot(x,sqrt(Uvar),'b-.','linewidth',2)
plot(x,Ulow,'k-','linewidth',2)
plot(x,Uup,'k-','linewidth',2)
hold off
xlabel('x','fontsize',12)
ylabel('u','fontsize',12)
legend('mean','std','bound')
```

B.6 Numerical tests with sparse grids

B.6.1 Numerical tests for the multivariate Test equation

The script for testing the sparse grid is implemented as

```
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Test-equation: Multidimensional Collocation Method
    SCL %%%%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Constants:
dt = 0.05;

tBeg = 0; tEnd = 1;
tspan = tBeg:dt:tEnd;

d=2;
level = 2;

AlpDist = 'uniform';
BetDist = 'uniform';

% Parameters for computing the random variables alpha
```

```
a=-1; b = 1;
AlpPar = [a,b];

% Parameters for computing the random variable beta
a=0; b = 2;
BetPar = [a,b];

% Computing the number of grid points
Pn = sparse_grid_gl_size (d, level);

% Computing sparse grid
[gw, gp] = sparse_grid_gl (d, level, Pn);

gp = gp';
z1 = gp(:,1); z2 = gp(:,2);

% Scaling the disturbances.
Alp = ((z1+1)/2)*(AlpPar(2)-AlpPar(1))+AlpPar(1);
Bet = ((z2+1)/2)*(BetPar(2)-BetPar(1))+BetPar(1);

% Scaling the weights.
ZW = gw/(2^d);

AN = length(gw);

% Solving deterministic system.
U = zeros(AN,length(tspan)); % Pre-allocation.
U(:,1) = Bet; % Initial Condition

for ti = 1:length(tspan)-1
    U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,Alp)
        ;
end

% Computing the statistics.
[Umean, Uvar] = TestEqStat(U,ZW,AN,tspan);

% Computing the exact mean and variance as well as the
    errors.
[MuExact, SigmaExact, MuErr, SigErr] = TestEqExact(
    AlpDist,AlpPar,BetDist,BetPar,tspan,Umean,Uvar);

figure
plot(tspan,Umean,'b-')
```

```
figure
plot(tspan,Uvar,'r-')

figure
plot(tspan,Umean,'b-',tspan,MuExact,'r--')
figure
plot(tspan,Uvar,'b-',tspan,SigmaExact,'r--')
%%

%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Test-equation: Multidimensional Collocation Method
    CC %%%%%
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Constants:
dt = 0.05;

tBeg = 0; tEnd = 1;
tspan = tBeg:dt:tEnd;

d=2;
level = 5;

AlpDist = 'uniform';
BetDist = 'uniform';

% Parameters for computing the random variables alpha
a=-1; b = 1;
AlpPar = [a,b];

% Parameters for computing the random variable beta
a=0; b = 2;
BetPar = [a,b];

% Computing the number of grid points
Pn= sparse_grid_cc_size_old(d, level);

% Computing sparse grid
[gw, gp] = sparse_grid_cc(d,level,Pn);

```

```matlab
gw = gw'; gp = gp';
z1 = gp(:,1); z2 = gp(:,2);


% Scaling the disturbances.
Alp = ((z1+1)/2)*(AlpPar(2)-AlpPar(1))+AlpPar(1);
Bet = ((z2+1)/2)*(BetPar(2)-BetPar(1))+BetPar(1);

Z = [Alp Bet];

% Scaling the weights.
ZW = gw/2^d;

AN = length(gw);

% Solving deterministic system.
U = zeros(AN,length(tspan)); % Pre-allocation.
U(:,1) = Bet; % Initial Condition

for ti = 1:length(tspan)-1
    U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,Alp)
        ;
end

% Computing the statistics.
[Umean, Uvar] = TestEqStat(U,ZW,AN,tspan);

% Computing the exact mean and variance as well as the
    errors.
[MuExact, SigmaExact, MuErr, SigErr] = TestEqExact(
    AlpDist,AlpPar,BetDist,BetPar,tspan,Umean,Uvar);

figure
plot(tspan,Umean,'b-')
figure
plot(tspan,Uvar,'r-')

figure
plot(tspan,Umean,'b-',tspan,MuExact,'r--')
figure
plot(tspan,Uvar,'b-',tspan,SigmaExact,'r--')

%%

```

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%% Test-equation: Multidimensional Collocation Method - error plot %%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants:
dt = 0.005;

tBeg = 0; tEnd = 1;
tspan = tBeg:dt:tEnd;

d=2;

AlpDist = 'uniform';
BetDist = 'uniform';

% Parameters for computing the random variables alpha
a=-1; b = 1;
AlpPar = [a,b];

% Parameters for computing the random variable beta
a=0; b = 2;
BetPar = [a,b];


% Initialization and pre-allocation.
aN = 1:12; aN = [aN 20 50 80];
ANV = zeros(size(aN));
UM = zeros(length(tspan),length(aN));
UV = UM; MuE = UM; SigE = UM;
Time=zeros(length(aN),1);

% Looping to compute the errors for different full tensor grids.
for i = 1:length(aN)
    alphaN = aN(i); betaN = aN(i);

    % Computing the random variables alpha and beta
    [alpha,alphaW]= RandVar(AlpPar(1),AlpPar(2),alphaN, AlpDist);
    [beta,betaW]= RandVar(BetPar(1),BetPar(2),betaN,
        BetDist);


    % Computing the grid of random variables
    [Alpha,Beta] = meshgrid(alpha,beta);
    [AlphaW,BetaW] = meshgrid(alphaW,betaW);

    % Vectorizing the matrices with the meshgrids of
        variables and weights.
    AlpW = AlphaW(:); BetW = BetaW(:);
    Alp = Alpha(:); Bet = Beta(:);
    AN = length(Alp);

    ANV(i) = AN;

    % Computing the product of the weights which will be
        used for computing the
    % statistics of the solution.
    ABW = AlpW.*BetW;

    % Solving deterministic system.
    U = zeros(AN,length(tspan)); % Pre-allocation.
    U(:,1) = Bet; % Initial Condition
    tic
    for ti = 1:length(tspan)-1
        U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,
            Alp);
    end
    time = toc;
    Time(i,1) = time;

    % Computing the statistics.
    [Umean, Uvar] = TestEqStat(U,ABW,AN,tspan);
    UM(:,i) = Umean';
    UV(:,i) = Uvar';

    % Computing the exact mean and variance as well as
        the errors.
    [MuExact, SigmaExact, MuErr, SigErr] = TestEqExact(
        AlpDist,AlpPar,BetDist,BetPar,tspan,Umean,Uvar);
    MuE(:,i) = MuErr';
    SigE(:,i) = SigErr';
end
```

```



Level = 0:6;

% Pre-allocation for CC grid
UMCC = zeros(length(tspan),length(Level));
UVCC = UMCC;
MuECC = UMCC;
SigECC = UMCC;
CCN = zeros(length(Level),1);
TimeCC = zeros(length(Level),1);


% Pre-allocation for SGL grid
UMSGL = UMCC;
UVSGL = UMCC;
MuESGL = UMCC;
SigESGL = UMCC;
SGLN = zeros(length(Level),1);
TimeSGL= zeros(length(Level),1);

for i = 1:length(Level)
    level = Level(i);

    %%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%% CC grid  %%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%


    % Computing the number of grid points
    Pn= sparse_grid_cc_size_old(d, level);

    % Computing sparse CC grid
    [gw, gp] = sparse_grid_cc(d,level ,Pn);

    gw = gw'; gp = gp';
    z1 = gp(:,1); z2 = gp(:,2);

    % Scaling the disturbances.
    Alp = ((z1+1)/2)*(AlpPar(2)-AlpPar(1))+AlpPar(1);
    Bet = ((z2+1)/2)*(BetPar(2)-BetPar(1))+BetPar(1);

    % Scaling the weights.
    ZW = gw/2^d;

    AN = length(gw);

    CCN(i) = AN;

    % Solving deterministic system.
    U = zeros(AN,length(tspan)); % Pre-allocation.
    U(:,1) = Bet; % Initial Condition
    tic
    for ti = 1:length(tspan)-1
        U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,
            Alp);
    end
    time = toc;

    TimeCC(i,1) = time;
    % Computing the statistics.
    [Umean, Uvar] = TestEqStat(U,ZW,AN,tspan);
    UMCC(:,i) = Umean';
    UVCC(:,i) = Uvar';

    % Computing the exact mean and variance as well as
        the errors.
    [MuExact, SigmaExact, MuErr, SigErr] = TestEqExact(
        AlpDist,AlpPar,BetDist,BetPar,tspan,Umean,Uvar);
    MuECC(:,i) = MuErr';
    SigECC(:,i) = SigErr';


    %%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%% SGL grid %%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%

    % Computing the number of grid points
    Pn = sparse_grid_gl_size (d, level);

    % Computing sparse grid
    [gw, gp] = sparse_grid_gl (d, level, Pn);

    gp = gp';
    z1 = gp(:,1); z2 = gp(:,2);

    % Scaling the disturbances.
```

```matlab
        Alp = ((z1+1)/2)*(AlpPar(2)-AlpPar(1))+AlpPar(1);
        Bet = ((z2+1)/2)*(BetPar(2)-BetPar(1))+BetPar(1);

        Z = [Alp Bet];

        % Scaling the weights.
        ZW = gw/2^d;

        AN = length(gw);

        SGLN(i) = AN;

        % Solving deterministic system.
        U = zeros(AN,length(tspan)); % Pre-allocation.
        U(:,1) = Bet; % Initial Condition

        tic
        for ti = 1:length(tspan)-1
            U(:,ti+1) = ERK(tspan(ti),U(:,ti),@rhsSCMtest,dt,
                Alp);
        end
        time = toc;

        TimeSGL(i,1) = time;

        % Computing the statistics.
        [Umean, Uvar] = TestEqStat(U,ZW,AN,tspan);
        UMSGL(:,i) = Umean';
        UVSGL(:,i) = Uvar';

        % Computing the exact mean and variance as well as
            the errors.
        [MuExact, SigmaExact, MuErr, SigErr] = TestEqExact(
            AlpDist,AlpPar,BetDist,BetPar,tspan,Umean,Uvar);
        MuESGL(:,i) = MuErr';
        SigESGL(:,i) = SigErr';

end
```

The sparse grid functions can be found at [2] and the rest of the functions have been included previously.

B.6.2 Numerical tests for the multivariate Burgers' equation

The script for testing the sparse grid is implemented as

```
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%% Burgers Equation: Collocation method with sparse grids %%%%%%%%%
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Constants:
nu = 0.05;
alpha = 0; beta = 0;
xN = 30;
ErrorBar = 0.1e-6;
MaxIter = 5000000;

level = 2;

ZN = sparse_grid_gl_size (d, level);

dStart = [0,0];
dEnd = [0.1,0.1];


dt = 0.002;

% The parameters for rhs and the Gauss-Lobatto nodes.
[x,param,paramKron] = xParBurg(alpha,beta,xN,ZN,nu);

% Initial condition
[InitCond,U,z,zW] = InitBurg2DSparse(level,nu,x,dStart,
    dEnd);

% The determinstic solutions are computed
[Usol,t,time] = BurgDetSolv(U,dt,param,ErrorBar,MaxIter);

xL=param.xL;

```

```

% Computing the mean.
UTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UTemp(:,zi+1) = Usol(zi*xL+1:xL*(zi+1))*zW(zi+1);
end
Umean = sum(UTemp,2);

% Computing the variance
UvTemp=zeros(xL,ZN);
for zi = 0:ZN-1
    UvTemp(:,zi+1) = (Usol(zi*xL+1:xL*(zi+1))-Umean).^2*
        zW(zi+1);
end

Uvar = sum(UvTemp,2);
```

The script for computing the initial condition is implemented as

```
function [InitCond,U,Z,ZW] = InitBurg2DSparse(level,nu,x,
     dStart,dEnd)

d = 2;
ck=0; xL = length(x);
Uexact = @(x,t,nu) -tanh((x-t)/(2*nu))+ck;

% Computing the number of grid points
Pn = sparse_grid_gl_size (d, level);

% Computing sparse grid
[gw, gp] = sparse_grid_gl (d, level, Pn);

gw = gw'; gp = gp';
z1 = gp(:,1); z2 = gp(:,2);


% Scaling the disturbances.
z1 = ((z1+1)/2)*(dEnd(1)-dStart(1))+dStart(1);
z2 = ((z2+1)/2)*(dEnd(2)-dStart(2))+dStart(2);

Z1 = z1; Z2 = z2;
Z = [Z1 Z2];

% Scaling the weights
```

```
ZW = gw/2^d;

% Initialization and pre-allocation.
delta1 = Z1;
delta2 = Z2;
ZN = length(Z1);
InitCond = Uexact(x,0,nu);
U = zeros(xL*ZN,1);

for i = 1:ZN
    InitTemp = InitCond;
    InitTemp(InitTemp>0) = InitTemp(InitTemp>0)*(1+delta1
        (i));
    InitTemp(InitTemp<0) = InitTemp(InitTemp<0)*(1-delta2
        (i));
    U(xL*(i-1)+1:xL*(i)) = InitTemp;
end
```

The sparse grid functions can be found at [2] and the rest of the functions have been included previously.

Bibliography

[1] P. Beckmann. Orthogonal polynomials for engineers and physicists. 1973.

[2] J. Burkardt. Sparse grids based on the gauss-legendre rule. `http://people.sc.fsu.edu/~jburkardt/m_src/sparse_grid_gl/sparse_grid_gl.html`.

[3] J. Burkardt. Slides: Sparse grid collocation for uncertainty quantification. 2012.

[4] J. Burkardt and C. Webster. Slow exponential growth for clenshaw curtis sparse grids. September 2012.

[5] P. G. Constantine, M. S. Eldred, and E. T. Phipps. Sparse pseudospectral approximation method. *Computer Methods in Applied Mechanics and Engineering*, 229–232, 2012.

[6] A. Doostan and H. Owhadi. A non-adapted sparse approximation of pdes with stochastic inputs. *Journal of Computational Physics*, 230(8):3015–3034, 2011.

[7] A. P. Engsig-Karup. Slides for dtu course 02689 in spectral methods. 2011.

[8] Y. Filmus. Two proofs of the central limit theorem. February 2010.

[9] J. Foo and G. E. Karniadakis. Multi-element probabilistic collocation method in high dimensions. *Journal of Computational Physics*, 229(5):1536–1557, 2010.

[10] J. A. Gubner. Gaussian quadrature and the eigenvalue problem. September 2009.

[11] F. Heiss and V. Winschel. Quadrature on sparse grids: Code to generate and readily evaluated nodes and weights. `http://www.sparse-grids.de/`, 2007.

[12] J. Jakeman. Polynomial chaos, uncertainty quantification. 2008.

[13] A. Klimke. Sparse Grid Interpolation Toolbox – user's guide. Technical Report IANS report 2007/017, University of Stuttgart, 2007.

[14] A. Klimke and B. Wohlmuth. Algorithm 847: spinterp: Piecewise multilinear hierarchical sparse grid interpolation in MATLAB. *ACM Transactions on Mathematical Software*, 31(4), 2005.

[15] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations, Algorithms for scientists and Engineers.* Springer, 2009.

[16] O. Le Maître and O. Knio. *Spectral Methods for Uncertainty Quantification.* Springer, 2010.

[17] E. Morey. Joint density functions, marginal density functions, conditional density functions, expectations and independence. 2002.

[18] M. R. Spiegel and J. Liu. *Mathematical Handbook of formulas and tables.* Schaum's, 1999.

[19] D. Xiu. *Numerical methods for stochastic computations: a spectral method approach.* Princeton University Press, 2010.

[20] D. Xiu and J. S. Hesthaven. High-order collocation methods for differential equations with random inputs. *SIAM Journal on Scientific Computing (SISC)*, 27(3):1118–1139, 2005.

[21] L. Yan, L. Guo, and D. Xiu. Stochastic collocation algorithms using ℓ_1–minimization. *International Journal for Uncertainty Quantification*, 2(3):279–293, 2012.

[22] X. Yang, M. Choi, G. Lin, and G. E. Karniadakis. Adaptive anova decomposition of stochastic incompressible and compressible flows. *Journal of Computational Physics*, 231(4):1587–1614, 2012.

[23] Z. Zhang, M. Choi, and G. E. Karniadakis. Error estimates for the anova method with polynomial chaos interpolation: Tensor product functions. *SIAM Journal on Scientific Computing (SISC)*, 34(2):A1165—A1186, 2012.