A non-intrusive Galerkin method for PDEs with random data based on adaptive sparse grids

Andreas Doll

Master thesis

supervised by

Prof. Dr. Volker JOHN
Dr. Martin EIGEL

submitted to

Freie Universität Berlin
Fachbereich für Mathematik und Informatik
Berlin, Germany

on November 6, 2015

Abstract

This thesis deals with non-intrusive solution methods of Galerkin systems arising from stochastic PDE discretizations. We review the classic (intrusive) stochastic FEM method and discuss two non-intrusive techniques, the simpler discrete projection method and a more elaborate low-rank tensor approximation. The latter technique is coupled with error estimators to adaptivly generate spatial and stochastic refinement. Both methods rely on high-dimensional quadrature which is realized with dimension adaptive sparse grid methods, which are introduced along with implementation details.

Contents

1. Introduction

Partial differential equations (PDE) which describe natural or technical systems are of great interest in many scientific areas like physics, chemistry, biology, medicine, engineering, economics and many more. Numerical sciences has made great progress in the past decades to increase the accuracy of solutions. However, there are situations where uncertainty of input data can easily outweigh elaborate numerical algorithms.

Often the data of a system has a certain degree of uncertainty. For example, data may be obtained by measurements which naturally have measurement errors. Or there might be a parameter of the system which is only known by upper and lower bounds. We distinguish *epistemic uncertainty*, arising from incomplete knowledge of the system (like the examples above), and *aleatoric uncertainty*, arising from phenomena which cannot be quantified further by observation (like unexpected fluctuations of wind speed) [GWZ14]. A simple approach is to ignore the uncertainty and use averaged quantities. This is only feasible if changing the parameter has neglectable impact on the solution. If this is not the case, more elaborate techniques have to be employed.

Techniques which aim to quantify the effects of such uncertainty are termed *Uncertainty Quantification* (UQ) and have received much interest lately. One major branch of UQ are *probabilistic methods*, which aim to model uncertainty as stochastic entities by the statistical characterization of the input data. See [GWZ14], [Zan12] and references therein for a more detailed overview over UQ methods.

In this work we will use another major technique, namely *random field expansions*, which model uncertainty as a random field. The resulting system is a stochastical partial differential equation (SPDE). The focus of this thesis are *non-intrusive methods*, an umbrella term for methods which rely on the reusage of existing solver code. Often there is solver software available which solves the problem for given parametric values. It is evident that the solutions for different parametric values can be obtained independently of each other (and possibly in parallel), which is also termed *uncoupled*. In contrast, methods labeled as *intrusive* do not allow the incorporation of parametric solver software, or where the solver software requires some degree of rewriting in order to use it for solving the stochastic problem. Often, *coupled* methods arising from Galerkin of Rayleigh-Ritz discretizations are labeled intrusive, which is not always the case. In this thesis we consider Galerkin type SPDE discretizations and solve them non-intrusively [GLL$^+$14],[EGSZ15].

The outline of this work is as follows. In Chapter 2 we introduce a model problem along with random fields and their discretizations. Then we review the stochastic Finite Element Method, the classic but intrusive method for solving SPDEs. In Chapter 3 we discuss two basic non-intrusive methods for creating an approximation of the SPDE solution. Chapter 4 introduces a low-rank tensor approximation, for which we will discuss two construction methods. Both are non-intrusive methods, one is a more general method, the other is limited to linear problems but possesses a more elaborate update step. Since all non-intrusive techniques rely on high-dimensional quadrature, we introduce in Chapter 5 sparse grid methods, and more efficient dimension-adaptive methods along with implementation details. In Chapter 6 numerical results are discussed before drawing a conclusion and providing an outlook in Chapter 7.

I would like to thank my supervisors Prof. Dr. Volker John and Dr. Martin Eigel who provided much insight in numerous meetings. I am grateful for the competent help and code of Dipl.-Math. Johannes Neumann regarding parallelization.

Moreover i would like to thank Mag. Christoph Walzer who raised my interest in mathematics, and everyone else who supported and encouraged me on this way. Danke!

2. Stochastic finite element method

Here we introduce the *stochastic finite element method* (SFEM), a method for solving a stochastic boundary value problem which was first introduced in [GS91]. The method involves a eigenvalue problem and leads to a large coupled Galerkin system. It is an intrusive method, making no use of a deterministic solver.

2.1. Boundary value problem with random data

2.1.1. Model problem

Throughout this thesis we denote by $\mathbb{N}$ the naturals without zero and write $\mathbb{N}_0$ if we want to include it.

Definition 2.1. *Denote by $D \subset \mathbb{R}^d$ a domain with boundary ∂D. The data consists of the random field $\kappa : D \to \mathbb{R}$ and $f : D \to \mathbb{R}$, both smooth in C^2. The deterministic boundary value problem with homogeneous boundary conditions seeks the solution $u : \bar{D} \to \mathbb{R}$ such that*

$$-\nabla \cdot (\kappa(x)\nabla u(x)) = f(x), \quad x \in D, \tag{2.1}$$

$$u(x) = 0, \quad x \in \partial D. \tag{2.2}$$

We derive a weak formulation by choosing an appropriate function space $W := H_0^1(D)$ (compare Def. A.4) in which the solution is sought. Multiplication with a test function $v \in W$, followed by integration over the domain and using Greens first Theorem A.2 leads to the problem of finding $u \in W$ such that

$$\forall v \in W : \quad a(u,v) = \ell(v)$$

with

$$a(u,v) = \int_D \kappa \nabla u \cdot \nabla v \,\mathrm{d}x, \tag{2.3}$$

$$\ell(v) = \int_D f v \,\mathrm{d}x. \tag{2.4}$$

We transform the deterministic problem (2.1), (2.2) into a stochastic one by letting κ, f become random fields.

Definition 2.2. *Let $(\Omega, \mathfrak{A}, \mu)$ be a complete probability space with set of outcomes Ω, σ-algebra $\mathfrak{A} \subset 2^\Omega$ and probability measure $\mu : \mathfrak{A} \to [0,1]$. A random field $\kappa : D \times \Omega \to \mathbb{R}$ is defined such that for all $x \in D$, the map $\kappa(x,\cdot)$ is a random variable with respect to $(\Omega, \mathfrak{A}, \mu)$. In other words: for each realization $\omega \in \Omega$, the random field $\kappa(x,\omega)$ is a real valued random variable for each $x \in D$ (and analogously for f).*

Definition 2.3. *Let $\mathcal{L}$ be a possible nonlinear elliptic operator such that*

$$\mathcal{L}(\kappa)(u) = f \quad \text{in } D \tag{2.5}$$

for $u : \bar{D} \times \Omega \to \mathbb{R}$, a random field κ as above and suitable boundary conditions. Let $W(D)$ be a Banach space consisting of functions $v : D \to \mathbb{R}$, we define the space

$$L^q_\mu(\Omega; W(D)) := \left\{ v : \Omega \to W(D) \mid v \text{ is strongly measurable}, \int_\Omega \|v(\cdot,\omega)\|^q_{W(D)} \,\mathrm{d}\mu(\omega) < \infty \right\}$$

for $q \in [1, \infty)$.

A concrete example of such an operator $\mathcal{L}$ is given in the following stochastic model problem.

Definition 2.4. *Let $\kappa : D \times \Omega \to \mathbb{R}$ be a random field which is continuously differentiable in $\bar{D}$ and μ-almost surely uniformly bounded from above and below such that*

$$\exists a_{min}, a_{max} \in (0, \infty) : \; \mu\left(\omega \in \Omega \mid \forall x \in \bar{D} : \kappa(x,\omega) \in [a_{min}, a_{max}]\right) = 1,$$

and $f : D \times \Omega \to \mathbb{R}$ square integrable with respect to μ, that is

$$\int_D \mathbb{E}[f^2] \,\mathrm{d}x = \int_D \int_\Omega f^2(x,\omega) \,\mathrm{d}\mu(\omega) \,\mathrm{d}x < \infty.$$

The stochastic boundary value problem seeks a solution $u : \bar{D} \times \Omega \to \mathbb{R}$ such that

$$-\nabla \cdot (\kappa(x,\omega)\nabla u(x)) = f(x,\omega), \quad x \in D, \tag{2.6}$$

$$u(x) = 0, \quad x \in \partial D \tag{2.7}$$

with $\omega \in \Omega$.

The assumptions guarantee that for $W = H^1_0(D)$ the operator (2.5) has realizations in the Banach space W. In other words, μ-almost surely it holds $u(\cdot,\omega) \in W$, and for all $\omega \in \Omega$ it holds

$$\|u(\cdot,\omega)\|_W \le \tilde{C} \|f(\cdot,\omega)\|_{W^*}$$

with W^* denotes the dual of W and $\tilde{C}$ a constant independent of ω. Moreover the assumptions imply that $f \in L^2_\mu(\Omega, W^*(D))$ is such that the solution u is uniquely defined and bounded in $L^2_\mu(\Omega; W(D))$ [EEU07], [GWZ14, Part 2].

2.1.2. Karhunen–Loève expansion

The Karhunen–Loève expansion is a popular technique for modeling the random field. As we will see in Chapter 2.2.1, the SFEM discretizes the variables $\boldsymbol{x}, \omega$ in (2.6) independently of each other. The Karhunen–Loève expansion approximates the random field κ as a sum of terms only depending in those variables.

We start with some stochastic terminology.

Definition 2.5. *Let $X : \Omega \to \mathbb{R}$ be a random variable, we define its mean as*

$$\langle X \rangle := \int_\Omega X(\omega)\,\mathrm{d}\mu(\omega)$$

and the mean (or expectation) of the random field κ at any point $x \in D$ by

$$\bar{\kappa}(x) := \langle \kappa(x,\cdot) \rangle = \int_\Omega \kappa(x,\omega)\,\mathrm{d}\mu(\omega). \tag{2.8}$$

The covariance of κ at any $x, y \in D$ is defined to be

$$Cov_\kappa(x,y) := \langle (\kappa(x,\cdot) - \bar{\kappa}(x)) \cdot (\kappa(y,\cdot) - \bar{\kappa}(y)) \rangle$$

and the variance of κ at any x is

$$\mathrm{Var}_\kappa(x) := Cov_\kappa(x,x) = \left\langle (\kappa(x,\cdot) - \bar{\kappa}(x))^2 \right\rangle$$

and finally the standard deviation at x is

$$\sigma_\kappa(x) := \sqrt{\mathrm{Var}_\kappa(x)}.$$

We introduce the space of square integrable random variables

$$L^2_\mu := L^2(\Omega, \mathfrak{A}, \mu) = \left\{ X : \Omega \to \mathbb{R} \text{ measurable}, \int_\Omega X^2(\omega)\,\mathrm{d}\mu(\omega) < \infty \right\}$$

equipped with inner product

$$\langle X, Y \rangle_{L^2_\mu} := \langle XY \rangle = \int_\Omega X(\omega) Y(\omega)\,\mathrm{d}\mu(\omega)$$

for $X, Y \in L^2_\mu$.

Now we are able to make some assumptions on the random field κ. We assume that it is *homogeneous*, i.e., that the random field is invariant under coordinate shift, or equivalently: the covariance of the random field depends only on the translation $\boldsymbol{\tau} = x - y$. That means the covariance only depends on relative rather than absolute locations, which implies $\bar{\kappa} \equiv$ constant. We further assume it to be *isotropic*, i.e., that the random field is invariant under coordinate rotation, or equivalently: the covariance of the random field depends only on the distance $r := \|\boldsymbol{\tau}\| = \|\boldsymbol{x} - \boldsymbol{y}\|$ [Van84, Chapter 2.2].

Consider the bounded and real-valued operator

$$\begin{aligned} C \colon L^2(D) &\longrightarrow L^2(D) \\ u(y) &\longmapsto \int_D \mathrm{Cov}_\kappa(x,y) u(x)\,\mathrm{d}x, \end{aligned} \tag{2.9}$$

with associated eigenproblem

$$\int_D \operatorname{Cov}_\kappa(x,y)\kappa_j(x)\,\mathrm{d}x = \lambda_j\kappa_j(y) \tag{2.10}$$

with eigenvalues λ_j and eigenfunction κ_j. The operator $\operatorname{Cov}_\kappa$ is symmetric and positive definite, such an operator can be expanded as a sum of eigenpairs of κ converging in $L^2(D\times D)$ as

$$\operatorname{Cov}_\kappa(x,y) = \sum_{j=0}^{\infty} \lambda_j\kappa_j(x)\kappa_j(y). \tag{2.11}$$

The eigenfunctions form an orthonormal basis of $L^2(D)$ such that

$$\int_D \kappa_j(x)\kappa_k(x)\,\mathrm{d}x = \delta_{jk} \tag{2.12}$$

using the Kronecker delta δ_{jk}. For continuous covariance of κ, Mercers Theorem [GWZ14, Thm. B.1] yields absolute and uniform convergence of (2.11) in $D\times D$. The operator $\operatorname{Cov}_\kappa$ possesses countable many eigenvalues which have accumulation point zero. Hence we may assume the eigenvalues to be ordered decreasingly. The decay rate depends on the choice of covariance kernel and increases with correlation length, compare [GWZ14, Thm B.2, B.3]. [EEU07][GWZ14, Part 2, Appendix B]

We may decompose the random field as

$$\kappa(x,\omega) = \bar{\kappa}(x) + \theta(x,\omega) \tag{2.13}$$

for a stochastic process $\theta(x,\omega)$ with zero mean and covariance equal to the covariance of κ.

Lemma 2.6. *The expansion*

$$\theta(x,\omega) = \sum_{j=0}^{\infty} \sqrt{\lambda_j}\kappa_j(x)\xi_j(\omega) \tag{2.14}$$

satisfies $\bar{\theta}(x,\omega) = 0$ *and* $\operatorname{Cov}_\theta(x,y) = \operatorname{Cov}_\kappa(x,y)$.

Proof. The first claim $\bar{\theta}(x,\omega) = 0$ follows directly from (2.13). For the second claim we multiply (2.14) with $\theta(y,\omega)$ and take the mean, which yields

$$\begin{aligned}\operatorname{Cov}_\theta(x,y) &= \langle \theta(x,\omega)\theta(y,\omega)\rangle \\ &= \sum_{j=0}^{\infty}\sum_{k=0}^{\infty} \sqrt{\lambda_j\lambda_k}\kappa_j(x)\kappa_k(y)\langle \xi_j(\omega)\xi_k(\omega)\rangle.\end{aligned} \tag{2.15}$$

Multiplication of (2.15) with $\kappa_m(y)$, integrating over the domain along y and using (2.10) and the orthonormalization of the eigenfunctions (2.12) yields

$$\begin{aligned}\lambda_m\kappa_m(x) &= \int_D \mathrm{Cov}_\kappa(x,y)\kappa_m(y)\,\mathrm{d}y \\ &= \sum_{j=0}^{\infty}\sum_{k=0}^{\infty}\sqrt{\lambda_j\lambda_k}\langle\xi_j(\omega)\xi_k(\omega)\rangle\kappa_j(x)\int_D\kappa_k(y)\kappa_m(y)\,\mathrm{d}y \\ &= \sum_{j=0}^{\infty}\sqrt{\lambda_j\lambda_m}\langle\xi_j(\omega)\xi_m(\omega)\rangle\kappa_j(x).\end{aligned} \tag{2.16}$$

Similarly we multiply (2.16) with $\kappa_n(x)$ and integrate along x, so it holds

$$\begin{aligned}\lambda_m\delta_{mn} &= \int_D \lambda_m\kappa_m(x)\kappa_n(x)\,\mathrm{d}x \\ &= \sqrt{\lambda_m\lambda_n}\langle\xi_m(\omega)\xi_n(\omega)\rangle\end{aligned} \tag{2.17}$$

which is equivalent to

$$\delta_{mn} = \langle\xi_m(\omega)\xi_n(\omega)\rangle. \tag{2.18}$$

By inserting (2.18) into the covariance of θ (2.15) we see that it is equal to the spectral decomposition of the covariance of κ equation (2.11).

□

Definition 2.7. *The Karhunen–Loève expansion of a random field κ is given by*

$$\kappa(x,\omega) = \bar{\kappa}(x) + \sum_{j=1}^{\infty}\sqrt{\lambda_j}\kappa_j(x)\xi_j(\omega). \tag{2.19}$$

Theorem 2.8. *The KL-expansion is unique, i.e., the random variables in the KL–expansion (2.19) satisfy orthonormality (2.18) if and only if λ_n, κ_n are an eigenpair of the covariance kernel such that they solve (2.10).*

Proof. The "if" direction was just discussed. The "only if" direction can be seen by using orthonormality (2.18) in equation (2.15), multiplication with $\kappa_m(y)$ followed by integration over the domain along y such that

$$\begin{aligned}\int_D \mathrm{Cov}_\theta(x,y)\kappa_m(y)\,\mathrm{d}y &= \sum_{j=0}^{\infty}\lambda_j\kappa_j(x)\delta_{jm} \\ &= \lambda_m\kappa_m(x).\end{aligned}$$

□

An explicit expression of the random variables is given by

$$\xi_j(\omega) = \frac{1}{\sqrt{\lambda_j}}\int_D \theta(x,\omega)\kappa_j(x)\,\mathrm{d}x,$$

which can be seen by the usual routine of multiplying (2.14) with κ_n and integrating along x.

Note that it holds

$$\int_D \mathrm{Var}_\kappa(x)\,\mathrm{d}x = \int_D \int_\Omega \theta^2(x,y)\,\mathrm{d}\omega\,\mathrm{d}x = \int_D \mathrm{Cov}_\theta(x,\omega)\,\mathrm{d}x \tag{2.20}$$

$$= \sum_{j=1}^{\infty}\sum_{k=1}^{\infty} \sqrt{\lambda_j\lambda_k}\langle \xi_j(\omega)\xi_k(\omega)\rangle \int_D \kappa_j(x)\kappa_k(x)\,\mathrm{d}x \tag{2.21}$$

$$= \sum_{j=1}^{\infty} \lambda_j, \tag{2.22}$$

where we make use of the eigenvalue expansion (2.15) and (2.18), the orthogonality of the eigenvalues (2.12) and Lemma 2.6. We may assume a decreasing ordering of the eigenvalues, and since they converge to zero, a truncated KL-expansion

$$\kappa(x,\omega) = \bar{\kappa}(x) + \sum_{j=1}^{M} \sqrt{\lambda_j}\kappa_j(x)\xi_j(\omega) \tag{2.23}$$

yields more and more of the total variance of κ with increasing M [GS91, Chapter 2.3].

Remark 2.9 (On the covariance functions)**.** *Denote by $r = \|\boldsymbol{x}-\boldsymbol{y}\|_2$ the Euclidian distance and consider a homogeneous and isotropic random field such that $Cov_\kappa(\boldsymbol{x},\boldsymbol{y}) = \zeta(r)$. We introduce the correlation length, a length scale for the distance over which the random fields exhibits significant correlations. It is defined to be $\frac{1}{\zeta(0)}\int_0^\infty \zeta(r)\,\mathrm{d}r$, and the correlation between $\boldsymbol{x}$ and $\boldsymbol{y}$ as $\frac{Cov_\kappa(\boldsymbol{x},\boldsymbol{y})}{\sigma_\kappa(\boldsymbol{x})\sigma_\kappa(\boldsymbol{y})}$ which for a homogeneous and isotropic random field is equivalent to $\frac{\zeta(r)}{\zeta(0)}$. Some common choices of (geostatistical) covariance functions are listed below:*

$$Cov_\kappa(\boldsymbol{x},\boldsymbol{y}) = \sigma_\kappa^2 \cdot \exp\left(-\frac{|x_1-y_1|}{c_1} - \frac{|x_2-y_2|}{c_2}\right),$$

$$Cov_\kappa(\boldsymbol{x},\boldsymbol{y}) = \sigma_\kappa^2 \cdot \exp\left(-\frac{r}{c}\right),$$

$$Cov_\kappa(\boldsymbol{x},\boldsymbol{y}) = \sigma_\kappa^2 \cdot \exp\left(-\frac{r^2}{c^2}\right),$$

$$Cov_\kappa(\boldsymbol{x},\boldsymbol{y}) = \sigma_\kappa^2 \cdot \left(-\frac{r}{c}\right) \cdot Y_1\left(\frac{r}{c}\right),$$

for correlation lengths c, c_1, c_2, and Y_1 the modified Bessel function of second kind with order one.

A widley used modeling assumption is that the random field is of second order, that is, for any fixed point $\boldsymbol{x} \in D$, the random field $\kappa(\boldsymbol{x},\cdot)$ is a random variable with finite mean and variance. In that case, mean and covariance are well defined; yet such a random field is not uniquely defined by these two properties unless it is a Gaussian random field [Ull08, Chapter 2.2], [EEU07].

2.2. Stochastic finite element method

2.2.1. Discretization

We are going to discretize the spatial variables independently from the discretization of the random variables. To this end, let

$$W(D)^h = \operatorname{span}\{\phi_1, \dots, \phi_{N_x}\} \subset W(D) \tag{2.24}$$

be any suitable finite-dimensional subspace with N_x basis elements ϕ_i.

The discretization of the random variables is more involving. As first step we are going to discretize Ω by truncating the KL-expansion after M terms (compare (2.23)) such that the stochastic dependence of the approximated random field is now only on the random variables $\xi_1, \dots, \xi_M$:

$$\kappa(\boldsymbol{x}, \omega) = \kappa(\boldsymbol{x}, \boldsymbol{\xi}(\omega)) := \kappa(\boldsymbol{x}, \xi_1(\omega), \dots, \xi_M(\omega)), \tag{2.25}$$

$$f(\boldsymbol{x}, \omega) = f(\boldsymbol{x}, \boldsymbol{\xi}(\omega)). \tag{2.26}$$

Let $\Gamma_m := \xi_m(\Omega)$ denote the range of ξ_m and assume for all ξ_m probability denisty $\rho_m : \Gamma_m \to [0, \infty)$. By independence of the random variables, their joint probability density is given by

$$\rho(\boldsymbol{\xi}) = \rho_1(\xi_1), \dots, \rho_M(\xi_M), \quad \boldsymbol{\xi} \in \Gamma := \Gamma_1 \times \cdots \times \Gamma_M.$$

We now replace the space of random variables with finite variance $L^2_\mu(\Omega)$ by $L^2_\rho(\Gamma)$ such that we seek $u \in H^1_0(D) \otimes L^2_\rho(\Gamma)$ solving

$$\forall v \in H^1_0(D) \otimes L^2_\rho(\Gamma)\colon \quad \langle a(u, v) \rangle = \langle \ell(v) \rangle \tag{2.27}$$

for

$$\langle a(u, v) \rangle = \int_\Gamma \rho(\boldsymbol{\xi}) \int_D \kappa(\boldsymbol{x}, \boldsymbol{\xi}) \nabla u(\boldsymbol{x}, \boldsymbol{\xi}) \cdot \nabla v(\boldsymbol{x}, \boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{\xi}, \tag{2.28}$$

$$\langle \ell(v) \rangle = \int_\Gamma \rho(\boldsymbol{\xi}) \int_D f(\boldsymbol{x}, \boldsymbol{\xi}) v(\boldsymbol{x}, \boldsymbol{\xi}) \, \mathrm{d}\boldsymbol{x} \, \mathrm{d}\boldsymbol{\xi}. \tag{2.29}$$

In the next step we construct a finite-dimensional subspace

$$W(\Gamma)^h = \operatorname{span}\left\{\psi_1(\boldsymbol{\xi}), \dots, \psi_{N_\xi}(\boldsymbol{\xi})\right\} \subset L^2_\rho(\Gamma),$$

with basis functions $\psi_i, i = 1, \dots, N_\xi$, so we can finally seek the solution u^h in

$$W(D)^h \otimes W(\Gamma)^h = \left\{ v \in L^2(D \times \Gamma) \mid v \in \operatorname{span}\{\phi(\boldsymbol{x})\psi(\boldsymbol{x}) : \phi \in W(D)^h, \psi \in W(\Gamma)^h\} \right\}. \tag{2.30}$$

There are several choices for the basis functions ψ_i, a popular choice is inspired by the tensor product structure of

$$W(\Gamma)^h = \bigotimes_{i=1}^{M} L^2_{\rho_i}(\Gamma_i)$$

and uses Hermite polynomials, i.e., multivariant polynomials consisting of univariant Hermite polynomials

$$W(\Gamma)^h = \operatorname{span}\left\{ \psi_\alpha(\boldsymbol{\xi}) = \prod_{m=1}^{M} \psi_{\alpha_m}(\xi_m) \colon \psi_{\alpha_m} \in W(\Gamma)_m^h \subset L_{\rho_m}(\Gamma_m) \right\} \tag{2.31}$$

for multiindices $\alpha \in \mathbb{N}_0^M$ [EEU07].

2.2.2. Structure of Galerkin equations

From (2.30) it follows that the solutions $u^h \in W(D)^h \otimes W(\Gamma)^h$ can be represented by a sum

$$u^h(\boldsymbol{x}, \boldsymbol{\xi}) = \sum_{i=1}^{N_x} \sum_{j=1}^{N_\xi} u_{i,j} \phi_i(\boldsymbol{x}) \psi_j(\boldsymbol{\xi})$$

with $N_x \cdot N_\xi$ coefficients $u_{i,j}$. By using this sum and test functions $v(\boldsymbol{x}, \boldsymbol{\xi}) = \phi_k(\boldsymbol{x})\psi_l(\boldsymbol{\xi})$ for $k = 1, \ldots, N_x, l = 1, \ldots, N_\xi$ in (2.28),(2.29) in the weak formulation (2.27) we derive a coupled system of equations

$$\begin{aligned} \forall k, l : \quad \sum_{i=1}^{N_x} \sum_{j=1}^{N_\xi} \left(\int_\Gamma \rho(\boldsymbol{\xi}) \psi_j(\boldsymbol{\xi}) \psi_l(\boldsymbol{\xi}) \left[\boldsymbol{K}(\boldsymbol{\xi})\right]_{i,k} \, \mathrm{d}\boldsymbol{\xi} \right) u_{i,j} & \\ = \int_\Gamma \rho(\boldsymbol{\xi}) \psi_l(\boldsymbol{\xi}) \left[\boldsymbol{f}(\boldsymbol{\xi})\right]_k \, \mathrm{d}\xi & \end{aligned}$$

where we have defined the matrices

$$[\boldsymbol{K}(\boldsymbol{\xi})]_{i,k} := \int_D \kappa(\boldsymbol{x}, \omega) \nabla\phi_i(\boldsymbol{x}) \cdot \nabla\phi_k(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \qquad \boldsymbol{K}(\boldsymbol{\xi}) \in \mathbb{R}^{N_x \times N_x}, \tag{2.32}$$

$$[\boldsymbol{f}(\boldsymbol{\xi})]_k := \int_D f(\boldsymbol{x}, \omega) \phi_k(\boldsymbol{x}) \, \mathrm{d}\boldsymbol{x}, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^{N_x}. \tag{2.33}$$

The Galerkin formulation is of coupled structure and takes the form

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{f} \tag{2.34}$$

for block matrix and block vector

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{1,1} & \cdots & \boldsymbol{A}_{1,N_x} \\ \vdots & & \vdots \\ \boldsymbol{A}_{N_\xi,1} & \cdots & \boldsymbol{A}_{N_\xi,N_x} \end{pmatrix}, \quad \boldsymbol{f} = \begin{pmatrix} \boldsymbol{f}_1 \\ \vdots \\ \boldsymbol{f}_{N_\xi} \end{pmatrix} \tag{2.35}$$

consisting of matrices respectively vectors

$$\boldsymbol{A}_{l,j} := \langle \psi_j(\boldsymbol{\xi}) \psi_l(\boldsymbol{\xi}) \boldsymbol{K}(\boldsymbol{\xi}) \rangle, \quad \boldsymbol{A}_{l,j} \in \mathbb{R}^{N_x \times N_x}, \tag{2.36}$$

$$\boldsymbol{f}_l := \langle \psi_l(\boldsymbol{\xi}) \boldsymbol{f}(\boldsymbol{\xi}) \rangle, \quad \boldsymbol{f}_l \in \mathbb{R}^{N_x}, \tag{2.37}$$

such that the system matrix $\boldsymbol{A}$ is of size $(N_\xi \cdot N_x) \times (N_\xi \cdot N_x)$ and system right-hand side $\boldsymbol{f}$ is of size $(N_\xi \cdot N_x) \times 1$ [EEU07].

2.2.3. Structure of the random field

If the random field is approximated by a truncated KL-expansion, the expansion consists of terms linear in the random variables $\{\xi_m(\boldsymbol{x})\}_{m=1}^M$. Moreover, if the random field is a Gaussian process, the random variables $\xi_m(\boldsymbol{x})$ are Gaussian as well, and since they are uncorrelated they are independent as well [GS91, Chapter 2.3].

For random fields

$$\kappa(\boldsymbol{x},\boldsymbol{\xi}) = \kappa_0(\boldsymbol{x}) + \sum_{m=1}^{M} \kappa_m(\boldsymbol{x})\xi_m, \tag{2.38}$$

$$f(\boldsymbol{x},\boldsymbol{\xi}) = f_0(\boldsymbol{x}) + \sum_{m=1}^{M} f_m(\boldsymbol{x})\xi_m \tag{2.39}$$

the associated matrices are of the form

$$\boldsymbol{K}(\boldsymbol{\xi}) = \boldsymbol{K}_0 + \sum_{m=1}^{M} \boldsymbol{K}_m\xi_m, \tag{2.40}$$

$$\boldsymbol{f}(\boldsymbol{\xi}) = \boldsymbol{f}_0 + \sum_{m=1}^{M} \boldsymbol{f}_m\xi_m \tag{2.41}$$

for $m = 0,\ldots,M$, where we defined the matrices

$$[\boldsymbol{K}_m]_{i,k} := \langle \kappa_m \nabla\phi_k, \nabla\phi_i\rangle_{L^2(D)}\,, \tag{2.42}$$

$$[\boldsymbol{f}_m]_k := \langle f_m, \phi_k\rangle_{L^2(D)} \tag{2.43}$$

with $i,k = 1,\ldots,N_x$, where the matrix respectively vector $\boldsymbol{K}_0, \boldsymbol{f}_0$ correspond to the mean of the random fields. Thus, the system matrix $\boldsymbol{A}$ and right-hand side $\boldsymbol{f}$ in (2.34) possess a tensor product structure

$$\boldsymbol{A} = \boldsymbol{G}_0 \otimes \boldsymbol{K}_0 + \sum_{m=1}^{M} \boldsymbol{G}_m \otimes \boldsymbol{K}_m, \tag{2.44}$$

$$\boldsymbol{f} = \boldsymbol{g}_0 \otimes \boldsymbol{f}_0 + \sum_{m=1}^{M} \boldsymbol{g}_m \otimes \boldsymbol{f}_m \tag{2.45}$$

with matrices $\boldsymbol{G}_m$ and vectors $\boldsymbol{g}_m$ given in terms of the stochastic basis $W(\Gamma)^h$ and random variables $\{\xi_m\}_{m=1}^M$ as

$$[\boldsymbol{G}_0]_{l,k} = \langle \psi_k\psi_l\rangle, \qquad [\boldsymbol{G}_m]_{l,k} = \langle \xi_m\psi_k\psi_l\rangle, \tag{2.46}$$

$$[\boldsymbol{g}_0]_l = \langle \psi_l\rangle, \qquad [\boldsymbol{g}_m]_l = \langle \xi_m\psi_l\rangle \tag{2.47}$$

for $m = 1,\ldots,M$ and $k,l = 1,\ldots,N_x$.

The matrices $\boldsymbol{G}_m$ in (2.46) can be brought into diagonal form, such that $\boldsymbol{A}$ becomes a block diagonal matrix. To this end, one has to choose global orthogonal polynomials as basis functions of $W(\Gamma)^h$, see [EEU07] and references therein.

2.3. Computational aspects

2.3.1. The eigenvalue problem

The KL-expansion requires solving the covariance eigenproblem (2.10). As in Remark 2.9 we consider covariance kernels $\zeta(r)$ depending on the Euclidian distance on a homogeneous and isotropic random field. Although we could use the spatial discretization $W(D)^h$ in (2.24), the eigenvalue problem typically has other discretization requirements. Therefore we introduce another spatial discretization

$$Y^h = \operatorname{span}\{\eta_1, \dots, \eta_N\} \subset L^2(D).$$

The Galerkin condition

$$\forall v \in Y^h : \quad \langle Cu, v \rangle_{L^2(D)} = \lambda \langle u, v \rangle_{L^2(D)}$$

is equivalent to the eigenvalue problem in matrix form

$$\boldsymbol{C}\boldsymbol{u} = \lambda \boldsymbol{M}\boldsymbol{u}$$

with symmetric positive semidefinite matrix $\boldsymbol{C}$ and symmetric positive definite matrix $\boldsymbol{M}$ given by

$$\boldsymbol{C}_{i,j} := \langle C\eta_j, \eta_i \rangle_{L^2(D)}, \quad \boldsymbol{C} \in \mathbb{R}^{N \times N}, \tag{2.48}$$

$$\boldsymbol{M}_{i,j} := \langle \eta_j, \eta_i \rangle_{L^2(D)}, \quad \boldsymbol{M} \in \mathbb{R}^{N \times N}. \tag{2.49}$$

The number of terms M in the truncated KL-expansion (2.23) is usually smaller than N. In this setting, Krylov subspace methods may be used to solve the eigenvalue problem. In [EEU07], the authors propose the usage of Lanczos thick-restart method, another popular choice is an implicitly restarted Arnoldi method, see [EEU07] and references therein.

2.3.2. Solving the Galerkin system

Usually, the spatial discretization number N_x is rather large, such that even for moderate choices of N_ξ the block matrix A becomes huge as it is of size $(N_x \cdot N_\xi) \times (N_x \cdot N_\xi)$. Solving such system is an expensive task. One may use double orthogonal polynomials for the discretization of $W(\Gamma)^h$ to derive a system matrix $\boldsymbol{A}$ which is of block diagonal form, such that there are N_ξ uncoupled problems of blocksize $N_x \times N_x$ to solve. Such smaller problems can be solved in parallel.

There are settings where this task becomes simpler. For a *stochastic right-hand side problem*, that is only the right-hand side data is random, and left-hand side is deterministic (as opposed to the model problem (2.6), (2.7)), one may choose a orthonormal basis of $W(\Gamma)^h$. Now the system matrix $\boldsymbol{A}$ becomes a block diagonal matrix with constant blocks $\boldsymbol{A}_{l,j}$. This allows the usage of block Krylov solvers. A *stochastic left-hand side problem* deals with the opposite, so there is a random field κ on the left while the right-hand side $\boldsymbol{f}$ is deterministic, see [EEU07] and references therein.

3. Non-intrusive Galerkin method

Following [GLL⁺14], we introduce a *non-intrusive* method for solving a parameter-dependent stochastical problem. That means the method makes use of deterministic solver software for solving the stochastic problem. It is apparent that already available deterministic software makes this method attractive. The method computes coefficients to a parametric equation such that once those coefficients are computed, solutions to new sets of parameters can be computed quickly.

3.1. Parameter-dependent deterministic problems

Here we review a classical iterative scheme to solve a parameter-dependent deterministic problem, that is for a fixed set of parameters.

A discretized parameter-dependent problem may be written in the form of

$$A(p;u) = f(p) \tag{3.1}$$

with maps $A : \mathcal{P} \times \mathcal{U} \to \mathcal{U}$ and $f : \mathcal{P} \to \mathcal{U}$ for $u \in \mathcal{U}$ element of a finite-dimensional Hilbert space with $\dim \mathcal{U} := N < \infty$ and $p \in \mathcal{P}$ element of the parameter space.

Assume now that for all $p \in \mathcal{P}$, equation (3.1) is a well-posed problem. That means that for fixed p, the mapping $u \mapsto A(p;u)$ is bijective and continuously invertible. In that case, for all $p \in \mathcal{P}, f(p) \in \mathcal{U}$ there exists a unique solution $u^*(p)$ satisfying $A(p, u^*(p)) = f(p)$ for all p.

An iterative solver converging for all p generates successive iterates

$$u^{(k+1)}(p) = \mathscr{C}\left(p; u^{(k)}, R\left(p; u^{(k)}(p)\right)\right) \tag{3.2}$$

for $k = 0, \ldots$ with $u^{(k)}(p) \xrightarrow{k\to\infty} u^*(p)$. It denotes $\mathscr{C}$ one cycle of the solver, taking inputs p, the current iterate $u^{(k)}$ and the current residual

$$R\left(p; u^{(k)}(p)\right) = f(p) - A\left(p; u^{(k)}\right) \tag{3.3}$$

which we will conveniently refer to as $R^{(k)}$. A fixed-point solution $u^*(p)$ is found when the residual vanishes.

One may rewrite (3.2) as

$$u^{(k+1)} = u^{(k)} + \Delta u^{(k)} \tag{3.4}$$

with

$$\Delta u^{(k)} := \mathscr{C}\left(p; u^{(k)}, R^{(k)}\right) - u^{(k)}, \tag{3.5}$$

$$P(\Delta u^{(k)}) = R^{(k)} \tag{3.6}$$

Algorithm 3.1 Iteration for (3.2)

```
Use some initial guess $u^{(0)}$
$k \leftarrow 0$
**while** *no convergence* **do**
    Compute $\Delta u^{(k)}$ using (3.5) or (3.6)
    $u^{(k+1)} \leftarrow u^{(k)} + \Delta u^{(k)}$
    $k \leftarrow k + 1$
**end while**
```

for a preconditioner P may depending on the parameter p, the iteration counter k and the current iterate $u^{(k)}$ such that

$$\mathscr{C}\Big(p; u^{(k)}, R^{(k)}\Big) = u^{(k)} + P^{-1} R^{(k)}.$$

In either case, we assume P to be linear in its arguments and non-singular in $\Delta u^{(k)}$.

For the following convergence analysis we assume that the iteration converges at least linearly such that

$$\left\|\Delta u^{(k+1)}(p)\right\|_{\mathcal{U}} \leq \varrho(p) \left\|\Delta u^{(k)}(p)\right\|_{\mathcal{U}}$$

with Lipschitz constant $\varrho(p) < 1$ which we assume to be bounded uniformly for all $p \in \mathcal{P}$, so there is a constant ϱ^* with $\varrho(p) \leq \varrho^* < 1$, such that the solver $\mathscr{C}$ is a uniformly Lipschitz continuous contraction

$$\left\|\mathscr{C}\Big(p; u(p), R(p; u(p))\Big) - \mathscr{C}\Big(p; v(p), R(p; v(p))\Big)\right\|_{\mathcal{U}} \leq \varrho^* \left\|u(p) - v(p)\right\|_{\mathcal{U}} \tag{3.7}$$

for all $u, v \in \mathcal{U}$.

The deterministic iteration is given in Algorithm 3.1, where an initial guess $u^{(0)}$ and a convergence criterion has to be supplied by the user.

For later use we state

Theorem 3.1 (Banach fixed-point Theorem [Zei95, Thm. 1.A])**.** *For a Banach space $\mathcal{U}$ and a contraction mapping $\mathscr{C} : \mathcal{U} \to \mathcal{U}$ with Lipschitz factor $\varrho^* < 1$ such that (3.7) holds, there exists a unique fixed-point u^* and the iteration converges to it. Moreover the a posteriori error estimate*

$$\left\|u^*(p) - u^{(k+1)}(p)\right\|_{\mathcal{U}} \leq \frac{\varrho^*}{1 - \varrho^*} \left\|\Delta u^{(k)}(p)\right\|_{\mathcal{U}} \tag{3.8}$$

holds true.

3.2. Galerkin approximation of parameter dependence

We want to approximate the parameter dependent solution $u^*(p)$ in the form

$$u^*(p) \approx u_{\mathcal{I}}(p) := \sum_{\alpha \in \mathcal{I}} u_\alpha \psi_\alpha(p) \tag{3.9}$$

with row vectors $u_\alpha \in \mathcal{U}$ to be determined, and $\{\psi_\alpha\}_{\alpha\in\mathcal{I}} : \mathcal{P} \to \mathbb{R}$ basis functions spanning the space of ansatz functions $\mathcal{Q}_\mathcal{I}$, which is a finite-dimensional subspace of a Hilbert space $\mathcal{Q}$.

Note that representation (3.9) allows quick computation of solutions for any set of parameters p, once the coefficients u_α are determined. This property is termed *rapid evaluation.*

Definition 3.2 ([GWZ14, Chapter 4.2]). *We choose an ansatz degree n and define the total degree (TD) index set*

$$\mathcal{I}_{\mathrm{TD}} := \left\{ \alpha \in \mathbb{N}_0^d : \sum_{j=1}^{d} \alpha_j \le n \right\},$$

the sparse Smolyak (SS) index set

$$\mathcal{I}_{\mathrm{SS}} := \left\{ \alpha \in \mathbb{N}_0^d : \sum_{j=1}^{d} \gamma(\alpha_j) \le \gamma(n) \right\}$$

with

$$\gamma(n) := \begin{cases} 0 & n = 0, \\ 1 & n = 1, \\ \lceil \log_2(n) \rceil & n \ge 2, \end{cases}$$

and the hyperbolic cross (HC) index set

$$\mathcal{I}_{\mathrm{HC}} := \left\{ \alpha \in \mathbb{N}_0^d : \sum_{j=1}^{d} \log_2(\alpha_j + 1) \le \log_2(n+1) \right\}.$$

Compare Figure 3.1 for their cardinality. We assume that an arbitrary but fixed ordering of I is chosen, e.g., lexicographically.

3.2.1. Galerkin equations for the residual

Using a Galerkin method to solve (3.1), we must reformulate the problem as weak formulation, i.e., multiplying the problem with linearly independent parametric test functions $\{\varphi_\beta\}_{\beta\in\mathcal{I}} : \mathcal{P} \to \mathbb{R}$ spanning the space of test functions $\hat{\mathcal{Q}}_\mathcal{I}$, followed by integration over the domain and usually applying Gauss' or Greens theorem (compare Appendix A). We denote this integration step as Galerkin projection $\boldsymbol{G}_\mathcal{Q}$. The coefficients u_α of the approximation $u_\mathcal{I}(\cdot)$ are determined by requiring that

$$\forall \beta \in \mathcal{I} : \quad \boldsymbol{G}_\mathcal{Q}\left(\varphi_\beta(\cdot) R(\cdot, u_\mathcal{I})\right) = \boldsymbol{G}_\mathcal{Q}\left(\varphi_\beta(\cdot)\left(f(\cdot) - A(\cdot, u_\mathcal{I})\right)\right) = 0. \tag{3.10}$$

The choice of the ansatz space $\mathcal{Q}_I$ determines how well the solution $u^*(p)$ is approximated as a parametric function $u_\mathcal{I}(p)$. The choice of the test space $\hat{\mathcal{Q}}_I$ is responsible for the

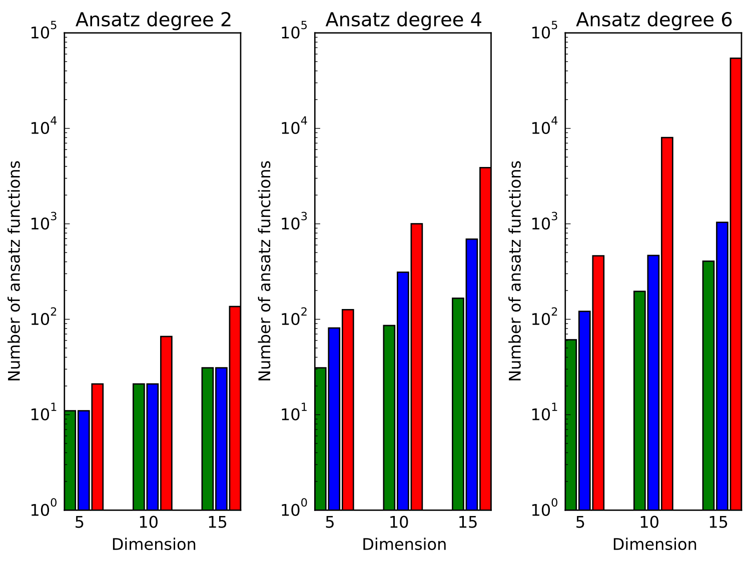


Figure 3.1.: Ansatz index sets cardinality for total degree (red), sparse Smolyak (blue) and hyperbolic cross (green)

stability of the method, as it determines the projection onto the ansatz space in such a way that it is orthogonal to $\hat{\mathcal{Q}}_I$:

$$\forall \varphi \in \hat{\mathcal{Q}}_{\mathcal{I}} : \quad \boldsymbol{G}_{\mathcal{Q}} \left(\varphi(\cdot)(u^*(\cdot) - u_{\mathcal{I}}(\cdot)) \right) = 0. \tag{3.11}$$

For sake of simplicity, we assume a Galerkin method such that ansatz and test spaces coincide $\mathcal{Q}_{\mathcal{I}} = \hat{\mathcal{Q}}_I, \forall \alpha \in \mathcal{I} : \varphi_\alpha = \psi_\alpha$, in contrast to Petrov-Galerkin methods where those spaces differ $\mathcal{Q}_I \neq \hat{\mathcal{Q}}_I$.

We introduce

$$\forall w \in \mathcal{U} : \quad \left\langle \boldsymbol{G}_{\mathcal{Q}} \left(\varphi(\cdot)\psi(\cdot)v \right), w \right\rangle_{\mathcal{U}} = \langle \varphi, \psi \rangle_{\mathcal{Q}} \langle v, w \rangle_{\mathcal{U}} \tag{3.12}$$

for $v \in \mathcal{U}$ and a bilinear duality pairings $\langle \cdot, \cdot \rangle_{\mathcal{Q}}, \langle \cdot, \cdot \rangle_{\mathcal{U}}$. As the residual $R(p; u_{\mathcal{I}}(p))$ is a parametric function for $p \in \mathcal{P}$, it may be represented as a sum

$$R(p; u_{\mathcal{I}}(p)) = \sum_{\alpha \in \mathcal{I}} \psi_\alpha(p) v_\alpha \tag{3.13}$$

with orthonormal basis functions v_α spanning $\mathcal{U}$.

Using the linearity of the Galerkin projection and (3.13) we can rewrite (3.10) as

$$\begin{aligned} \forall \beta \in \mathcal{I} : \quad 0 = \boldsymbol{G}_{\mathcal{Q}} \left(\varphi_\beta(\cdot) R(\cdot; u_{\mathcal{I}}) \right) &= \boldsymbol{G}_{\mathcal{Q}} \left(\varphi_\beta(\cdot) \sum_{\alpha \in \mathcal{I}} \psi_\alpha(\cdot) v_\alpha \right) \\ &= \sum_{\alpha \in \mathcal{I}} \boldsymbol{G}_{\mathcal{Q}} \left(\varphi_\beta(\cdot) \psi_\alpha(\cdot) v_\alpha \right) = \sum_{\alpha \in \mathcal{I}} \langle \varphi_\beta, \psi_\alpha \rangle_{\mathcal{Q}} v_\alpha. \end{aligned}$$

If $\mathcal{P}$ is a measure space with measure μ, the choice of bilinearity pairing is often

$$\langle \varphi, \psi \rangle_{\mathcal{Q}} = \int_{\mathcal{P}} \varphi(p)\psi(p)\,\mu(\mathrm{d}p), \tag{3.14}$$

if furthermore $\mu(\mathcal{P}) = 1$ holds we can consider $\mathcal{P}$ as probability space with induced L^2 norm and expectation operator

$$\mathbb{E}(\varphi) = \int_{\mathcal{P}} \varphi(p)\,\mu(\mathrm{d}p)$$

such that $\mathbb{E}(\varphi\psi) = \langle \varphi, \psi \rangle_{\mathcal{Q}}$ holds.

Inserting the solution (3.9) into the projected residual (3.10) yields

$$\forall \beta \in \mathcal{I}: \quad \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)\left(f(\cdot) - A\left(\cdot; \sum_{\alpha\in\mathcal{I}} u_\alpha \psi_\alpha(\cdot)\right)\right)\right) = 0 \tag{3.15}$$

which is a *coupled system* of equations with M unknowns $u_\alpha \in \mathcal{U} = \mathbb{R}^N$. Due to the different structure of (3.15) and (3.1) one may conclude that a solver of type (3.2) cannot be used here. In the following, [GLL+14] show that this is not true in general, and the above problem may be solved non-intrusively.

3.2.2. Galerkin fixed-point equations

We will use equation (3.2) as starting point for the non-intrusive computation, rather than (3.1) or (3.3). First we show that this is feasible.

Lemma 3.3. *Projecting the fixed-point equation associated to* (3.2)*, namely*

$$u_{\mathcal{I}} = u_{\mathcal{I}} + P^{-1}R(\cdot; u_{\mathcal{I}}) \tag{3.16}$$

is equivalent to projecting the preconditioned residual such that

$$\forall \beta \in \mathcal{I}: \quad \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)P^{-1}R(\cdot; u_{\mathcal{I}})\right) = 0. \tag{3.17}$$

Moreover, if the preconditioner P from equation (3.6) *does not depend on p nor u, we derive*

$$\forall \beta \in \mathcal{I}: \quad \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)R(\cdot; u_{\mathcal{I}})\right) = 0. \tag{3.18}$$

Proof. We start with the equivalence of (3.16) and (3.17). From the linearity of the Galerkin projection it follows

$$\begin{aligned}\forall \beta \in \mathcal{I}: \quad & \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)(u_{\mathcal{I}} + P^{-1}R(\cdot; u_{\mathcal{I}}))\right) \\ &= \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)u_{\mathcal{I}}\right) + \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)P^{-1}R(\cdot; u_{\mathcal{I}})\right) = 0.\end{aligned} \tag{3.19}$$

Now if (3.16) holds it follows that $P^{-1}R(\cdot; u_{\mathcal{I}}) = 0$, and with (3.19) it is immediate that (3.17) holds as a fixed point is found. On the other hand, if (3.17) holds it follows $P^{-1}R(\cdot; u_{\mathcal{I}}) = 0$ since $P^{-1}R(\cdot; u_{\mathcal{I}}) \in \mathcal{U}$, and thus (3.16) holds.

For the second claim note that for a linear map L on $\mathcal{U}$ we have

$$\begin{aligned}\boldsymbol{G}_{\mathcal{Q}}\left(\varphi(\cdot)L(\psi(\cdot)v)\right) &= \int_{\mathcal{P}} \varphi(p)L\psi(p)v\,\mu(\mathrm{d}p) = \langle\varphi, L\psi\rangle_{\mathcal{Q}}\, v = L\,\langle\varphi,\psi\rangle_{\mathcal{Q}}\, v \\ &= L\boldsymbol{G}_{\mathcal{Q}}\left(\varphi(\cdot)\psi(\cdot)v\right).\end{aligned}$$

In case the preconditioner does not depend on p or u, it follows immediately that

$$\forall\beta\in\mathcal{I}: 0 = \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)P^{-1}R^{(k)}\right) = P^{-1}\boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)R^{(k)}\right)$$

implies

$$\forall\beta\in\mathcal{I}: \quad \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)R^{(k)}\right) = 0$$

because P was assumed to be non-singular. □

Expanding the k-th iterate as

$$u^{(k)} = \sum_{\alpha\in\mathcal{I}} u_\alpha^{(k)}\psi_\alpha(p)$$

and inserting it into the Galerkin projection in (3.19) yields

$$\begin{aligned}\forall\beta: \quad \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)\sum_{\alpha\in\mathcal{I}} u_\alpha^{(k+1)}\psi_\alpha(\cdot)\right) &= \\ \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)\sum_{\alpha\in\mathcal{I}} u_\alpha^{(k)}\psi_\alpha(\cdot)\right) &+ \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)R^{-1}R^{(k)}\right).\end{aligned}$$

This can be written in matrix form

$$\forall\beta: \sum_{\alpha\in\mathcal{I}} \boldsymbol{M}_{\beta,\alpha}u_\alpha^{(k+1)} = \sum_{\alpha\in\mathcal{I}} \boldsymbol{M}_{\beta,\alpha}u_\alpha^{(k)} + \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\beta(\cdot)R^{-1}R^{(k)}\right) \tag{3.20}$$

where we use (3.12) for the entries of $\boldsymbol{M}$, and arrange the coefficients and the residual projection in a column-wise form

$$\boldsymbol{M}_{\beta,\alpha} = \langle\varphi_\beta, \psi_\alpha\rangle_{\mathcal{Q}}, \quad \boldsymbol{M}\in\mathbb{R}^{M\times M}, \tag{3.21}$$

$$\boldsymbol{u}_\alpha^{(k)} = \left[\ldots, u_\alpha^{(k)}, \ldots\right] \in \mathbb{R}^{N\times M}, \tag{3.22}$$

$$\boldsymbol{G}_{\mathcal{Q}}\left(P^{-1}R^{(k)}\right) = \left[\ldots, \boldsymbol{G}_{\mathcal{Q}}\left(\varphi_\alpha(\cdot)P^{-1}R^{(k)}\right), \ldots\right] \in \mathbb{R}^{N\times M}. \tag{3.23}$$

Equation (3.20) may be written in the form

$$\boldsymbol{u}^{(k+1)}\boldsymbol{M}^T = \boldsymbol{u}^{(k)}\boldsymbol{M}^T + \boldsymbol{G}_{\mathcal{Q}}\left(P^{-1}R^{(k)}\right),$$

or equivalently

$$\boldsymbol{u}^{(k+1)} = \boldsymbol{u}^{(k)} + \boldsymbol{G}_{\mathcal{Q}}\left(P^{-1}R^{(k)}\right)\boldsymbol{M}^{-T}. \tag{3.24}$$

Algorithm 3.2 block Jacobi iteration of (3.24)

```
Use some initial guess $\boldsymbol{u}^{(0)} \in \mathbb{R}^{N \times M}$
$k \leftarrow 0$
while no convergence do
    Compute $\boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)})$ using (3.25) in the form of (3.23)
    $\boldsymbol{u}^{(k+1)} \leftarrow \boldsymbol{u}^{(k)} + \boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)})$
    $k \leftarrow k + 1$
end while
```

For biorthogonal bases of ansatz and test functions (3.21) the situation simplifies to $\forall \alpha, \beta \in \mathcal{I} : M_{\beta,\alpha} = \delta_{\beta\alpha} M_{\alpha,\alpha}$, hence $\boldsymbol{M} = \operatorname{diag}(M_{\alpha,\alpha})$. In case the basis functions are normalized (which we assume from now on) it holds $M_{\alpha,\alpha} = 1$, and thus the matrix $\boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)})$ has the form of (3.23).

The block Jacobi fixed-point iteration associated to (3.24) is given in Algorithm 3.2, where we use the definition

$$\boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)}) := \boldsymbol{G}_{\mathcal{Q}} \left(P^{-1} R^{(k)} \right) \boldsymbol{M}^{-T}. \tag{3.25}$$

An approximation of the residual $\boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)})$ will be computed non-intrusively in Chapter 3.3. Beforewe analyze the convergence of such a coupled iteration.

3.2.3. Convergence analysis

Recall from Chapter 3.1 that we assumed the iteration cycle $\mathscr{C}$ to be a contraction with Lipschitz constant $\varrho^* < 1$. We now want to show that the map

$$\begin{aligned} \boldsymbol{S}_{\mathcal{Q}} \colon \mathcal{U}^{\mathcal{I}} &\longrightarrow \mathcal{U}^{\mathcal{I}} \\ \boldsymbol{u}^{(k)} &\longmapsto \boldsymbol{u}^{(k)} + \boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)}). \end{aligned} \tag{3.26}$$

is also a contraction with the same Lipschitz constant.

Since the residual (3.13) is a parametric map $p \mapsto u(p)$, it can be written as sum $\sum_{\alpha \in \mathcal{I}} \psi_\alpha(p) w_\alpha$ with $w_\alpha \in \mathcal{U}$, which we identify with a sum in the Hilbert tensor product space

$$\sum_{\alpha \in \mathcal{I}} \psi_\alpha(p) w_\alpha \in \mathcal{Q} \otimes \mathcal{U}.$$

We define the inner product in this space as the product of inner products of the respective spaces $\langle \psi_1 \otimes v_1, \psi_2 \otimes v_2 \rangle_{\mathcal{Q} \otimes \mathcal{U}} := \langle \psi_1, \psi_2 \rangle_{\mathcal{Q}} \langle w_1, w_2 \rangle_{\mathcal{U}}$. Since we assumed $\mathcal{Q}$ to be a Hilbert space with closed subspace $\mathcal{Q}_{\mathcal{I}}$ we can apply the orthogonal decomposition theorem [Kan03, Thm. 1.36] to derive a decomposition into a direct sum $\mathcal{Q} = \mathcal{Q}_{\mathcal{I}} \oplus \mathcal{Q}_{\mathcal{I}}^{\perp}$ which leads to the decomposition $\mathcal{Q} \otimes \mathcal{U} = (\mathcal{Q}_{\mathcal{I}} \otimes \mathcal{U}) \oplus (\mathcal{Q}_{\mathcal{I}}^{\perp} \otimes \mathcal{U})$ of the tensor product space by using distributivity of the tensor product over the direct sum [Eps10, Chapter 3.1.1].

In the following we factorize $\boldsymbol{S}_{\mathcal{Q}}$ to see that its Lipschitz factor only depends on the solver cycle $\mathscr{C}$. To this end we define $\mathcal{U}^{\mathcal{I}} := \mathbb{R}^{N\times M}$ consisting of $\boldsymbol{u} = [\ldots, u_\alpha, \ldots] \in \mathcal{U}^{\mathcal{I}}$ with $\alpha \in \mathcal{I}$ and a map

$$\begin{aligned} J\colon \mathcal{U}^{\mathcal{I}} &\longrightarrow (\mathcal{Q}_{\mathcal{I}} \otimes \mathcal{U}) \subset (\mathcal{Q} \otimes \mathcal{U}) \\ \boldsymbol{u} &\longmapsto \sum_{\alpha\in\mathcal{I}} \psi_\alpha(\cdot) u_\alpha, \end{aligned} \tag{3.27}$$

which is bijective onto $\mathcal{Q}_{\mathcal{I}} \otimes \mathcal{U}$ since $\{\psi_\alpha\}_{\alpha\in\mathcal{I}}$ form a basis of $\mathcal{Q}_{\mathcal{I}}$. If we define

$$\|\boldsymbol{u}\|^2_{\mathcal{U}^{\mathcal{I}}} := \|J\boldsymbol{u}\|^2_{\mathcal{Q}\otimes\mathcal{U}} = \left\|\sum_{\alpha\in\mathcal{I}} \psi_\alpha(\cdot) u_\alpha\right\|^2_{\mathcal{Q}\otimes\mathcal{U}} = \sum_{\alpha\in\mathcal{I}} \|u_\alpha\|^2_{\mathcal{U}} \tag{3.28}$$

using unit measure of the basis functions in the last equality, we see that $\langle J\boldsymbol{u}, J\boldsymbol{u}\rangle_{\mathcal{Q}\otimes\mathcal{U}} = \sum_\alpha \langle u_\alpha, u_\alpha\rangle_{\mathcal{U}}$, so J is a unitary map and thereby $\|J\| = 1$. We can conclude that J induces norm and inner product on $\mathcal{U}^{\mathcal{I}}$ by (3.28). Now consider $J : \mathcal{U}^{\mathcal{I}} \to \mathcal{Q}\otimes\mathcal{U}$ as a mapping into the larger tensor product space extended by inclusion.

Lemma 3.4. *The maps $\boldsymbol{G}_{\mathcal{Q}}\colon \mathcal{Q}\otimes\mathcal{U} \to \mathcal{U}^{\mathcal{I}}$ and $J\colon \mathcal{U}^{\mathcal{I}} \to \mathcal{Q}\otimes\mathcal{U}$ are adjoint, with $\|\boldsymbol{G}_{\mathcal{Q}}\| = \|J\| = 1$.*

Proof. For $\phi \otimes w \in \mathcal{Q}\otimes\mathcal{U}$ and $\boldsymbol{v} \in \mathcal{U}^{\mathcal{I}}$ it holds

$$\begin{aligned} \langle \boldsymbol{G}_{\mathcal{Q}}(\phi\otimes w), \boldsymbol{v}\rangle_{\mathcal{U}^{\mathcal{I}}} &= \big\langle [\ldots, \langle\psi_\alpha, \phi\rangle_{\mathcal{Q}}\, w, \ldots], [\ldots, v^\alpha, \ldots]\big\rangle_{\mathcal{U}^{\mathcal{I}}} && (3.29)\\ &= \sum_{\alpha\in\mathcal{I}} \langle\psi_\alpha, \phi\rangle_{\mathcal{Q}} \langle w, v^\alpha\rangle_{\mathcal{U}} && (3.30)\\ &= \left\langle \phi\otimes w, \sum_{\alpha\in\mathcal{I}} \psi_\alpha \otimes v^\alpha \right\rangle_{\mathcal{Q}\otimes\mathcal{U}} && (3.31)\\ &= \langle \phi\otimes w, J\boldsymbol{v}\rangle_{\mathcal{Q}\otimes\mathcal{U}} && (3.32) \end{aligned}$$

where we use in (3.29) the form of the Galerkin projection (3.12), in (3.30) bilinearity of the inner product, in (3.31) the definition of the inner product (3.28) and in (3.32) the definition of J in (3.27). Hence $\boldsymbol{G}_{\mathcal{Q}}$ and J are indeed adjoint. As above discussed J is an isometry $\|J\| = 1$, and since adjoint operators in Hilbert spaces have equal norms the claim follows. □

By projecting equation (3.2) and using linearity of the Galerkin projection along with definitions (3.25),(3.26) and Lemma 3.4 we derive

$$\begin{aligned} \boldsymbol{G}_{\mathcal{Q}}\left(\mathscr{C}\left(\cdot; u^{(k)}(\cdot), R^{(k)}(\cdot)\right)\right) &= \boldsymbol{G}_{\mathcal{Q}}\left(u^{(k)}(\cdot) + P^{-1}R^{(k)}(\cdot)\right) \\ &= \boldsymbol{u}^{(k)} + \boldsymbol{G}_{\mathcal{Q}}\left(P^{-1}R^{(k)}(\cdot)\right) \\ &= \boldsymbol{u}^{(k)} + \boldsymbol{\Delta}_{\mathcal{Q}}(\boldsymbol{u}^{(k)}) = \boldsymbol{S}_{\mathcal{Q}}(\boldsymbol{u}^{(k)}). \end{aligned}$$

With this observation we can factor the map

$$\boldsymbol{S}_{\mathcal{Q}} \colon \mathcal{U}^{\mathcal{I}} \xrightarrow{J} \mathcal{Q} \otimes \mathcal{U} \xrightarrow{\tilde{S}} \mathcal{Q} \otimes \mathcal{U} \xrightarrow{\boldsymbol{G}_{\mathcal{Q}}} \mathcal{U}^{\mathcal{I}} \tag{3.33}$$

with

$$\begin{aligned} \tilde{S} \colon \mathcal{Q} \otimes \mathcal{U} &\longrightarrow \mathcal{Q} \otimes \mathcal{U} \\ u(\cdot) &\longmapsto \mathscr{C}\Big(\cdot; u(\cdot), R(\cdot; u(\cdot))\Big) \end{aligned} \tag{3.34}$$

such that $\boldsymbol{S}_{\mathcal{Q}} = \boldsymbol{G}_{\mathcal{Q}} \circ \tilde{S} \circ J$ holds. Using this factorization we are now able to prove the contraction factor of the map $\boldsymbol{S}_{\mathcal{Q}}$.

Proposition 3.5. *The map $\tilde{S}$ in* (3.34) *has the same Lipschitz constant $\varrho^* < 1$ as the deterministic solver cycle $\mathscr{C}$ in (3.2).*

Proof. Recall from (3.14 that we assumed the inner product on $\mathcal{Q}$ to be given in the form of an integral $\langle \varphi, \psi \rangle_{\mathcal{Q}} = \int_{\mathcal{P}} \varphi(p)\psi(p)\, \mu(\mathrm{d}p)$. Thus we can identify $\mathcal{Q}$ with $L_2(\mathcal{P}, \mu; \mathbb{R})$ such that $\mathcal{Q} \otimes \mathcal{U}$ is isometrically isomorph to $L_2(\mathcal{P}, \mu; \mathcal{U})$. Using the contraction property (3.7) we have for all $u(\cdot), v(\cdot) \in L_2(\mathcal{P}, \mu; \mathcal{U})$:

$$\begin{aligned} &\left\| \tilde{S}(u(\cdot)) - \tilde{S}(v(\cdot)) \right\|^2_{L_2(\mathcal{P},\mu;\mathcal{U})} \\ &\qquad = \int_{\mathcal{P}} \| S(p; u(p), R(p; u(p)) - S(p; v(p), R(p; v(p)) \|^2_{\mathcal{U}} \, \mu(\mathrm{d}p) \\ &\qquad \le (\varrho^*)^2 \int_{\mathcal{P}} \| u(p) - v(p) \|^2_{\mathcal{U}} \, \mu(\mathrm{d}p) \\ &\qquad = (\varrho^*)^2 \, \| u(\cdot) - v(\cdot) \|^2_{L_2(\mathcal{P},\mu;\mathcal{U})} \, . \end{aligned}$$

The claim follows by taking square roots. □

Corollary 3.6. *The map $\boldsymbol{S}_{\mathcal{Q}}$ in (3.26) is a contraction with Lipschitz factor $\varrho^* < 1$, i.e.*

$$\forall \boldsymbol{u}, \boldsymbol{v} \in \mathcal{U}^{\mathcal{I}} : \quad \| \boldsymbol{S}_{\mathcal{Q}}(\boldsymbol{u}) - \boldsymbol{S}_{\mathcal{Q}}(\boldsymbol{v}) \|_{\mathcal{U}^{\mathcal{I}}} \le \varrho^* \, \| \boldsymbol{u} - \boldsymbol{v} \|_{\mathcal{U}^{\mathcal{I}}} \, ,$$

moreover there exists a unique fixed-point solution $\boldsymbol{u}^* \in \mathcal{U}^{\mathcal{I}}$.

Proof. We use the decomposition (3.33), apply norms and Lemma 3.4, Proposition 3.5 to derive

$$\begin{aligned} \| \boldsymbol{S}_{\mathcal{Q}} \|_{\mathcal{U}^{\mathcal{I}}} &= \left\| \boldsymbol{G}_{\mathcal{Q}} \circ \tilde{S} \circ J \right\|_{\mathcal{U}^{\mathcal{I}}} \\ &\le \| \boldsymbol{G}_{\mathcal{Q}} \|_{\mathcal{U}^{\mathcal{I}}} \cdot \left\| \tilde{S} \right\|_{\mathcal{U}^{\mathcal{I}}} \cdot \| J \|_{\mathcal{U}^{\mathcal{I}}} = \left\| \tilde{S} \right\|_{\mathcal{U}^{\mathcal{I}}} \le \varrho^* < 1. \end{aligned}$$

Applying Banachs fixed-point Theorem 3.1 finishes the proof. □

Using above results, we can now relate the convergence speed of Algorithm 3.2 to the convergence speed of Algorithm 3.1.

Theorem 3.7. *Algorithm 3.2 converges with the same linear speed as Algorithm 3.1. Moreover, the following a posteriori error estimate holds:*

$$\left\|\boldsymbol{u}^* - \boldsymbol{u}^{(k+1)}\right\|_{\mathcal{U}^\mathcal{I}} \leq \frac{\varrho^*}{1-\varrho^*} \left\|\boldsymbol{\Delta}_\mathcal{Q}(\boldsymbol{u}^{(k)})\right\|_{\mathcal{U}^\mathcal{I}} .$$

Proof. This is a consequence from Corollary 3.6, the error estimate is proved by Banachs fixed-point Theorem 3.1. □

3.3. Non-intrusive methods

Now we introduce the non-intrusive methods. The first is the interpolation method – a non-intrusive computation of identity (3.24) with $\boldsymbol{\Delta}_\mathcal{Q}$ in the form of (3.23). The second method is the discrete projection, which relies on the best approximation property.

3.3.1. Numerical integration

Both methods use numerical integration and interpolation, for which we define some notation. We want to approximate an integral representation by some numerical quadrature in the form of

$$\int_\mathcal{P} \phi(p)\, \mu(\mathrm{d}p) \approx \sum_{z=1}^{Z} w_z \phi(p_z) \tag{3.35}$$

for quadrature points $p_z, z = 1, \ldots, Z$ and associated weights w_z. We will use Gauss-Legendre quadrature later in the example (Chapter 3.4). A more sophisticated quadrature is developed in Chapter 5.

Using the above approximation we can compute (3.23) without any further assumptions on A as

$$\boldsymbol{G}_\mathcal{Q}\left(\varphi_\alpha(\cdot) P^{-1} R^{(k)}\right) \approx \Delta_{Z,\alpha} u^{(k)} := \sum_{z=1}^{Z} w_z \varphi_\alpha(p_z) \Delta u_z^{(k)}, \tag{3.36}$$

where

$$\Delta u_z^{(k)} := \mathscr{C}\Big(p; u^{(k)}(p_z), R^{(k)}(p_z)\Big) \tag{3.37}$$

for the expansion

$$u^{(k)}(p_z) = \sum_{\alpha\in\mathcal{I}} u_\alpha^{(k)} \psi_\alpha(p_z).$$

In case the deterministic solver cycle $\mathscr{C}$ in (3.2) returns the next iterate $u^{(k+1)}$ instead of the preconditioned residual $P^{-1}R^{(k)}$, equation (3.37) becomes

$$\Delta u_z^{(k)} = \mathscr{C}\Big(p; u^{(k)}(p_z), R^{(k)}(p_z)\Big) - u^{(k)}.$$

3.3.2. Interpolation method

In this section we present two algorithms, one is the non-intrusive counterpart of Algorithm 3.1, the other computes the preconditioned residuals non-intrusively using the deterministic solver cycle $\mathscr{C}$ and numerical integration. The former is given in Algorithm 3.3. It is a variant of the block Jacobi Algorithm 3.1, iterating equation (3.25). We denote the coefficients with $\tilde{\boldsymbol{u}}$ to emphasize the approximation $\boldsymbol{\Delta}_Z$ of $\boldsymbol{\Delta}_\mathcal{Q}$, in contrast to the exact Algorithm 3.2.

Algorithm 3.4 computes the increment $\boldsymbol{\Delta}_Z(\tilde{\boldsymbol{u}}^{(k)})$ for Algorithm 3.3. It is an approximation by numerical integration

$$\boldsymbol{\Delta}_\mathcal{Q}(\boldsymbol{u}^{(k)}) = \left[\dots, \boldsymbol{G}_\mathcal{Q}\left(\varphi_\alpha(\cdot)P^{-1}R^{(k)}\right), \dots\right] \approx \Delta_Z(\boldsymbol{u}^{(k)}) = \left[\dots, \Delta_{Z,\alpha}u^{(k)}, \dots\right] \tag{3.38}$$

which is computed using (3.36).

Algorithm 3.3 block Jacobi iteration of (3.25)

```
Use some initial guess $\tilde{\boldsymbol{u}}^{(0)} = [\dots, \tilde{u}_\alpha^{(0)}, \dots] \in \mathbb{R}^{N\times M}$
$k \leftarrow 0$
**while** *no convergence* **do**
    Compute $\boldsymbol{\Delta}_Z(\tilde{\boldsymbol{u}}^{(k)})$ using Algorithm 3.4
    $\tilde{\boldsymbol{u}}^{(k+1)} \leftarrow \tilde{\boldsymbol{u}}^{(k)} + \boldsymbol{\Delta}_Z(\tilde{\boldsymbol{u}}^{(k)})$
    $k \leftarrow k+1$
**end while**
```

Algorithm 3.4 non-intrusive residual (3.23) in the form (3.38)

```
**for** $\alpha \in \mathcal{I}$ **do**
    $\Delta_{Z,a}u^{(k)} \leftarrow \mathbf{0}_\mathcal{U}$
**end for**

**for** $z \leftarrow 1, \dots, Z$ **do**
    Compute $\Delta u_z^{(k)}$ using (3.36)
    $r_z \leftarrow w_z \Delta u_z^{(k)}$
    **for** $\alpha \in \mathcal{I}$ **do**
        $\Delta_{Z,\alpha}u^{(k)} \leftarrow \Delta_{Z,\alpha}u^{(k)} + \psi_\alpha(p_z)r_z$
    **end for**
**end for**
```

Now the question arises how well the original sequence $\{\boldsymbol{u}^{(k)}\}_k$ is approximated by $\{\tilde{\boldsymbol{u}}^{(k)}\}_k$.

Theorem 3.8 ([MZ12, Thm. 4.1])**.** *Assume the numerical integration in Algorithm 3.4 to satisfy*

$$\left\|\boldsymbol{G}_\mathcal{Q}\left(\psi_\alpha(\cdot)P^{-1}R(\cdot;\tilde{u}^{(k)})\right) - \Delta_{Z,\alpha}\tilde{u}^{(k)}\right\|_{\mathcal{U}^\mathcal{I}} \leq \frac{\epsilon}{\sqrt{M}}. \tag{3.39}$$

The error in (3.38) is estimated by

$$\left\| \boldsymbol{\Delta}_{\mathcal{Q}}(\tilde{\boldsymbol{u}}^{(k)}) - \boldsymbol{\Delta}_{Z}(\tilde{\boldsymbol{u}}^{(k)}) \right\|_{\mathcal{U}^{\mathcal{I}}} \leq \epsilon \tag{3.40}$$

and the a posteriori error estimate holds

$$\left\| \boldsymbol{u}^* - \tilde{\boldsymbol{u}}^{(k+1)} \right\|_{\mathcal{U}^{\mathcal{I}}} \leq \frac{\varrho^*}{1-\varrho^*} \left\| \boldsymbol{\Delta}_{Z}(\tilde{\boldsymbol{u}}^{(k)}) \right\|_{\mathcal{U}^{\mathcal{I}}} + \frac{\epsilon}{1-\varrho^*}. \tag{3.41}$$

Moreover, it holds

$$\limsup_{k\to\infty} \left\| \boldsymbol{u}^* - \tilde{\boldsymbol{u}}^{(k)} \right\|_{\mathcal{U}^{\mathcal{I}}} \leq \frac{\epsilon}{1-\varrho^*}. \tag{3.42}$$

Proof. Squaring equation (3.39) followed by summing up over $\mathcal{I}$ yields

$$\sum_{\alpha\in\mathcal{I}} \left\| \boldsymbol{G}_{\mathcal{Q}} \left(\psi_\alpha(\cdot) P^{-1} R(\cdot; \tilde{u}^{(k)}) \right) - \Delta_{Z,\alpha} \tilde{u}^{(k)} \right\|^2_{\mathcal{U}^{\mathcal{I}}} \leq M \frac{\epsilon^2}{M}.$$

Using the triangle inequality and taking square root proves (3.40). The other identities are proved in [MZ12, Thm. 4.1]. □

Equation (3.41) shows that the approximated sequence doesn't necessarily converge to $\boldsymbol{u}^*$, even if $\boldsymbol{\Delta}_Z(\boldsymbol{u}^{(k)}) \to 0$ for $k \to \infty$, but by (3.42) the sequence clusters around $\boldsymbol{u}^*$ in an ϵ–neighborhood.

3.3.3. Discrete projection

Discrete projection is as an alternative to the interpolation method for computing the coefficients. Recall that for Hilbert spaces $\mathcal{Q}, \mathcal{Q}_{\mathcal{I}}$ an orthonormal projection onto the subspace $\mathcal{Q} \to \mathcal{Q}_{\mathcal{I}}$ has minimal error in $\|\cdot\|_{\mathcal{Q}}$ [KS10, Satz 2.8]. With $\{\varphi_\alpha\}_{\alpha\in\mathcal{I}}$ being orthonormal, such a projection $u^* \mapsto u_{\mathcal{I}}$ is given by

$$u^\alpha = \langle \varphi_\alpha, u^* \rangle_{\mathcal{Q}} = \int_{\mathcal{P}} \varphi_\alpha(p) u^*(p)\, \mu(\mathrm{d}p) \approx \sum_{z=1}^{Z} w_z \varphi_\alpha(p_z) u^*(p_z). \tag{3.43}$$

Hence, as an alternative to the interpolation method we can compute the coefficients u^α by Algorithm 3.5. Computing the coefficients with the discrete projection has the benefit of integrating only once to compute the coefficient u_α, rather than integrating for each solver cycle to compute iterates $u_\alpha^{(k)}$ as in the interpolation method. The quadrature method in Chapter 5 possesses some algorithmical overhead, thus we will subsequently focus on this method.

3.3.4. Computational effort

We will show now that the computational effort of the interpolation method and the discrete projection are comparable when using an quadrature method of type 3.3.1.

Algorithm 3.5 Discrete projection for (3.43)

```
for $\alpha \in \mathcal{I}$ do
    $u^\alpha \leftarrow \mathbf{0}_\mathcal{U}$
end for

for $z \leftarrow 1, \dots, Z$ do
    Compute $u(p_z)$ using algorithm 3.1
    $r_z \leftarrow w_z u(p_z)$
    for $\alpha \in \mathcal{I}$ do
        $u^\alpha \leftarrow u^\alpha + \psi_\alpha(p_z) r_z$
    end for
end for
```

When examining Algorithms 3.4, 3.5 we expect the invocations of the solver cycle $\mathscr{C}$ respectively the invocation of Algorithm 3.1 to be the dominating factor, neglecting the additional effort for linear algebra operations and calculating basis functions. For large enough systems this assumption is certainly true.

Recall that we assumed the iteration in Algorithm 3.1 to posses a contraction factor ϱ^*. Assume that this iterations needs L invocations of the solver $\mathscr{C}$ to converge with desired accuracy ϵ_{tol}. The interpolation requires the evaluation of Z solver points for the non-intrusive residual (Algorithm 3.4), and this residual is computed L times by the block Jacobi Algorithm 3.3. Thus we have in total $Z \times L$ invocations of the solver cycle. The discrete projection (Algorithm 3.5) requires for each of the Z integration points the invocation of the deterministic solver. The deterministic solver takes L iterations, hence also here we find $Z \times L$. We conclude that those methods are comparable in terms of computational effort.

3.4. Example

We present an example of the theory developed in the previous sections. Consider the electrical network in Figure 3.2, it has nodes $\{1, \dots, 6\}$ of which node 6 is grounded, hence $u_6 = 0$. We seek the solution for nodes $u_1, \dots, u_5$ in form of a vector $\boldsymbol{u} \in \mathcal{U} = \mathbb{R}^5$ because we can omit the grounded node. Moreover there are resistors R with equal resistance $R = 1/100$.

Ohm's law [Ler97, Chapter 27.4] states that for current f [ampere], voltage u [volt] and resistance R [ohm], it holds

$$\frac{u}{R} = f.$$

Kirchhoffs current law [Che04, Chapter 2.3] states that for any node in an electrical network, the inflow of current equals the outflow.

We formulate the system for the remaining 5 nodes using above laws to derive the formulation

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$$

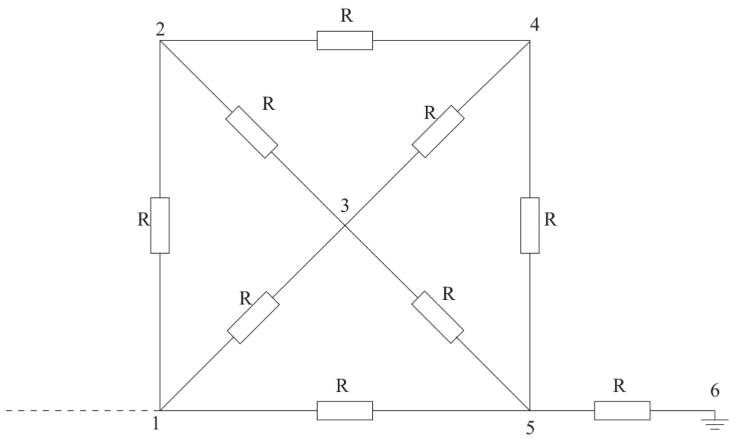


Figure 3.2.: Electrical network with nodes $1, \dots, 6$ and resistors R. Figure taken from [GLL+14].

with

$$\boldsymbol{K} = \frac{1}{R}\begin{pmatrix} 3 & -1 & -1 & 0 & -1 \\ -1 & 3 & -1 & -1 & 0 \\ -1 & -1 & 4 & -1 & -1 \\ 0 & -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & -1 & 4 \end{pmatrix}.$$

We add a global cubic nonlinearity to our system by introducing the term $(\boldsymbol{u}^T\boldsymbol{u})\boldsymbol{u}$, and add uncertainty to the feed-in $\boldsymbol{f}$ and the cubic term such that

$$\begin{aligned} \boldsymbol{A}(\boldsymbol{p};\boldsymbol{u}) &:= \boldsymbol{K}\boldsymbol{u} + (p_1+2)(\boldsymbol{u}^T\boldsymbol{u})\boldsymbol{u} = (p_2+25)\boldsymbol{f}_0 =: \boldsymbol{f}(\boldsymbol{p}), \\ \boldsymbol{f}_0 &:= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \end{pmatrix}^T \end{aligned}$$

for a random parameter $\boldsymbol{p} = (p_1, p_2)$ with p_1, p_2 independently distributed in $[-1, 1]$. This nonlinear terms have no particular physical meaning and are just meant to make the system nonlinear. Still, according to [GLL+14], the term $(\boldsymbol{u}^T\boldsymbol{u})\boldsymbol{u}$ can be interpreted as additional conductivity between the nodes (including node 6), where conductivity increases with the total dissipated power, which is proportional to $(\boldsymbol{u}^T\boldsymbol{u})$.

Note that (3.4) is a concrete example of (3.1), with the residual given as (compare (3.3))

$$R(\boldsymbol{p};\boldsymbol{u}) = (p_2+25)\boldsymbol{f}_0 - \boldsymbol{K}\boldsymbol{u} - (p_1+2)(\boldsymbol{u}^T\boldsymbol{u})\boldsymbol{u}.$$

We choose the preconditioner $P := \boldsymbol{K} = \mathrm{D}_u\boldsymbol{A}(\boldsymbol{p},\boldsymbol{0})$, hence it is independent of $\boldsymbol{p}$ and $\boldsymbol{u}$. We seek the iterative solution to the fixed-point equation

$$\begin{aligned} \boldsymbol{u}^{(k+1)} &= \mathscr{C}\Big(\boldsymbol{p};\boldsymbol{u}^{(k)}, R(\boldsymbol{p};\boldsymbol{u}^{(k)})\Big) = \\ &= P^{-1}\left((p_2+25)\boldsymbol{f}_0 - (p_1+2)\left(\left(\boldsymbol{u}^{(k)}\right)^T\boldsymbol{u}^{(k)}\right)\boldsymbol{u}^{(k)}\right). \end{aligned}$$

For the integration and interpolation we choose a variant of Gauss-Legendre quadrature. Recall that Legendre polynomials ℓ_n of degree $n \in \mathbb{N}_0$ have unit measure

$$\int_{-1}^{1} \ell_n(p)\,\mathrm{d}p = 1$$

and that they are orthogonal

$$\int_{-1}^{1} \ell_n(p)\ell_m(p)\,\mathrm{d}p = 0$$

for $n \neq m$. Moreover it holds

$$\int_{-1}^{1} \ell_n^2(p)\,\mathrm{d}p = \frac{2}{2n+1},$$

so we can normalizing the Legendre polynomials by multiplication with a factor $\sqrt{\frac{2n+1}{2}}$ to derive

$$\int_{-1}^{1} \left(\sqrt{\frac{2n+1}{2}}\ell_n(p)\right)^2 \mathrm{d}p = 1. \tag{3.44}$$

This leads to the following definition.

Definition 3.9. *The multivariant normalized Legendre polynomials in dimension d are constructed as a tensor product of univariant normalized Legendre polynomials*

$$\tilde{L}_\alpha(\boldsymbol{p}) := \prod_{i=1}^{d} \sqrt{\frac{2\alpha_i+1}{2}}\ell_{\alpha_i}(p_i).$$

The multivariant Legendre polynomials are a possible choice for the ansatz functions, another common choice are multivariant Hermite polynomials. Due to their non-compact support we subsequently use Legendre polynomials as ansatz functions, so for the electrical network the ansatz functions are

$$\psi_\alpha(\boldsymbol{p}) := \tilde{L}_\alpha(\boldsymbol{p}) = \prod_{i=1}^{2} \sqrt{\frac{2\alpha_i+1}{2}}\ell_{\alpha_i}(p_i). \tag{3.45}$$

Possible choices for the multi-index set have been introduced in Def. 3.2. For this example we need two-dimensional quadrature, the quadrature points are constructed as cartesian product of one-dimensional Gaussian quadrature points, e.g. for three points in one dimension as

$$(p_{i,j})_{i,j=1,\dots,3} = \begin{pmatrix} -\sqrt{3/5} \\ 0 \\ \sqrt{3/5} \end{pmatrix} \times \begin{pmatrix} -\sqrt{3/5} \\ 0 \\ \sqrt{3/5} \end{pmatrix},$$

with corresponding quadrature weights

$$(w_{i,j})_{i,j=1,\dots,3} = \begin{pmatrix} 5/9 & 8/9 & 5/9 \end{pmatrix} \begin{pmatrix} 5/9 \\ 8/9 \\ 5/9 \end{pmatrix}.$$

Using this we can apply the non-intrusive theory from Chapter 3.3, and represent the solution in terms of (3.9) as

$$\boldsymbol{u}^*(\boldsymbol{p}) \approx \boldsymbol{u}_{\mathcal{I}}(\boldsymbol{p}) = \sum_{\alpha \in \mathcal{I}} \boldsymbol{u}_\alpha \tilde{L}_\alpha(\boldsymbol{p})$$

with $\boldsymbol{u}_\alpha = \left(u_{\alpha,1}, \dots, u_{\alpha,5}\right)^T \in \mathcal{U} = \mathbb{R}^5$ computed non-intrusively.

As convergence criterion we may choose $\left\|\Delta u^{(k)}\right\|_2 > \epsilon_{tol}$ in the deterministic Algorithm 3.1, and $\max_{\alpha \in \mathcal{I}} \left\|\Delta_{Z,\alpha} \tilde{u}^{(k)}\right\|_2 > \epsilon_{tol}$ in the non-intrusive block Jacobi iteration (Algorithm 3.3) for a suitable error tolerance ϵ_{tol}.

We want to compare the results obtained non-intrusively with solutions obtained by the deterministic solver.

Definition 3.10. *The root-mean-squared (RMS) error of N random points $q_n \in [-1,1]^2$ is given by*

$$\epsilon = \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left\|\boldsymbol{u}(q_n) - \boldsymbol{u}_{\mathcal{I}}(q_n)\right\|^2}.$$

where $\boldsymbol{u}(q_n)$ are deterministic solutions obtained by Algorithm 3.1.

Due to the simplicity of this example we refrain from giving examples, and rather discuss the results for the model problem (2.6), (2.7) in Chapter 6.

4. Non-intrusive tensor approximation

In this chapter we construct a low-rank tensor approximation for nonlinear problems. To be precise it is a rank 1 approximation which is non-intrusive and allows rapid evaluation. The approximation is done in a greedy fashion.

Subsequently we replace the greedy update mechanism by a more elaborate algorithm which introduces adaptiveness in terms of the spatial mesh and the stochastic dimensions and stochastic basis elements. However, this technique is only applicable for linear problems.

4.1. General construction

This section follows [GLMN15].

4.1.1. Alternating updates

Here we want to express the approximation $u_r(p) \approx u^*(p)$ of (3.1) in the form of

$$u_r(p) := \sum_{i=1}^{r} \lambda_i(p) v_i, \tag{4.1}$$

where

$$\lambda_i(p) := \psi_{\iota(i)}(p)$$

with a map $\iota : \mathbb{N} \to \mathcal{I}$, such that $\lambda_i(p) \in \mathcal{Q}$ and $v_i \in \mathcal{U}$. In other words, the map ι uniquely identifies a rank i with a multi-index α. We refer to $\lambda_i \otimes v_i$ as rank i, and u_r as rank r approximation.

The following Theorem provides the foundation of the method.

Theorem 4.1 ([GLMN15, Thm. 2.1]). *Assume that*

1. *the map $v \mapsto J(v;p)$ is strongly convex uniformly in p, meaning that there exists a constant $C > 0$ independent of p such that for all $v, w \in \mathcal{U}$ and for all $t \in [0,1]$ we have*

$$J(tv + (1-t)w;p) \le tJ(v;p) + (1-t)J(w;p) - \frac{C}{2}t(1-t)\|v-w\|_{\mathcal{U}}^2,$$

2. *the map $v \mapsto J(v;p)$ is Fréchet differentiable with gradient*

$$\nabla J(v;p) = A(v;p) - b(p),$$

3. *the map* $p \mapsto A(0;p) - b(p)$ *is in* $L^k(\mathcal{P};\mathcal{U})'$ *for some* $k \geq 2$,
4. *the map* $p \mapsto J(v;p)$ *is* μ*-integrable and*
5. *the map* $v \mapsto A(v;p) - b(p)$ *is locally Lipschitz continuous such that for all* $v, w \in \mathcal{U}$ *with* $\max\{\|v\|_{\mathcal{U}}, \|w\|_{\mathcal{U}}\} \leq r$ *it holds*
$$\|A(v;p) - A(w;p)\|_{\mathcal{U}} \leq K(p,r)\,\|v-w\|_{\mathcal{U}},$$
where $K : \mathcal{P} \times \mathbb{R}^+ \to \mathbb{R}^+$,
6. *and finally that for* $R \in L^k(\mathcal{P})$, *the function* $p \mapsto K(p, R(p))$ *is in* $L^j(\mathcal{P})$ *with* $\frac{1}{j} + \frac{2}{k} = 1$.

Then a solution of (3.1) *exists and is unique for all* p *so that we can define a solution map* $u : \mathcal{P} \to \mathcal{U}$. *Moreover,* u *is the unique minimizer in* $L^k(\mathcal{P};\mathcal{U})$ *of the functional*
$$J_{\mathcal{P}} : u \mapsto \int_{\mathcal{P}} J(u(p);p)\,\mu(\mathrm{d}p),$$
and is equivalently characterized by
$$\int_{\mathcal{P}} \langle A(u(p);p) - b(p), \delta u(p)\rangle_{\mathcal{U}}\,\mu(\mathrm{d}p) = 0 \quad \forall \delta u \in L^k(\mathcal{P};\mathcal{U}). \tag{4.2}$$

Assume now we computed u_r and want to compute a new rank $\lambda \otimes v := \lambda_{r+1} \otimes v_{r+1}$. By using Theorem 4.1, we can compute the two updates (λ, v) by solving
$$\min_{(\lambda,v)\in\mathcal{Q}\otimes\mathcal{U}} J_{\mathcal{P}}(u_r + \lambda v).$$
In [GLMN15] it is proposed to use an alternating minimization algorithm, which computes for fixed λ the update v by minimizing
$$\int_{\mathcal{P}} \langle A(u_r(p) + \lambda(p)v;p) - b(p), \lambda(p)\delta v\rangle_{\mathcal{U}}\,\mu(\mathrm{d}p) = 0 \quad \forall \delta v \in \mathcal{U}. \tag{4.3}$$
By setting
$$R_\lambda(v) := \int_{\mathcal{P}} (b(p) - A(u_r(p) + \lambda(p)v;p))\,\lambda(p)\,\mu(\mathrm{d}p) \tag{4.4}$$
we can reformulate (4.3) as
$$\langle R_\lambda(v), \delta v\rangle_{\mathcal{U}} = 0 \quad \forall \delta v \in \mathcal{U}. \tag{4.5}$$
The update λ is obtained similarly for fixed v by solving
$$\int_{\mathcal{P}} \langle A(u_r(p) + \lambda(p)v;p) - b(p), \delta\lambda(p)v\rangle_{\mathcal{U}}\,\mu(\mathrm{d}p) = 0 \quad \forall \delta\lambda \in \mathcal{Q},$$
which can be expressed equivalently as
$$\langle R_v(\lambda), \delta\lambda\rangle_{\mathcal{Q}} = \int_{\mathcal{P}} R_v(\lambda)(p)\delta\lambda(p)\,\mu(\mathrm{d}p) = 0 \quad \forall \delta\lambda \in \mathcal{Q} \tag{4.6}$$
by defining
$$R_v(\lambda) : \quad p \mapsto \langle b(p) - A(u_r(p) + \lambda(p)v;p), v\rangle_{\mathcal{U}}. \tag{4.7}$$
The basic alternating minimization algorithm is given in Algorithm 4.6, the *proper generalized decomposition* (PGD).

Algorithm 4.6 Basic PGD

```
Initialize $u_0$
$r = 0$
while no convergence of $u_r$ do
    Initialize $v, \lambda$
    while no convergence of $\lambda \otimes v$ do
        $\lambda = \lambda / \|\lambda\|_{\mathcal{Q}}$
        $v =$ solve (4.5)
        $v = v / \|v\|_{\mathcal{U}}$
        $\lambda =$ solve (4.6)
    end while
    $u_{r+1} = u_r + \lambda \otimes v$
    $r = r + 1$
end while
```

4.1.2. Computation of the projected residual

It remains to solve equations (4.5), (4.6) for v and λ respectively. Assuming again a quadrature method of type (3.35), we can discretize (4.4) as

$$\begin{aligned} R_\lambda(v) &:= \int_{\mathcal{P}} R\left(u_r(p) + \lambda(p)v; p\right) \lambda(p)\, \mu(\mathrm{d}p) \\ &\approx \sum_{z=1}^{Z} w_z \lambda(p_z) R(u_r(p_z) + \lambda(p_z)v; p_z) \end{aligned}$$

where we use again the short notation for the residual as in (3.3). Hence (4.5) becomes

$$\begin{aligned} \langle R_\lambda(v), \delta v\rangle_{\mathcal{U}} &= \int_{\mathcal{P}} \langle R\left(u_r(p) + \lambda(p)v; p\right), \delta v\rangle_{\mathcal{U}}\, \lambda(p)\, \mu(\mathrm{d}p) \\ &\approx \sum_{z=1}^{Z} w_z \lambda(p_z) \left\langle R\left(u_r(p_z) + \lambda(p_z)v; p_z\right), \delta v\right\rangle_{\mathcal{U}} \quad \forall \delta v \in \mathcal{U}. \end{aligned}$$

Analogously we derive for (4.6) the quadrature

$$\langle R_v(\lambda), \delta\lambda\rangle_{\mathcal{Q}} \approx \sum_{z=1}^{Z} w_z \left\langle R\left(u_r(p_z) + \lambda(p_z)v; p_z\right), v\right\rangle_{\mathcal{U}} \delta\lambda(p_z) \quad \forall \delta v \in \mathcal{U}.$$

Having discretized the integrals, we need to choose a method for solving

$$\begin{aligned} R_\lambda(v) &= -\nabla J_\lambda(v) = 0, \\ R_v(\lambda) &= -\nabla J_v(\lambda) = 0. \end{aligned}$$

Newton's method uses the iteration scheme

$$\begin{aligned} v_{k+1} &= v_k - \left[\nabla R_\lambda(v_k)\right]^{-1} R_\lambda(v_k), \\ \lambda_{k+1} &= \lambda_k - \left[\nabla R_v(\lambda_k)\right]^{-1} R_v(\lambda_k), \end{aligned}$$

but requires the computation of Hessian matrices for the functional. For the computation of v we need to compute

$$-\nabla^2 J_\lambda(v) = -\nabla R_\lambda(v) = \int_{\mathcal{P}} \lambda(p)^2 \nabla A\left(u_r(p) + \lambda(p)v; p\right) \mu(\mathrm{d}p),$$

and for the λ updates we have

$$\begin{aligned}-\nabla^2 J_v(\lambda)(\delta\lambda_1, \delta\lambda_2) &= -\left\langle \nabla R_v(\lambda)\delta\lambda_1, \delta\lambda_2 \right\rangle_{\mathcal{Q}} \\ &:= \int_{\mathcal{P}} \left\langle \nabla A\left(u_r(p) + \lambda(p)v; p\right) v, v \right\rangle_{\mathcal{U}} \delta\lambda_1(p)\delta\lambda_2(p)\, \mu(\mathrm{d}p).\end{aligned}$$

Both Hessian matrices require the matrix $\nabla A(\cdot; p)$, which prevents the application of Newtons method in a non-intrusive, nonlinear setting.

Aiming for nonlinear problems, [GLMN15] propose the usage of a quasi-Newton method, namely the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) method. Using this method, we need only to compute the residuals which can be carried out in a non-intrusive fashion by numerical quadrature. Before giving the algorithm in Algorithm 4.7 we fix some notation. Denote by x one of the updates v, λ, any y the other, moreover $\mathcal{X}$ to be one of the spaces $\mathcal{U}, \mathcal{Q}$ with corresponding inner product $\langle \cdot, \cdot \rangle_{\mathcal{X}}$.

The BFGS algorithm iterates

$$x^{(k+1)} = x^{(k)} + \rho_k C_k R_y(x^{(k)}),$$

where $\rho_k \in \mathbb{R}$ to be explained later, and a matrix C_k being an approximation of the Hessian $\nabla^2 R_y(x_k)$.

The BFGS algorithm starts by initializing an initial value for x, a good choice to gauge convergence is the deterministic solution (e.g. at the origin), but may also be a static defined vector.

We need to choose a suitable initial preconditioner C_0. In case of solving for v such that $R_\lambda(v) = 0$, a reasonable preconditioner is given by the inverse of the system matrix $A(\mathbf{0})$. The algorithm is formulated in a matrix-free manner, such that we only need to store the current preconditioner C_k, and the next preconditioner C_{k+1} is computed in each iteration step based on the current state. To this end we define

$$\begin{aligned} z_k &= -\left(R_y(x_{k+1}) - R_y(x_k)\right), \\ t_k &= x_{k+1} - x_k, \\ s_k &= C_k z_k. \end{aligned}$$

The next preconditioner matrix C_{k+1} is computed recursively by

$$C_{k+1} = C_k + \frac{\langle z_k, t_k \rangle_{\mathcal{X}} + \langle z_k, s_k \rangle_{\mathcal{X}}}{\langle z_k, t_k \rangle_{\mathcal{X}}^2} \left(t_k \otimes t_k\right) - \frac{1}{\langle z_k, t_k \rangle_{\mathcal{X}}} \left(s_k \otimes t_k + t_k \otimes s_k\right) \tag{4.8}$$

and satisfies $C_{k+1} z_k = t_k$.

Finally we describe the scalar factor ρ, it acts as insurance policy against divergence. We introduce the maps

$$\begin{aligned} J_v :\ \lambda &\longmapsto J_{\mathcal{P}}(u_r + \lambda v), \\ J_\lambda :\ v &\longmapsto J_{\mathcal{P}}(u_r + \lambda v). \end{aligned}$$

The factor ρ is computed by a coarse linesearch, meaning that we want to approximate the solution of the problem

$$\min_{\rho \in \mathbb{R}} J_y \left(x^{(k)} + \rho d_l \right).$$

This problem can be solved in the same non-intrusive manner by numerical quadrature. The linesearch can be carried out very coarse, and we may sample only a few ρ here. Due to the convex nature of the functional it is sufficient to evaluate 3 samples and perform a second-order least-square fit on this data to find its minimum. At the minimum it holds

$$\frac{\partial}{\partial \rho} J_y \left(x^{(k)} + \rho d_k \right) = 0,$$

and hence

$$\left\langle R_y \left(x^{(k)} \right), d_l \right\rangle_{\mathcal{X}} = 0.$$

Algorithm 4.7 BFGS algorithm for solving $R_y(x) = 0$

```
Initialize $x^{(0)}$
$C_0 = P_y^{-1}$
$k = 0$
$d_0 = C_k \cdot R_y\left(x^{(0)}\right)$
while no convergence of $x$ do
    $\rho_k = \min_\rho J_y\left(x^{(k)} + \rho d_k\right)$
    $t_k = \rho_k d_k$
    $x^{(k+1)} = x^{(k)} + t_k$
    $d_{k+1} = C_k \cdot R_y\left(x^{(k+1)}\right)$
    compute $C_{k+1}$ according to (4.8)
    $k = k + 1$
end while
return $x$
```

4.2. Adaptive updates

We restrict ourself now to linear problems such as the model problem (2.6), (2.7). For such a linear problem one does not have to employ the BFGS method, but rather compute the updates directly.

Moreover we want to replace the basic PGD algorithm by a variant of the `ASGFEM` algorithm proposed in [EGSZ15], which is only applicable if the problem is linear. This algorithm uses error indicators to select multiindices α to be added to the tensor approximation. Moreover, spatial error indicators decide upon adaptive refinement of the spatial mesh, which is an advantage of using this method. The described routines are provided by the uncertainty quantification framework `ALEA` [EZ] and used in our implementation.

4.2.1. Preliminaries

The basic idea of `ASGFEM` is to adaptively enlarge the stochastic basis (by adding new ranks with new α), or to adaptively refine the spatial mesh. In each step both errors are estimated a posteriori, and the larger receives subsequent refinement.

First, note that for a linear operator A, equation (4.3) becomes

$$\int_{\mathcal{P}} \lambda(p)^2 \left\langle A(v;p), \delta v\right\rangle_{\mathcal{U}} \mu(\mathrm{d}p) = \int_{\mathcal{P}} \lambda(p) \left\langle b(p) - A(u_r(p);p), \delta v\right\rangle_{\mathcal{U}} \mu(\mathrm{d}p) \quad \forall \delta v \in \mathcal{U}, \tag{4.9}$$

such that updates v are computable without the BFGS algorithm by integrating both the left-hand side and the right-hand side and solving the system

$$\left[\sum_{z=1}^{Z_1} w_z \lambda(p_z)^2 A(p_z)\right] v = \left[\sum_{z=1}^{Z_2} w_z \lambda(p_z) \left(b(p_z) - A(u_r(p_z);p_z)\right)\right] \tag{4.10}$$

for v, where we use for clarity the notation $A(p_z)v := A(v;p_z)$. Hence an update v requires two quadratures and one solver call. The quadrature of both sides is performed separately, because the variance of the left-hand side is mainly governed by the factor $\lambda(p)^2$, where the left hand side depends mainly on the factor $\lambda(p)$. There is another reason for this resulting from using sparse grid quadrature, which will be discussed in Chapter 5.3.3.

In contrast to the discrete projection method, we do not choose a predefined set $\mathcal{I}$ but construct the multi-index set $\mathcal{I}$ adaptively. We need to adapt the definition slightly.

Definition 4.2. *The index set $\mathcal{I}$ is a subset of*

$$\mathcal{F} := \{\alpha \in \mathbb{N}_0^\infty \mid \operatorname{supp}(\alpha) < \infty\}$$

with support

$$\operatorname{supp}(\alpha) := \{m \in \mathbb{N} \mid \alpha_m \neq 0\},$$

furthermore we define for any subset $\mathcal{I} \subset \mathcal{F}$ the support

$$\operatorname{supp}(\mathcal{I}) := \bigcup_{\alpha \in \mathcal{I}} \operatorname{supp}(\alpha).$$

Let $e_m := (\delta_{mn})_{n=1}^\infty$ denote the Kronecker sequence. The boundary of $\mathcal{I}$ is the infinite set

$$\partial\mathcal{I} := \{\alpha \in \mathcal{F} \setminus \mathcal{I} \mid \exists m \in \mathbb{N} : \nu - e_m \in \mathcal{I} \vee \nu + e_m \in \mathcal{I}\}$$

and the active boundary

$$\partial^\circ\mathcal{I} := \{\nu \in \mathcal{F} \setminus \mathcal{I} \mid \exists m \in \operatorname{supp}(\mathcal{I}) : \nu - e_m \in \mathcal{I} \vee \nu + e_m \in \mathcal{I}\}$$

which is for finite $\mathcal{I}$ a finite set with cardinality

$$|\partial^\circ \mathcal{I}| \leq 2 \cdot |\mathcal{I}| \cdot |\mathrm{supp}(\mathcal{I})|.$$

4.2.2. Spatial error estimator and refinement

We assume a spatial mesh $\mathcal{T}$ consisting of triangular cells $T \in \mathcal{T}$. The spatial residual error estimator considers each mesh cell T for all $\alpha \in \mathcal{I}$. The error per cell T and $\alpha \in \mathcal{I}$ is given by

$$\eta_{\alpha,T}(u_r) = \left(h_T^2 \left\| \bar{\kappa}^{-1/2} \left(f\delta_{\alpha 0} + \nabla \cdot \sigma_\alpha(u_r) \right) \right\|_{L^2(T)}^2 + h_T \left\| \bar{\kappa}^{-1/2}, [\![\sigma_\alpha(u_r)]\!] \right\|_{L^2(\partial T \cap D)}^2 \right)^{1/2},$$

where $[\![\cdot]\!]$ denotes the normal jump over the facets of a triangular cell T and σ the flow such that

$$[\![\sigma]\!] := \sigma|_{T_1} \cdot n_1 + \sigma|_{T_2} \cdot n_2$$

for n_i the exterior unit normal of a facet T_i, for details we refer to [EGSZ15].

Summing over $\alpha \in \mathcal{I}$, we define the error of a mesh cell T as

$$\eta_T(u_r, \mathcal{I}) := \left(\sum_{\alpha \in \mathcal{I}} \eta_{\alpha,T}(u_r)^2 \right)^{1/2}, \tag{4.11}$$

and for any subset $\mathcal{M} \subset \mathcal{T}$ the global error

$$\eta(u_r, \mathcal{I}, \mathcal{M}) := \left(\sum_{T \in \mathcal{M}} \eta_T(u_r, \mathcal{I})^2 \right)^{1/2}. \tag{4.12}$$

The routine

$$\eta(u_r, \mathcal{I}, \mathcal{T}),\ \{\eta_T(u_r, \mathcal{I})\}_{T \in \mathcal{T}} \leftarrow \texttt{Estimate}_x(u_r, \mathcal{I}, \mathcal{T})$$

outputs the error indicators (4.11), (4.12).

Next, the cell indicators are used to decide which cell has to be refined. For a weighting parameter $0 < \vartheta_x < 1$, the marking routine

$$\mathcal{M} \leftarrow \texttt{Mark}_x \left(\eta(u_r, \mathcal{I}, \mathcal{T}),\ \{\eta_T(u_r, \mathcal{I})\}_{T \in \mathcal{T}},\ \vartheta_x \right)$$

returns a subset $\mathcal{M} \subset \mathcal{T}$ satisfying the Dörfler property

$$\eta(u_r, \mathcal{I}, \mathcal{M}) \geq \vartheta_x \cdot \eta(u_r, \mathcal{I}, \mathcal{T}).$$

The subset $\mathcal{M}$ is used in the subsequent mesh refinement of the routine

$$\mathcal{T} \leftarrow \texttt{Refine}_x(\mathcal{T}, \mathcal{M})$$

where each cell in $\mathcal{M}$ is refined at least once.

4.2.3. Stochastic truncation error estimator and refinement

The multiindices $\alpha \in \mathcal{I}$ correspond to stochastic basis functions $\lambda_{\iota^{-1}(\alpha)}$. The multiindices $\mathcal{F} \setminus \mathcal{I}$ can be seen as basis elements which haven't been added to the basis yet. Hence we term the estimator a truncation error, or tail refinement.

As the construction of multiindices $\alpha \in \mathcal{I}$ suggests, we don't assume a fixed stochastic dimension now, but rather start with a 1-dimensional approximation and enlarge the set $\mathcal{I}$ adaptively. By enlarging we mean the addition of multiindices, either in the current dimension or a higher dimensional multi-index.

For any $\alpha \in \partial\mathcal{I}$ the error is estimated by

$$\zeta_\alpha(u_r) = \sum_{m=1}^{\infty} \left\| \frac{\kappa_m}{\bar{\kappa}} \right\|_{L^\infty(D)} \left(\beta^m_{\alpha_m+1} \left\| u_{r,\alpha+\epsilon_m} \right\|_V + \beta^m_{\alpha_m} \left\| u_{r,\alpha-\epsilon_m} \right\|_V \right), \tag{4.13}$$

for details see again [EGSZ15]. The summation of (4.13) over supp$(\mathcal{I})$ is a finite sum since all other terms are zero. For any subset $\Delta \subset \partial\mathcal{I}$ we define

$$\zeta(u_r, \Delta) := \left(\sum_{\alpha\in\Delta} \zeta_\alpha(u_r)^2 \right)^{1/2}.$$

Due to the infinite cardinality of $\partial\mathcal{I}$ the sum $\zeta(u_r, \partial\mathcal{I})$ is an infinite sum. However, for $\alpha \in \partial\mathcal{I} \setminus \partial^\circ\mathcal{I}$, i.e., $\alpha = \alpha' + \epsilon_m$ for $\alpha' \in \mathcal{I}$ and $m \in \mathbb{N} \setminus \operatorname{supp}(\mathcal{I})$ it holds

$$\zeta_\alpha(u_r) = \left\| \frac{\kappa_m}{\bar{\kappa}} \right\|_{L^\infty(D)} \beta^m_1 \left\| u_{r,\alpha'} \right\|_V.$$

Summing there terms over all inactive dimensions m leads to the lumped error indicator

$$\begin{aligned}
\bar{\zeta}_{\alpha'}(u_r, \mathcal{I}) &:= \left(\sum_{m\in\mathbb{N}\setminus\operatorname{supp}(\mathcal{I})} \zeta_{\alpha'+\epsilon_m}(u_r)^2 \right)^{1/2} \\
&= \left\| u_{r,\alpha'} \right\|_V \left(\sum_{m\in\mathbb{N}\setminus\operatorname{supp}(\mathcal{I})} \left(\left\| \frac{\kappa_m}{\bar{\kappa}} \right\|_{L^\infty(D)} \beta^m_1 \right)^2 \right)^{1/2}
\end{aligned}$$

for $\alpha' \in \mathcal{I}$. The infinite sum remaining in $\bar{\zeta}_{\alpha'}(u_r, \mathcal{I})$ is independent of u_r and α', depending only on supp$(\mathcal{I})$. Thus we can represent $\zeta(u_r, \partial\mathcal{I})$ by the finite sum

$$\zeta(u_r, \partial\mathcal{I})^2 = \sum_{\alpha\in\partial^\circ\mathcal{I}} \zeta_{\alpha'}(u_r)^2 + \sum_{\alpha'\in\mathcal{I}} \bar{\zeta}_\alpha(u_r, \mathcal{I})^2. \tag{4.14}$$

The routine

$$\zeta\left(u_r, \partial\mathcal{I}\right), \left\{\zeta_\alpha(u_r)\right\}_{\alpha\in\partial^\circ\mathcal{I}} \leftarrow \texttt{Estimate}_y\left(u_r, \mathcal{I}\right)$$

returns the global error indicator (4.14) and the error indicators (4.13) on the active boundary.

Next the choice of multiindices to be included in $\mathcal{I}$ has to be made. We allow multiple new multiindices to be added at a time, as to be explained in the next section. The choice of multiindices is carried out by the routine

$$\Delta \leftarrow \texttt{Mark}_y \left(\{\zeta_\alpha(u_r)\}_{\alpha \in \partial^\circ \mathcal{I}} \,, \vartheta_y \right)$$

with a user supplied choice of weighting parameter $0 < \vartheta_y < 1$. It constructs a nonempty set of new multiindices Δ such that the Dörfler property

$$\zeta\left(u_r, \Delta\right) \geq \vartheta_y \cdot \zeta\left(u_r, \partial\mathcal{I}\right).$$

is satisfied. The routine selects not only from multiindices on the active boundary. Suppose the current dimension is d; every time a multi-index of the for $\nu = \alpha + e_d$ with $\alpha \in \mathcal{I}$ is added to Δ the routine also estimates indices $\alpha + e_{d+1}$. In doing so we achieve dimension-adaptiveness in the sense that the truncation error governs the dimensionality of the tensor approximation.

The simplest choice for a new set of multiindices is $\mathcal{I}^* = \mathcal{I} \cup \Delta$, but the algorithm is not restricted in such a way. If favorable, other properties besides downward closedness of $\mathcal{I}$ can be enforced. To this end a larger set $\mathcal{I} \cup \Delta \subset \mathcal{I}^* \subset \partial\mathcal{I}$ is chosen, which can be implemented in the optional routine

$$\mathcal{I}^* \leftarrow \texttt{Refine}_y\left(\mathcal{I}, \Delta\right).$$

4.2.4. Algorithm

We formulate the `ASGFEM` update mechanism presented in Algorithm 4.8. It should be noted that the initialization of $\mathcal{I}$ is not limited to $d = 1$, as well as we may supply further predefined $\alpha \in \mathcal{I}$ besides $\mathbf{0}$. We assume the output of $\texttt{Refine}_y$ to be a set of new multiindices Δ as in $\texttt{Mark}_y$. By abuse of notation and for sake of clarity we denote by α_j the multi-index corresponding to rank j, and not its components. The order of multiindices added to the approximation in one step is not important. We just check if such a multi-index has larger dimension than the maximal dimension in $\mathcal{I}$, and if so we process all new multiindices in this new dimension.

An initial spatial mesh $\mathcal{T}$ has to be supplied, the triangulation of the spatial mesh may be limited by an appropriate criterion, e.g., the number of vertices. When the spatial mesh $\mathcal{T}$ of a rank r approximation is refined, all r ranks have to be updated. We refrain from interpolating $v_i, i = 1, \dots, r$ onto the new meshes, since only ranks $> r$ will profit from this refinement. Rather we perform an update where the tensor approximation is reconstructed, in the sense that we recompute all $v_i, i = 1 \dots, r$ successively using the spatial triangulation. Thus, the update step is expensive in terms of quadratures required.

Algorithm 4.8 Tensor approximation coupled with ASGFEM

```
$\mathcal{I} = \{\mathbf{0} =: \alpha_1\}$
$v_1 \leftarrow$ solve (4.10) for $\lambda_{\iota^{-1}(\alpha_1)}$ using initial mesh $\mathcal{T}$
$r \leftarrow 1$
while no convergence of $u_r$ do
    $\eta(u_r, \mathcal{I}, \mathcal{T}),\ \{\eta_T(u_r, \mathcal{I})\}_{T \in \mathcal{T}} \leftarrow \texttt{Estimate}_x(u_r, \mathcal{I}, \mathcal{T})$
    $\zeta(u_r, \partial\mathcal{I}),\ \{\zeta_\alpha(u_r)\}_{\alpha \in \partial^\circ \mathcal{I}} \leftarrow \texttt{Estimate}_y(u_r, \mathcal{I})$
    if $\eta(u_r, \mathcal{I}, \mathcal{T}) \geq \zeta(u_r, \partial\mathcal{I})$ then                ▷ spatial refinement
        $\mathcal{M} \leftarrow \texttt{Mark}_x\left(\eta(u_r, \mathcal{I}, \mathcal{T}),\ \{\eta_T(u_r, \mathcal{I})\}_{T \in \mathcal{T}},\ \vartheta_x\right)$
        $\mathcal{T} \leftarrow \texttt{Refine}_x(\mathcal{T}, \mathcal{M})$
        for $j = 1, \dots, r$ do                                        ▷ update meshes
            $v_j \leftarrow$ recompute (4.10) for $\lambda_{\iota^{-1}(\alpha_j)}$ using $\mathcal{T}$
        end for
    else                                                          ▷ enlarge stochastic basis
        $\Delta \leftarrow \texttt{Mark}_y\left(\{\zeta_\nu(u_r)\}_{\nu \in \partial^\circ \mathcal{I}},\ \vartheta_y\right)$
        $\Delta \leftarrow \texttt{Refine}_y(\mathcal{I}, \Delta)$                               ▷ optional
        for $\alpha \in \Delta$ do                                          ▷ add new ranks
            $r \leftarrow r + 1$
            $\alpha_r \leftarrow \alpha$
            $\mathcal{I} \leftarrow \mathcal{I} \cup \alpha_r$
            $v_r \leftarrow$ solve (4.10) for $\lambda_{\iota^{-1}(\alpha_r)}$ using $\mathcal{T}$
        end for
    end if
end while
return $\sum_{j=1}^{r} \lambda_{\iota^{-1}(\alpha_j)}(\cdot) v_j$
```

5. Sparse Grids

For the electric example network Chapter 3.4 we performed quadrature on a two-dimensional grid constructed as cartesian product of one-dimensional grid points. We call such a grid a *full grid*, and clearly the number of grid points in dimension d is N^d, where N is the number of one-dimensional grid points. This exponential behavior was termed *curse of dimensionality* by Richard Bellman [Bel72][1]. Even for moderate choices of N, the exponential growth of grid points limits the application of such grids to low-dimensional problems.

Sparse grids delay the curse of dimensionality by a more elaborate construction than full grid, leading to significantly fewer grid points. Using such grids, it is possible to extend the non-intrusive methods to higher-dimensional domains. The construction was proposed in 1963 by Soviet mathematician Sergey A. Smolyak [Smo63][2]. With the introduction of powerful computers in the 1990s, sparse grids became a popular technique with applications in many scientific areas. They are still an active field of research.

5.1. Classical sparse grids

The following material is gathered from [Pfl10], [Feu10], [Ach03].

5.1.1. Preliminaries

The goal is to approximate a function $f : \mathcal{P} \to \mathbb{R}$ with compact domain $\mathcal{P} \subset \mathbb{R}^d$, and without loss of generality we assume the domain the be the d-dimensional hypercube $\mathcal{P} = [0,1]^d$. By approximation we mean quadrature as well as interpolation. The domain boundary will be denoted as $\partial\mathcal{P}$ and the interpolant of f is denoted by u.

We use several terms related to grids in higher-dimensional settings as well as in the one-dimensional case: *grid*, *grid points* $x = (x_1, \dots, x_d) \in \mathcal{P}$ and step size h_l. We will frequently use d-dimensional multiindices $\boldsymbol{l} := (l_1, \dots, l_d)$ with the usual norms

$$|\boldsymbol{l}|_1 := \sum_{j=1}^{d} |l_j|, \quad |\boldsymbol{l}|_\infty := \max_{1 \le j \le d} |l_j|.$$

By abuse of notation we define for multiindices $\boldsymbol{l}, \boldsymbol{k}$ the relation

$$\boldsymbol{l} \le \boldsymbol{k} \;\Leftrightarrow\; l_j \le k_j,\, j = 1, \dots, d,$$

and moreover the multi index $\mathbf{1} = (1, \dots, 1)$.

[1]cited after [Feu10]
[2]cited after [GG98]

5.1.2. Hierarchical basis in one dimension

For now consider the one-dimensional case, i.e., a function $f : \mathcal{P} = \mathbb{R} \to \mathbb{R}$ to be approximated which vanishes on the boundary.

Definition 5.1. *The standard hat function is given by*

$$\Lambda\colon \mathbb{R} \to \mathbb{R}, \quad \Lambda(x) = \begin{cases} 1 - |x| & x \in [-1, 1], \\ 0 & \textit{else}. \end{cases}$$

By translating and dilating we derive the one-dimensional standard hat basis

$$\begin{aligned} \Lambda_{l,i}\colon [0,1] &\longrightarrow \mathbb{R}, \\ x &\longmapsto \Lambda\left(2^l x - i\right), \end{aligned}$$

with $l \in \mathbb{N}_0, 0 \le i \le 2^l$.

Definition 5.2. *The one-dimensional grid of level $l \in \mathbb{N}, l \geq 1$ (without boundary) is given by*

$$G_l^1 := \left\{ x_{l,i} := i h_l \mid 0 < i < 2^l \right\} \tag{5.1}$$

with step size $h_l = 2^{-l}$ and grid points $x_{l,i}$. The index i specifies the location of the grid point $x_{l,i}$ within the level l. The nodal basis space is defined to be

$$V_l^1 := \operatorname{span}\left\{ \Lambda_{l,i} \mid 0 < i < 2^l \right\}.$$

The generalized hat functions are centered at points $x_{l,i}$ (compare Figure 5.1). The space V_n^1 has the property of nonempty support with its neighboring functions, that is $\operatorname{supp}(\Lambda_{l,i}) = [x_{l,i-1}, x_{l,i+1}]$.

We now decompose this space into subspaces possessing disjoint supports on each level respectively.

Definition 5.3. *A one-dimensional hierarchical index set of level $l \geq 1$ (without boundary) is defined as*

$$I_l^1 := \left\{ i \in \mathbb{N} \mid 0 < i < 2^l, i \textit{ odd} \right\}$$

The one-dimensional hierarchical subspace of level $l \geq 1$ (without boundary) is defined as

$$W_l^1 := \operatorname{span}\left\{ \Lambda_{l,i} \mid i \in I_l^1 \right\}.$$

Distinct basis functions of W_l^1 have local support, such that the d-dimensional volume vanishes

$$\operatorname{vol}_d\left(\operatorname{supp}(\Lambda_{l,i_1}) \cap \operatorname{supp}(\Lambda_{l,i_2})\right) = 0,$$

while the support of basis functions from different levels might be nonempty depending on i (compare once more Figure 5.1). Note that we have the identities

$$\begin{aligned} V_1^1 &= W_1^1, \\ V_l^1 &= V_{l-1}^1 \oplus W_l^1, \quad l \geq 2, \end{aligned} \tag{5.2}$$

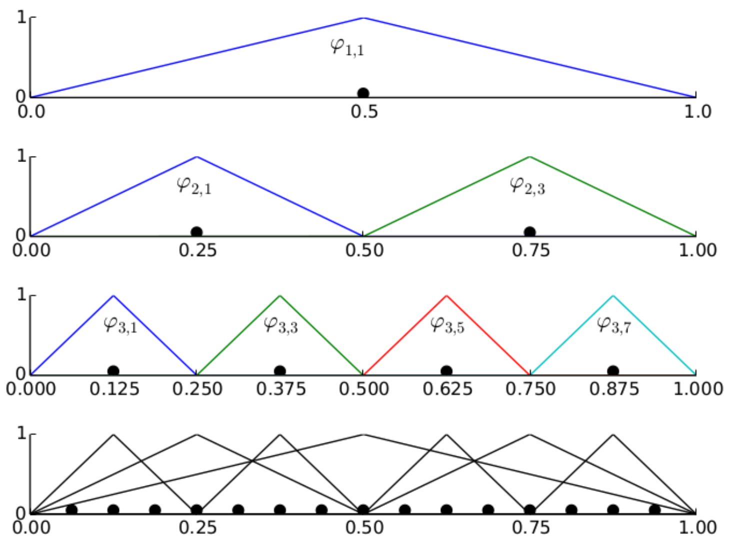


Figure 5.1.: Shown in the first three rows the one-dimensional spaces W_1, W_2, W_3 consisting of standard hat basis functions; black points indicate the corresponding grid points $x_{l,i}$. The approximation V_3 is shown in the fourth row, it is the direct sum of the above (compare (5.4)).

which imply the decomposition

$$V = \bigoplus_{j=1}^{\infty} W_j^1 \tag{5.3}$$

with an approximation of level l

$$V \approx V_l^1 = \bigoplus_{j=1}^{l} W_j^1. \tag{5.4}$$

Definition 5.4. *One-dimensional interpolations of level l can be written in the form*

$$f(x) \approx \Upsilon_l^1(x) := \sum_{1\le j\le l} \sum_{i\in I_j^1} \omega_{l,i} \Lambda_{l,i}(x) \tag{5.5}$$

with hierarchical surplusses $\omega_{l,i}$ corresponding to grid points $x_{l,i}$. In the same manner, one-dimensional integration of level l can be written as

$$\int_{\mathcal{P}} f(x)\,\mathrm{d}x \approx \Phi_l^1(f) := \sum_{1\le j\le l} \sum_{i\in I_j^1} \omega_{l,i} \int_{\mathcal{P}} \Lambda_{l,i}(x)\,\mathrm{d}x. \tag{5.6}$$

Now the hierarchical surplusses have to be computed, we use an idea dating back to Archimedes [Edw12]. Archimedes considered a parabola and approximated its area by subsequently inscribing triangles of smaller size as shown in Figure 5.2. Our problem is

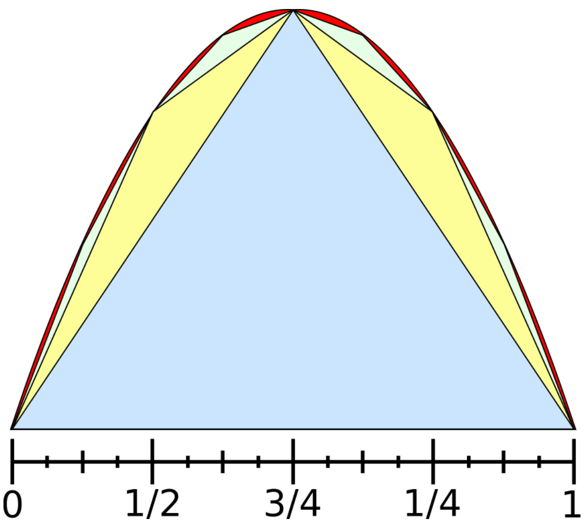


Figure 5.2.: Archimedes' idea of the numerical quadrature of a parabola: subsequent inscribing of triangles. The same principle is used with hierarchical hat basis functions. Figure taken from [Bel].

quite the same. We start on level $l = 1$ by setting the hierarchical surplus to $\omega_{1,1} := f(x_{1,1})$. For levels $l \geq 2$, we evaluate all grid points corresponding to that level – that is $f(x_{l,i})$ for $x_{l,i} \in G_n$. Next, we subtract for each point the interpolation of the lower levels. So, the hierarchical surplusses in one dimension are computed successively levelwise as the difference of the function value and the interpolation of lower levels at that point

$$\omega_{l,i} = f(x_{l,i}) - \Upsilon^1_{l-1}(x_{l,i}). \tag{5.7}$$

If the function f is twice continuously differentiable, it holds the estimate

$$\omega_{l,i} \approx -\frac{1}{2} h_l^2 \cdot \frac{\partial^2 f}{\partial x^2}(x_{l,i})$$

with an estimate

$$|\omega_{l,i}| \leq C \cdot 2^{-2l-1} \cdot \left\| \frac{\partial^2 f}{\partial x^2} \right\|_\infty \tag{5.8}$$

with a constant C independent of the step size h_l. It follows that the decay of hierarchical surplusses is $\mathcal{O}\left(4^{-l}\right)$, supporting our intuition that higher levels contribute less to the approximation.

Remark 5.5. *While we use only equidistant grids* (5.1)*, other choices have been proposed. The Clenshaw–Curtis formulas introduce non-equidistant grid points defined by the extreme points (or zeros) of Chebyshev polynomials. For the decomposition* (5.4) *it is crucial to have nested grid points, which is only satisfied using the extreme points. The Filippi formulas are a variant of Clenshaw–Curtis which omits boundary points. The Gauss–Patterson formulas are a nested extension of the common Gauss–Legendre quadrature which is in general not nested. See [GG98] and references therein.*

5.1.3. Higher dimensions

The extension to higher dimensions is based on tensoring of the one-dimensional construction.

Definition 5.6. *The d-dimensional hat basis is defined as*

$$\begin{aligned}\Lambda_{\boldsymbol{l},\boldsymbol{i}} \colon [0,1]^d &\longrightarrow \mathbb{R} \\ \boldsymbol{x} &\longmapsto \prod_{j=1}^{d} \Lambda_{l_j,i_j}(x_j),\end{aligned}$$

for $\boldsymbol{x} = (x_1, \dots, x_d)$ and $l_j \in \mathbb{N}_0, i_j = 0, \dots, 2^{l_j}$.

Definition 5.7. *The d-dimensional grid of level $\boldsymbol{l} = (l_1, \dots, l_d) : l_j \geq 1, j = 1, \dots, d$ (without boundary) is given by the cartesian product of one-dimensional grids*

$$\begin{aligned}G_{\boldsymbol{l}} &:= \bigtimes_{j=1}^{d} G^1_{l_j} \\ &= \left\{ x_{\boldsymbol{l},\boldsymbol{i}} := (x_{l_1,i_1}, \dots, x_{l_d,i_d}) \mid \forall j = 1, \dots, d : x_{l_1,i_1} \in G^1_{l_j} \right\}\end{aligned}$$

with grid points $x_{\boldsymbol{l},\boldsymbol{i}}$ and step sizes $(2^{-h_1}, \dots, 2^{-h_d})$. The space spanned by the piecewise linear generalized hat functions of level $n \geq 1$ (without boundary) is denoted by

$$V_{\boldsymbol{l}} := \operatorname{span} \left\{ \Lambda_{\boldsymbol{l},\boldsymbol{i}} \mid 0 < i_j < 2^{l_j}, 1 \leq j \leq d \right\}.$$

Definition 5.8. *Introducing the d-dimensional index set (without boundary) corresponding to the multi-index $\boldsymbol{l} = (l_1, \dots, l_d)$ as*

$$\begin{aligned}I_{\boldsymbol{l}} &:= \bigtimes_{j=1}^{d} I^1_{l_j} \\ &= \left\{ \boldsymbol{i} : 1 < i_j < 2^{l_j}, i_j \text{ odd}, 1 \leq j \leq d \right\}\end{aligned}$$

let us define the hierarchical subspace (without boundary) of level corresponding to $\boldsymbol{l}$

$$W_{\boldsymbol{l}} := \operatorname{span} \left\{ \Lambda_{\boldsymbol{l},\boldsymbol{i}} \mid \boldsymbol{i} \in I_{\boldsymbol{l}} \right\}.$$

Similar as before we have a decomposition

$$V = \bigoplus_{j=1}^{\infty} W_n,$$

along with an approximation

$$V \approx V_n = \bigoplus_{j=1}^{n} W_j, \tag{5.9}$$

for which we can formulate the following definition.

Definition 5.9. *The hierarchical surplus in d dimensions is a tensor product of one-dimensional surplusses*

$$\omega_{\boldsymbol{l},\boldsymbol{i}} := \prod_{j=1}^{d} \omega_{l_j,i_j}.$$

Interpolation in dimension $d > 1$ of level $n \geq 1$ using $u \in V_n$ can be written in the form

$$f(x) \approx \Upsilon_n(x) := \sum_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{l}} \sum_{\boldsymbol{i} \in I_{\boldsymbol{j}}} \omega_{\boldsymbol{l},\boldsymbol{i}} \Lambda_{\boldsymbol{l},\boldsymbol{i}}(x). \tag{5.10}$$

Integration in dimension $d > 1$ of level $n \geq 1$ using $u \in V_n$ can be written as

$$\int_{\mathcal{P}} f(x)\,\mathrm{d}x \approx \Phi_n(f) := \sum_{\boldsymbol{1} \leq \boldsymbol{j} \leq \boldsymbol{l}} \sum_{\boldsymbol{i} \in I_{\boldsymbol{j}}} \omega_{\boldsymbol{l},\boldsymbol{i}} \int_{\mathcal{P}} \Lambda_{\boldsymbol{l},\boldsymbol{i}}(x)\,\mathrm{d}x. \tag{5.11}$$

Analogously to (5.8) we have for $f \in C^2$ an estimate

$$|\omega_{\boldsymbol{l},\boldsymbol{i}}| \leq 2^{-d} \cdot 2^{-2|\boldsymbol{l}|_1} \cdot \left\| \frac{\partial^{2d} f}{\partial x_1^2 \cdots \partial x_d^2} \right\|_{\infty}, \tag{5.12}$$

hence also in higher dimensions the decay of hierarchical surplusses is of order $\mathcal{O}\left(4^{-|\boldsymbol{l}|_1}\right)$.

5.1.4. Sparse formulation

Until now we have not introduced sparsity to the grid, as decomposition (5.9) corresponds to a full grid (compare Figure 5.3). Our goal is to exclude grid points while maintaining accuracy. If there is a priori knowledge of f available, one may choose a suitable vector, with each component limiting the refinement of a dimension. However, such knowledge is usually not available. As already discussed, higher order levels contribute less to the approximation, therefore it is a reasonable choice to include only level indices $\boldsymbol{l} : |\boldsymbol{l}|_1 \leq d - 1 + n$ for a suitable maximal level n. This can be formulated for grids without boundary as

$$V_n = \bigoplus_{d \leq |\boldsymbol{l}|_1 \leq d-1+n} W_{\boldsymbol{l}}, \tag{5.13}$$

An example of a sparse grid without boundary in dimension 2 of level 4 is shown in Figure 5.8.

Sparse grids reduce the number of grid points significantly from $\mathcal{O}\left(h_n^{-d}\right)$ (full grid) to $\mathcal{O}\left(h_n^{-1} \cdot (\log h_n^{-1})^{d-1}\right)$, while the asymptotic accuracy detoriates only slightly from full grid $\mathcal{O}\left(h_n^2\right)$ to $\mathcal{O}\left(h_n^2 \cdot (\log h_n^{-1})^{d-1}\right)$.

A more efficient approach selects the refinement of dimensions at run time, this is called (dimension)-adaptivity and will be covered in Chapter 5.3.

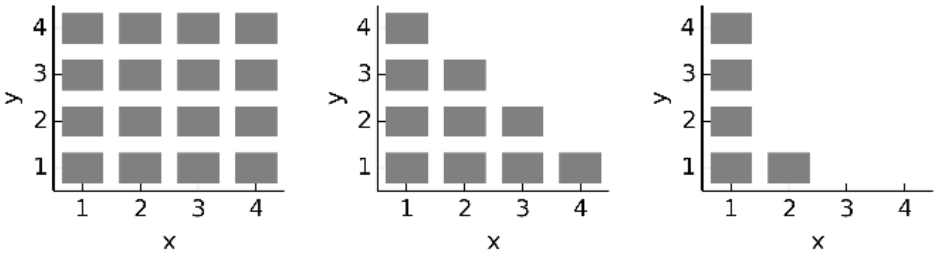


Figure 5.3.: Refinement strategies in two dimension. Left: decomposition (5.9) which corresponds to a full grid. Middle: sparse decomposition V_4 in the form of (5.13). Right: A priori chosen or adaptive refinement for a function which is quasi-linear in x and more stiff in y.

5.1.5. Boundary treatment

So far we considered only functions vanishing on the boundary. For functions with non-homogeneous Dirichlet boundary conditions or Neumann/Robin boundary conditions we have to modify the construction. In order to increase accuracy near the boundary, grid points on the boundary have to be constructed. For that, we include the boundary indices $0, 2^l$ in the previous definitions.

Definition 5.10. *One-dimensional grids of level $l \geq 0$ are defined by*

$$G_l^B := \left\{ x_{l,i} := jh_l \mid 0 \leq j \leq 2^l \right\},$$

and in dimension $d \geq 1$ by

$$G_{\boldsymbol{l}}^B := \bigtimes_{j=1}^{d} G_{l_j}^B.$$

The piecewise linear space in d dimensions is given by

$$V_{\boldsymbol{l}}^B := \operatorname{span} \left\{ \Lambda_{\boldsymbol{l},\boldsymbol{i}} \mid 0 \leq i_j \leq 2^{l_j}, 1 \leq j \leq d \right\}.$$

The index sets are defined as

$$\begin{aligned} I_1^B &:= \left\{ i \in \mathbb{N} \mid 0 < i < 2^l, i \text{ odd} \right\} \cup \{0,1\}, \\ I_l^B &:= \left\{ i \in \mathbb{N} \mid 0 < i < 2^l, i \text{ odd} \right\}, \quad l \leq 2, \\ I_{\boldsymbol{l}}^B &= I_{l_1}^B \times \cdots \times I_{l_d}^B \end{aligned}$$

with newly introduced one-dimensional basis functions $\Lambda_{0,0}(x) := 1 - x$ corresponding to $x_{0,0}$ and $\Lambda_{0,1}(x) := x$ corresponding to $x_{0,1}$. The hierarchical subspaces are

$$W_l^B := \operatorname{span} \left\{ \Lambda_{l,i} \mid i \in I_{\boldsymbol{l}}^B \right\}$$

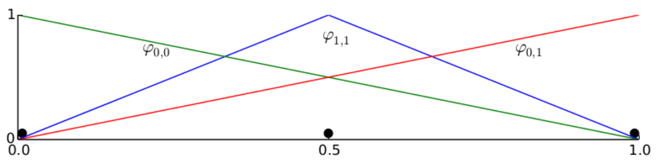


Figure 5.4.: Boundary modification on level 1: two new grid points and basis function.

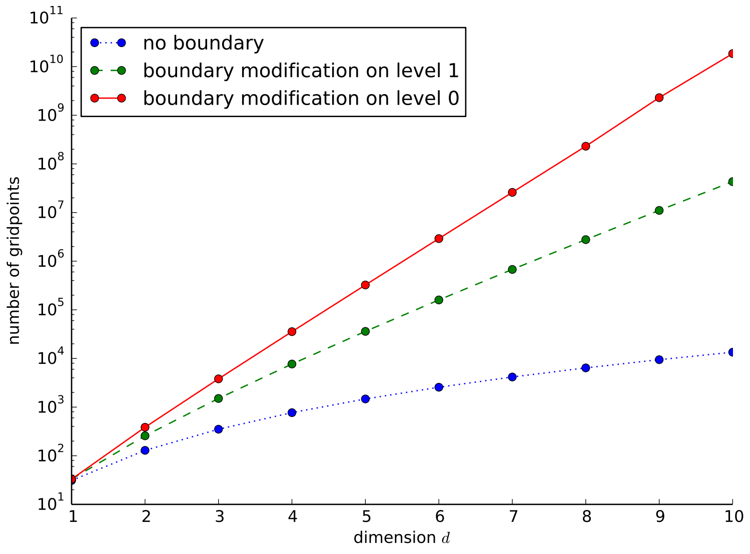


Figure 5.5.: Number of grid points for level $n = 5$ and dimension $d = 1, \ldots, 10$.

such that we have a decomposition

$$V_n^B = \bigoplus_{0 \le |\boldsymbol{l}|_1 \le d-1+n} W_{\boldsymbol{l}}^B, \tag{5.14}$$

compare (5.13).

In Figure 5.5 the number of grid points for this two possible boundary construction are compared with the number of grid points without boundary. Note that the majority of grid points are located on the boundary. The reason for locating the boundary points on level 1 rather then on a newly introduced level 0 comes from the fact that this would lead to additional subspaces in the decomposition (5.14). We see that the construction with level 0 possesses far to many grid points to be applicable even in medium dimensional settings. Still the vast number of newly introduced boundary points is problematic, even with the boundary points located on level 1. In settings where high accuracy close to the boundary is not required, or the function f admits extrapolation towards the boundary, it might not be necessary to introduce boundary points at all. A modified hat basis

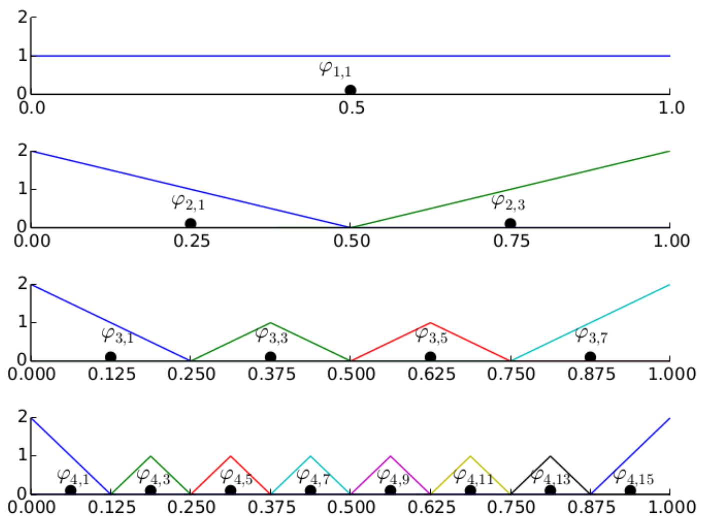


Figure 5.6.: Modified hat basis functions (5.15) on one-dimensional grids without boundary. Shown are grids corresponding to levels $1, \dots, 4$.

function folded up towards the boundary which is given by

$$\Lambda_{l,i}(x) = \begin{cases} 1 & \text{if } l = 1 \wedge i = 1, \\ \begin{cases} 2 - 2^l \cdot x & \text{if } x \in [0, h_l] \\ 0 & \text{else} \end{cases} & \text{if } l > 1 \wedge i = 1, \\ \begin{cases} 2^l \cdot x + 1 - i & \text{if } x \in [1 - h_l, 1] \\ 0 & \text{else} \end{cases} & \text{if } l > 1 \wedge i = 2^l - 1, \\ \Lambda\left(2^l \cdot x - i\right) & \text{else.} \end{cases} \tag{5.15}$$

can be used to extrapolate towards the boundary. We will use only this modified basis functions rather then the standard hat basis, it is shown in Figure 5.6 (compare Figure 5.1).

5.2. Implementation

5.2.1. Grid Structure

Implementation of sparse grids requires efficient data handling since the number of grid points is likely to be high. A sparse grid can be represented in a d-dimensional tree-like structure, with grid points of level n on tree height n. A grid point possesses up to $2d$ pointers to child nodes (on level $n + 1$) and up to d pointers to father nodes (on level $n - 1$), so the actual data structure is not a classical tree. A variety of methods dealing with such tree-like structures has been proposed, see [Feu10, Chapter. 2.2.3] and references therein. As the number of pointers is growing quick, managing those pointers can become algorithmically involving.

Therefore we use another approach. Since a grid point is completely described by its multiindices $\boldsymbol{l}, \boldsymbol{i}$ we will use those to perform grid operations like hierarchization, interpolation and quadrature. The drawback of using multiindices is that each time we want to access a grid point we have to perform $2d$ lookup operations in the multiindices $\boldsymbol{l}, \boldsymbol{i}$. To overcome this drawback we employ *hashing*, a mapping from an arbitrary sized domain to a codomain of fixed size. Common hash functions suffer from collisions, that is non-injectivity of the hash map, and non-surjectivity which introduces storage overhead in the codomain size. A well behaved hash function would be an injective and surjective function map from the set of grid points $\{(\boldsymbol{l}, \boldsymbol{i})\}_{|\boldsymbol{l}|_1 \leq d-1+n}$ onto the naturals $\{0, \ldots, |G_n|\}$. As it enumerates the grid points uniquely it is therefore called *unique hashing*.

5.2.2. Unique hashing

The unique hashing hashing of $(\boldsymbol{l}, \boldsymbol{i})$ was proposed in [Feu05]. We are going to use a variant of this subsequently proposed in [Feu10], it relies on hashing of the multiindices separately.

We start with the hashing of the level index $\boldsymbol{l}$ and introduce $n(\boldsymbol{l}) = |\boldsymbol{l}|_1 - d + 1$. We denote the number of points in dimension d with level n by

$$P^{n,d} := \left| \left\{ \boldsymbol{l} \in \mathbb{N}^d : n(\boldsymbol{l}) \right\} \leq n \right|$$

The recurrence formula

$$P^{n,d} = \sum_{m=1}^{n} P^{m,d-1} \tag{5.16}$$

with base cases $P^{n,0} = 0, P^{n,1} = n$ can be seen by induction with respect to d.

In the following consider the level indices forming a d-dimensional simplex. In order to align with our notation, we assume that the simplex has coordinates

$$(1, \ldots, 1), (n, 1, \ldots, 1), (1, n, 1, \ldots, 1), \ldots, (1, \ldots, 1, n),$$

rather than the usual definition which locates the simplex at the origin $(0, \ldots, 0)$. Levels $\boldsymbol{k}$ of level n are located on sections of the simplex, i.e., on a $d-1$ dimensional hyperplane intersected with the simplex

$$K_n^d := \left\{ \boldsymbol{k} \in \mathbb{N}^d : n(\boldsymbol{k}) = n \right\}. \tag{5.17}$$

Note that this is a simplex with one dimension less (compare Figure 5.7).

Consider an arbitrary but fixed $\boldsymbol{l} \in K_n^d$. The task is to enumerate the multi-index by a function $r_n(\boldsymbol{l}) =: r_n^d(\boldsymbol{l})$. Levels $\boldsymbol{l}$ on K_n^d can be described by

$$K_n^d = \left\{ \boldsymbol{l} = \begin{pmatrix} n+d-1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \sum_{m=2}^{d} \tilde{l}_m \begin{pmatrix} -1 \\ e_{m-1} \end{pmatrix} \mid \forall m = 2, \ldots, d : \tilde{l}_m > 0 \text{ such that } \boldsymbol{l} \in \mathbb{N}^d \right\}$$

where e_m denotes the m-th unit vector in $\mathbb{N}^{d-1}$. Hence we have a lower-dimensional representation of $\boldsymbol{l}$ in the form of $\tilde{\boldsymbol{l}} = (\tilde{l}_2, \dots, \tilde{l}_d) \in \mathbb{N}^{d-1}$, which is again a simplex. The idea is to proceed the enumeration process in this lower-dimensional simplex. For that we have to count the levels $\tilde{\boldsymbol{k}}$ which possess smaller $n(\tilde{\boldsymbol{k}})$ than $n(\tilde{\boldsymbol{l}})$, with $n(\tilde{\boldsymbol{l}}) = n(\boldsymbol{l}) - (l_1 - 1)$. Those levels are drawn blue in the Figure 5.7, the number of points is given by

$$P^{n(\tilde{\boldsymbol{l}})-1,d-1}.$$

Additionally we have to count the points with $n(\tilde{\boldsymbol{k}}) = n(\tilde{\boldsymbol{l}})$ which are enumerated before $\tilde{\boldsymbol{l}}$ (shown in red), those are given by

$$r^{d-1}_{n(\tilde{\boldsymbol{l}})} n(\tilde{\boldsymbol{l}}).$$

Hence we have a recursion formula

$$r^d_n(\boldsymbol{l}) = P^{n(\tilde{\boldsymbol{l}})-1,d-1} + r^{d-1}_{n(\tilde{\boldsymbol{l}})} n(\tilde{\boldsymbol{l}}) \tag{5.18}$$

with base case $r^1_n(\boldsymbol{l}) = 0$. By setting

$$n_m(\boldsymbol{l}) := \sum_{k=m}^{d} (l_k - 1) + 1 = n(\boldsymbol{l}) - \sum_{k<m} (l_k - 1)$$

we derive from (5.18) the hash function

$$\tilde{r}(\boldsymbol{l}) = \sum_{m=2}^{d} P^{n_m(\boldsymbol{l})-1,d-(m-1)} \tag{5.19}$$

with base cases $P^{n,1} = n, P^{n,0} = 0$.

It remains to specify the hashing of $\boldsymbol{i}$. The hash depends on the level index as well and is given by

$$r_{\boldsymbol{l}}(\boldsymbol{i}) := \sum_{j=1}^{d} \left\lfloor \frac{i_j}{2} \right\rfloor \prod_{k<j} 2^{l_k - 1}.$$

The values (5.16) can be precomputed by Algorithm 5.9 in time $\mathcal{O}\left(dn^2\right)$ with memory consumption $\mathcal{O}\left(dn\right)$ for maximal level n. The computation of (5.19) is described in Algorithm 5.10.

5.2.3. Grid storage

Using the hashing developed above, we set up our grid data structure as follows. Assume that level index $\boldsymbol{l}$ corresponds to level n, that is $\boldsymbol{l} : |\boldsymbol{l}|_1 = n - d + 1$. Assume further that there are g multiindices $\boldsymbol{i}_1, \dots, \boldsymbol{i}_g$ corresponding to level $\boldsymbol{l}$. Each time a level index is added to the grid[3] we compute $\boldsymbol{i}_1, \dots, \boldsymbol{i}_g$, so we have g new grid points

[3] in a non-adaptive grid, for level n the level indices $\boldsymbol{l} : |\boldsymbol{l}|_1 = n - d + 1$ are added simultaneously; in a dimension-adaptive grid only one level index is added at a time

Algorithm 5.9 Setup phase for Algorithm 5.10

```
$P = \text{Array}(\mathbb{N}, n \times d)$
for $k = 0, \dots, n$ do
    $P^{k,1} = k$
end for
for $m = 2, \dots, d$ do
    for $k = 0, \dots, n$ do
        $P^{k,m} = \sum_{q=1}^{k} P^{q,m-1}$
    end for
end for
return $P$
```

Algorithm 5.10 Unique level hashing of $\boldsymbol{l} = (l_1, \dots, l_d)$

```
$\hat{d} = 0$
$n = 1$
$\tilde{r} = 0$
for $m = d, \dots, 1$ do
    $\hat{d} = \hat{d} + 1$
    $n = n + l_m - 1$
    $\tilde{r} = \tilde{r} + P^{n-1,\hat{d}}$
end for
return $\tilde{r}$
```

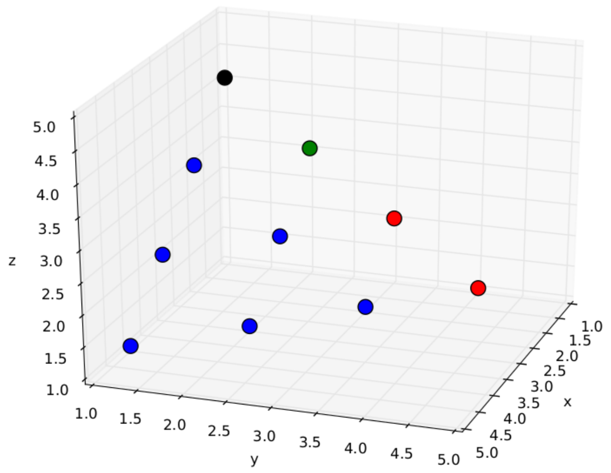


Figure 5.7.: Shown as colored dots are the level indices in K_4^3, so all those points have $n(\boldsymbol{l}) = 4$. We consider the green point $\boldsymbol{g} = (1, 2, 3)$, it holds $n(\tilde{\boldsymbol{g}}) = 4$. Marked as blue are points $\boldsymbol{k} \in K_4^3 : n(\tilde{\boldsymbol{k}}) < 4$, red points correspond to points with $n(\tilde{\boldsymbol{k}}) = 4$ which are counted before $\boldsymbol{g}$. The black point $(1, 1, 4)$ is the only level to be counted after $\boldsymbol{g}$.

$(\boldsymbol{l}, \boldsymbol{i}_1), \dots, (\boldsymbol{l}, \boldsymbol{i}_g)$. For each of those grid points we compute its coordinates, its evaluation and its hierarchical surplus. Now we compute the levelhash $\tilde{r}(\boldsymbol{l})$ using Algorithm 5.10. A pointer with index $\tilde{r}(\boldsymbol{l})$ is created, which points to an array holding the coordinates and surplusses in a row-wise form such that the data of grid point $(\boldsymbol{l}, \boldsymbol{i}_k)$ is stored in row $r_{\boldsymbol{l}}(\boldsymbol{i}_k), k = 1, \dots, g$.

Thus we can store and access a grid point in $\mathcal{O}(d)$ access operations, independent of the number of grid points N.

Remark 5.11. *Note that that the matrices $\{A(p) : p \in \mathcal{P}\}$ from (4.10) possess the same sparsity structure for a fixed triangulation $\mathcal{T}$. Hence, when performing the left-hand side quadrature in (4.10) we can economize the degrees of freedom which have to be stored in the grid by omitting the zero entries of the matrices. To this end we store the row and column pointers of the Compressed Sparse Row (CSR) format [Saa11, Chapter 2.2] of such a matrix, which is provided when using the `uBLAS`[4] linear algebra backend. Now only non-zero entries of the matrix have to be processed by the quadrature. The storage reduction is significant, and the matrix can be reassembled by using the stored row and column pointers.*

Another possibility is to store the whole sparse matrices, and use the fast `axpy` operation (compare Definition B.2) to perform additions and subtractions of matrices while hierarchizing and assembling the left-hand side matrix.

5.2.4. Hierarchization

In equation (5.7) the hierarchical surplusses are computed by using the interpolation of lower levels. If we have N grid points we need time $\mathcal{O}\left(N^2\right)$ to compute the surplusses of all grid points. This quadratic behaviour is a major bottleneck as the number of grid points grows. Note that the majority of grid points will not contribute to a certain surplus because of the local support of the basis functions. Thus, it would be beneficial to include just the points in the interpolation which are necessary.

Consider a d-dimensional grid point $x := (\boldsymbol{l}, \boldsymbol{i})$ of level n, with $\boldsymbol{l} := (l_1, \dots, l_d)$ and $\boldsymbol{i} := (i_1, \dots, i_d)$. From (5.7) it follows that contributing grid points are located on levels $1, \dots, n-1$. As described earlier, in a tree-like structure each grid point has up to d father nodes, that is maximal one in each dimension. The idea is to find all father nodes of x on level $n-1$, and recursively find their father nodes until we reach the ancestor of all points – that is $(\mathbf{1}, \mathbf{1})$. A father node may occur multiple times while following the paths upwards (compare Figure 5.8), we need to keep track of already identified fathers. We use again unique hashing to mark those fathers for each hierarchization $\omega_{\boldsymbol{l},\boldsymbol{i}}$ in the same manner as we store the grid.

The (possibly existing) father node of $x_{\{}\boldsymbol{l}, \boldsymbol{i}\}$ in dimension $k \in 1, \dots, d$ can be found easily. If $l_k = 1$, there is no father node in this direction, if $l_k > 1$ we subtract a level in in this component, so this father of x has level index

$$(l_1, \dots, l_{k-1}, l_k - 1, l_{k+1}, \dots, l_d)\,. \tag{5.20}$$

[4]as part of the C++ libraries `boost`

We need to update the index as well: if $i_k = 1$, the index is unchanged. If $i_k > 1$, we decide whether $\frac{i_k - 1}{2}$ or $\frac{i_k + 1}{2}$ is odd, and use the respective formula to get the father index

$$\left(i_1, \ldots, i_{k-1}, \frac{i_k \pm 1}{2}, i_{k+1} \ldots, i_d\right).$$

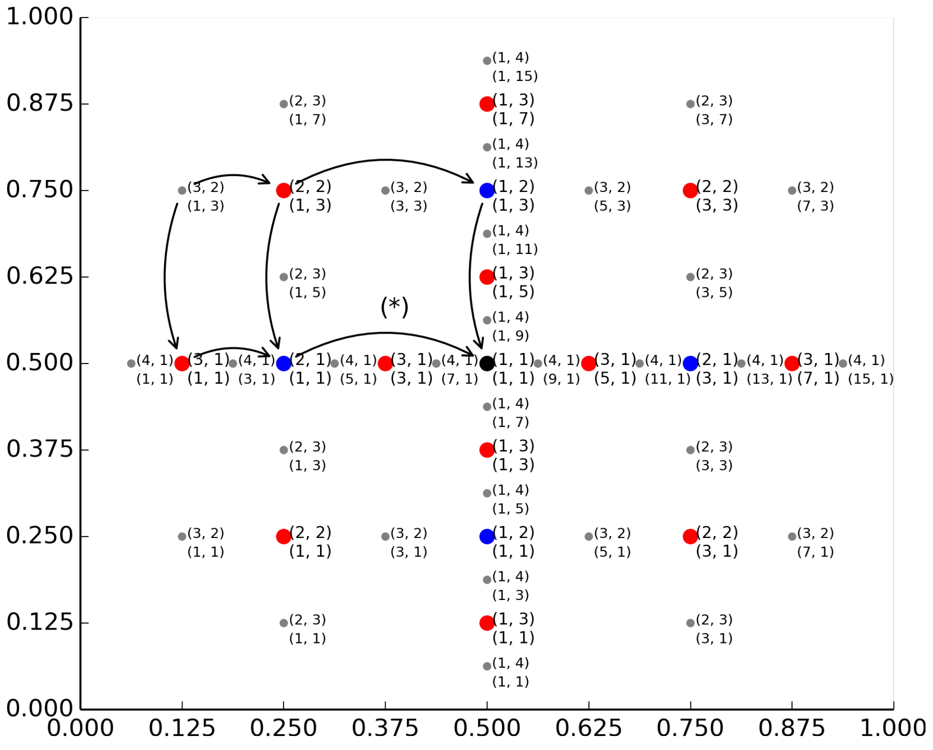


Figure 5.8.: The classical 2-dimensional grid of level 4. The black point corresponds to level 1, blue points to level 2 and red to level 3. The small gray points correspond to level 4, those will not contribute to the hierarchization of $((3,2),(1,3))$. The arrows represent the identification of a father in the respective dimension. The path marked with $(*)$ is only followed once, because we identify $((2,1),(1,1))$ as a known father as soon as it is met first.

5.3. Dimension-adaptive sparse grids

The sparse grids introduced so far possess significantly fewer grid points than a full grid. However, as noted in the beginning of this chapter, they just delay the curse of dimensionality. To see this, consider a function where only a minority of stochastic dimensions has crucial impact on the function value. To increase the accuracy of the approximation, *all* dimensions must be refined, while only the minority of dimensions profits significantly from this refinement. Since refinement corresponds to solving problems of type (3.1), this overhead is significant.

Dimension adaptive grids aim to refine only where it is beneficial for error reduction. This chapter follows [GG03] which introduces such an adaptive method. Rather than using their proposed array based data structure we use the hash structure developed above.

For further reference, there are even more sophisticated methods which are not only dimension-adaptive, but can also refine dimensions locally at areas where the function is stiff [JR11], [MZ09]. Such techniques will not be covered here.

5.3.1. Preliminaries

The idea of the algorithm is to heuristically construct a set of level indices $\boldsymbol{l} \in \mathcal{G}$ such that the integral

$$\int_{\mathcal{P}} f(x)\,\mathrm{d}x \approx \sum_{\boldsymbol{l}\in\mathcal{G}} \sum_{\boldsymbol{i}\in I_{\boldsymbol{l}}} \omega_{\boldsymbol{l},\boldsymbol{i}} \int_{\mathcal{P}} \Lambda_{\boldsymbol{l},\boldsymbol{i}}(x)\,\mathrm{d}x,$$

is a good approximation of (5.11) while using less work.

Note that the construction of the set $\mathcal{G}$ is governed by the progress of the algorithm, e.g., require points of $\boldsymbol{l} = (2,2)$ the existence of the points corresponding to $(1,2), (2,1), (1,1)$, thus it is not admissible to make a refinement selection $(1,1) \to (2,1) \to (2,2)$.

Definition 5.12. *We define the forward neighbourhood of an index $\boldsymbol{l}$ as*

$$\{\boldsymbol{l} + \boldsymbol{e}_j \mid j = 1, \dots, d\},$$

and analog its backward neighborhood

$$\{\boldsymbol{l} - \boldsymbol{e}_j \mid j = 1, \dots, d\}$$

for unit vectors $\boldsymbol{e}_j$.

The index set $\mathcal{G}$ to be constructed is partitioned into two disjoint sets, an old index set $\mathcal{O}$ and an active index set $\mathcal{A}$. We call members of this sets old indices and active indices, respectively. A multi-index $\boldsymbol{l}$ is admissible, if

$$\forall j = 1 \dots, d: \quad \boldsymbol{l} - \boldsymbol{e}_j \in \mathcal{O}. \tag{5.21}$$

We denote the differential integral of a level index $\boldsymbol{l}$ by

$$\Delta_{\boldsymbol{l}} f := \sum_{\boldsymbol{i}\in I_{\boldsymbol{l}}} \omega_{\boldsymbol{l},\boldsymbol{i}} \int_{\mathcal{P}} \Lambda_{\boldsymbol{l},\boldsymbol{i}}(x)\,\mathrm{d}x.$$

5.3.2. Method

We initialize the algorithm with $\mathcal{O} = \emptyset$ and $\mathcal{A} = \{\boldsymbol{l} := \mathbf{1}\}$. Each step of the algorithm removes the index with the largest error from active set $\mathcal{A}$, transfer this index to the old set $\mathcal{O}$ and computes all its admissible forward neighbors.

A possible choice to measure the error reduction of a level $\boldsymbol{l}$ is the error indicator

$$g_{\boldsymbol{l}} := \left| \frac{\Delta_{\boldsymbol{l}} f}{\Delta_{\text{Init}} f} \right| , \tag{5.22}$$

where we set $\Delta_{\text{Init}} f := \Delta_{\mathbf{1}} f$, which corresponds to a greedy selection of the levels to be refined.

It might happen that a level has small error reduction, but its forward neighbourhood contributes significantly. With a greedy approach, this level will not be refined further as long as there are active indices with higher error indicator.

Another possible choice is a generalized error indicator

$$\tilde{g}_{\boldsymbol{l}} := \max \left\{ \left| \frac{\Delta_{\boldsymbol{l}} f}{\Delta_{\text{Init}} f} \right| , (1-\varrho) \frac{n_{\text{Init}}}{n_{\boldsymbol{l}}} \right\} ,$$

with weighting parameter $\varrho \in [0,1]$, and denoting the involved work for a level $\boldsymbol{l}$ – that is the number of grid point evaluations – by

$$n_{\boldsymbol{l}} := \prod_{j=1}^{d} |I_{l_j}| ,$$

and set $n_{\text{Init}} := n_{\mathbf{1}}$. The choice $\varrho = 1$ corresponds to the greedy approach and $\varrho = 0$ to the classical sparse grid where only involved work matters. The benefit of using the generalized error indicator with values $\varrho \in (0,1)$ is that the algorithm will refine levels if they have small error reduction, but require few work compared to other levels with similar or higher error reduction. The drawback of using the generalized error indicator is that we mix convergence information of the differential integral $\Delta_{\boldsymbol{l}} f$ with the required work.

When dealing with high-dimensional problems, the cardinality of the active index set $\mathcal{A}$ grows fast. Therefore it is inefficient to search the largest error indicator in each step. As we are only interested in the current largest error indicator, we use a *heap data structure* for $\mathcal{A}$ (see Appendix B), which enables us to find the largest error indicator $g_{\boldsymbol{l}}$ efficiently in $\mathcal{O}(1)$.

5.3.3. Modifications for product integrals

In contrast to the Gaussian quadrature in the discrete projection (3.43) we cannot apply Algorithm 3.5 any more, since the hierarchical surplusses are not linear in the sense that

$$\omega_{\boldsymbol{l},\boldsymbol{i}}^{f\cdot\psi_\alpha} \neq \omega_{\boldsymbol{l},\boldsymbol{i}}^{f} \cdot \omega_{\boldsymbol{l},\boldsymbol{i}}^{\psi_\alpha}$$

Thus we have to compute the product integrals separately in the form of

$$u^\alpha \approx \sum_{\boldsymbol{l}\in\mathcal{G}} \sum_{\boldsymbol{i}\in I_{\boldsymbol{l}}} \omega_{\boldsymbol{l},\boldsymbol{i}}^{f\cdot\psi_\alpha} \int_{\mathcal{P}} \Lambda_{\boldsymbol{l},\boldsymbol{i}}(x)\,\mathrm{d}x. \tag{5.23}$$

Note that $\omega_{\mathbf{1},\mathbf{1}}^{f\cdot\psi_\alpha}$ vanishes for all α which have an odd entry, since any odd univariant Legendre polynomial vanishes at 0 and thus also the multivariant polynomial. An immediate consequence of this is that we cannot use the error indicator (5.22) as now $\Delta_{\mathbf{1}}(f\cdot\psi_\alpha) = 0$. Moreover, if α_j is odd for $j \in \{1,\dots,di\}$, and it holds $\boldsymbol{l}_j = 1$ it follows that $\omega_{\boldsymbol{l},\boldsymbol{i}}^{f\cdot\psi_\alpha} = 0$ and thus $\Delta_{\boldsymbol{l}}(f\cdot\psi_\alpha) = 0$ for the same reason as above. If α has many odd entries, the number of such vanishing levels will be vast and thus we cannot initialize the algorithm with $\mathbf{1}$ and refine aimless until we encounter a non-vanishing level. Hence we need to identify the first contributing level, this level is given by

$$\boldsymbol{l} : l_j = \begin{cases} 1 & \alpha_j \text{ even}, \\ 2 & \alpha_j \text{ odd}, \end{cases} \tag{5.24}$$

where we make the assumption that the solution of (3.1) does not vanish which is reasonable for $\dim \mathcal{U}$ large enough.

Consider now again the level indices as d-dimensional simplex. Levels $\boldsymbol{l} : l_j = 1$ correspond to the facet in dimension j, respectively multiple facets if there are multiple such entries. From the father identification (5.20) it follows that once we reach a grid point $(\boldsymbol{l},\boldsymbol{i})$ with $\boldsymbol{l}$ on such a vanishing facet, we can stop this hierarchization path early, since all its fathers will vanish as well. Moreover, we never need to compute the values and hierarchical surplusses of these points, since we know that their values will vanish and hence also their hierarchical surplusses.

Remark 5.13. *We face the situation of vanishing levels again when performing direct tensor approximation updates for linear problems by solving* (4.4)*. Furthermore, note that the solution of* (4.10) *is not possible in each quadrature point p_z, as the equation has no solution whenever $\lambda(p_z) = 0$, since the left-hand side vanishes. The reasoning from above applies here as well.*

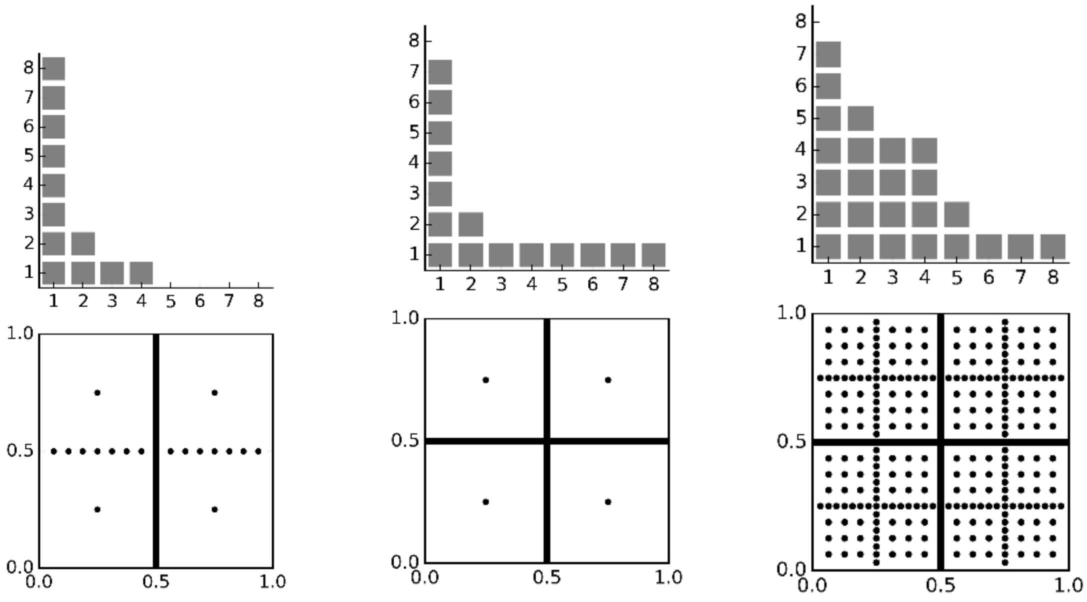
5.3.4. Algorithm

Before giving the algorithm we discuss an algorithmic modification for the discrete projection method which allows us to economize solver calls. This technique is not applicable for the tensor approximation, nevertheless we give the most general algorithm here, as the discussed modifications just become obsolete for the tensor product quadrature. We can apply the algorithm to product integrals as well as integrals of type $\int_{\mathcal{P}} f(x)\,\mathrm{d}x$, since this equals $\int_{\mathcal{P}} f(x)\cdot\psi_{\mathbf{0}}(x)\,\mathrm{d}x$ because the first univariant Legendre polynomial is the constant function 1.

As solving the problem (3.1) is costly, we want to reuse this information rather then recomputing it when encountering the evaluations of grid points in the different product integrals (5.23). To this end we store such evaluations separately; the quadrature of a product integral will perform a lookup if these evaluations were already done by a previous product integral quadrature. If this data is available we multiply with ψ_α and hierarchize, if not we evaluate the grid points and store this data, and then proceed as before. We store this function evaluations in the same manner as we store the grid (compare Chapter 5.2.3). We denote this hash structure by $\mathcal{B}$ and by $\mathcal{B}_{\boldsymbol{l}}$ we denote the function evaluations of a level $\boldsymbol{l}$, moreover it denotes $\mathcal{B}_{\boldsymbol{l},\boldsymbol{i}}$ a evaluation of a single grid point.

We denote by $D_{\boldsymbol{l}}$ the data of a level – that is an array holding $(\boldsymbol{l}, \boldsymbol{i}, x_{\boldsymbol{l},\boldsymbol{i}}, \omega_{\boldsymbol{l},\boldsymbol{i}})$ such that each row corresponds to a grid point. By $\mathcal{H}$ we denote the hash structure of the grid, that is the collection of all level data $D_{\boldsymbol{l}}$.

For sake of runtime we can parallelize the evaluation and hierarchization of grid points on different (local) CPU cores. Since an admissible forward neighbor may require only few work, while other admissible forward neighbors require more work, it is more efficient to gather the necessary points of all admissible forward neighbors in a list J and work on all of them in a parallelized manner.



(a) $f(x,y) = \frac{1}{10}e^{-x^2} + 10e^{-y^2}$ (b) $f(x,y) = e^x + e^y$ (c) $f(x,y) = e^x + e^y + e^{xy}$

Figure 5.9.: Two-dimensional adaptive refinements (top) and corresponding grids (bottom).

Algorithm 5.11 Dimension-adaptive grid generation with input $\alpha, \mathcal{B}$

```
$\boldsymbol{l} = (5.24)$
$\mathcal{A} = \{\boldsymbol{l}\}$
$D_{\boldsymbol{l}}, \mathcal{B}_{\boldsymbol{l}} = \text{computeLevel}(\boldsymbol{l}, \mathcal{B})$
$\Delta_{\text{Init}}(f \cdot \psi_\alpha) = \Delta_{\boldsymbol{l}}(f \cdot \psi_\alpha)$
Store $D_{\boldsymbol{l}}$ to $\mathcal{H}$
Store $\mathcal{B}_{\boldsymbol{l}}$ in $\mathcal{B}$ if it was computed newly
$\mathcal{O} = \emptyset$
$\mathcal{A} = \{(\boldsymbol{l}, g_{\boldsymbol{l}})\}$
$\eta = g_{\boldsymbol{l}}$
**while** $\eta < tol$ **do**                                        ▷ One step
    **if** $\mathcal{A} = \emptyset$ **then**                        ▷ No convergence within maximal level
        **break**
    **end if**
    Select $\boldsymbol{l}$ from $\mathcal{A}$ with largest $g_{\boldsymbol{l}}$
    $\mathcal{A} = \mathcal{A} \setminus \{(\boldsymbol{l}, g_{\boldsymbol{l}})\}$
    $\mathcal{O} = \mathcal{O} \cup \{\boldsymbol{l}\}$
    $\eta = \eta - g_{\boldsymbol{l}}$
    $J = \emptyset$
    **for** $j = 0, \dots, d$ **do**
        $\boldsymbol{k} = \boldsymbol{l} + \boldsymbol{e}_j$                              ▷ Find forward neighbors
        **if** $\forall q = 1, \dots, d : \boldsymbol{k} - \boldsymbol{e}_q \in \mathcal{O}$ **then**          ▷ Check for admissibility
            compute grid point locations $(\boldsymbol{l}, \boldsymbol{i}, x_{\boldsymbol{l},\boldsymbol{i}})$ corresponding to $\boldsymbol{k}$
            append them to $J$
        **end if**
        $\{D_{\boldsymbol{k}}\}_{\boldsymbol{k} \text{ admissible}}, \{\mathcal{B}_{\boldsymbol{k}}\}_{\boldsymbol{k} \text{ admissible}} = \texttt{computeLevel}(J, \mathcal{B})$     ▷ worker process
        **for** $\boldsymbol{k}$ admissible **do**
            Store $D_{\boldsymbol{k}}$ to $\mathcal{H}$
            Store $\mathcal{B}_{\boldsymbol{k}}$ in $\mathcal{B}$ if it was computed newly
            $\mathcal{A} = \mathcal{A} \cup \{(\boldsymbol{k}, g_{\boldsymbol{k}})\}$
            $\eta = \eta + g_{\boldsymbol{k}}$
        **end for**
    **end for**
**end while**
**return** $\mathcal{H}, \mathcal{B}$
```

Algorithm 5.12 `computeLevel`(J, $\mathcal{B}$)

```
for all grid points in $J$ do                                          ▷ parallelized
    if grid point $\in \mathcal{B}$ then                          ▷ lookup and hierarchize
        lookup $\mathcal{B}_{\boldsymbol{l},\boldsymbol{i}}$
        $\omega_{\boldsymbol{l},\boldsymbol{i}} =$ hierarchize $\mathcal{B}_{\boldsymbol{l},\boldsymbol{i}} \cdot \psi_\alpha(x_{\boldsymbol{l},\boldsymbol{i}})$
    else                                                      ▷ evaluate and hierarchize
        $\mathcal{B}_{\boldsymbol{l},\boldsymbol{i}} = f(x_{\boldsymbol{l},\boldsymbol{i}})$
        $\omega_{\boldsymbol{l},\boldsymbol{i}} =$ hierarchize $\mathcal{B}_{\boldsymbol{l},\boldsymbol{i}} \cdot \psi_\alpha(x_{\boldsymbol{l},\boldsymbol{i}})$
    end if
end for
return $D_{\boldsymbol{l}}, \mathcal{B}_{\boldsymbol{l}}$
```

6. Numerical Results

As a test example we use the Poisson model problem (2.6), (2.7) with homogeneous boundary conditions and an artificially generated random field.

The implementation is done in Python[1], a popular object-oriented language. The FEM problem is solved by using `FEniCS` [LMW12], a powerful framework for solving differential equations with Python and C++ interfaces. The model problem is provided by `ALEA`, which is also implemented in Python and uses `FeniCS` as backend.

6.1. Discrete Projection

As already said, due to the integration overhead we focus on the discrete projection method (Chapter 3.3.3) where the integrals have to be computed only once, not in each iteration cycle.

For the tests we use 60^2 degrees of freedom for the spatial mesh triangulation and perform tests in $3, 5, 8$ stochastic dimensions. We use the index sets from Def. 3.2 each of them with ansatz degree $0, \dots, 4$. We choose quadrature tolerances $10^{-2}, 10^{-3}, 10^{-4}$, and for the collocation method (sparse grid interpolation) we also compute with tolerance 10^{-5}. The grid generation tolerance is not an absolute error measure, but rather measures the error reduction starting with the initial differential integral. Different product integrals have initial values which may differ in the orders of magnitude. Thus the quadrature accuracy of product integrals is not directly comparable. To prevent the quadrature from getting stuck while reducing the error of a low initial differential integral, we limit the quadrature with 60 iterations. A more elaborate approach would be to implement a stagnation criterion based on (5.12).

We plot the RMS error against the amount of work measured in solver calls. The results are shown in Figure 6.1, 6.2, 6.3. As reference we plot the RMS error of the collocation method as well. We neglect the inaccurate ansatz degrees $0, 1$, and compute the total degree ansatz set only in dimension 3 due to its size. The similar amount of work between quadrature tolerances results from the limited quadrature iterations. For the RMS error we use 5000 Monte Carlo points.

The drawback of both methods from Chapter 3 has already been indicated in Figure 3.1 – the vast number of ansatz functions when using a predefined index set $\mathcal{I}$. Due to the index set cardinality we expect total degree index set to be most accurate, followed by sparse Smolyak and hyperbolic cross. The tests verify this behaviour, however the accuracy is paid with notably more work in terms of solver calls.

[1]`http://www.python.org`

We see that the discrete projection method is capable of producing higher accuracy results than the collocation method, for example in dimension 8 (Figure 6.3), the ansatz sets $\mathcal{I}_{\mathrm{HC}}$ and $\mathcal{I}_{\mathrm{SS}}$ with quadrature tolerance 10^{-3} and ansatz degrees $3, 4$ are more accurate than the collocation method with the same quadrature tolerance. However, the amount of work which has to be invested is magnitudes larger. It is doubtful if we can ever achieve such results within a comparable amount on work invested. For dimensions > 10, the ansatz degree has to be chosen very low in order to compute the results in reasonable time, yet this limits the accuracy of the approximation. All together, the method does not seem capable of tackling medium or high-dimensional problem within a reasonable amount of work.

6.2. Tensor approximation

Here, we focus on the tensor approximation with adaptive updates using the `ASGFEM` estimators (Chapter 4.2).

We use initial dimension 1 and a triangular mesh $\mathcal{T}$ consisting of $2 \cdot 30^2$ mesh cells and conforming piecewise cubic finite elements P_3 (compare [Joh13, Chapter 5.2]), resulting in 8281 degrees of freedom. We use 300 Monte Carlo points for the RMS error computation.

In Figure 6.4 we plot the RMS error against the tensor approximation rank r with deactivated mesh refinement. We plot two tests with initial dimension 1, one without predefined initial multiindices α besides (0), and one with predefined initial multiindices $(0), \ldots, (9)$ (compare Chapter 4.2.4). We see in both cases stagnation of the error reduction rather than convergence. The reason for this behaviour could not be found in time, but the test with predefined index set suggests that there is a bug when choosing the next multiindices to be added, since the better error reduction seems to result from the higher multiindices in the first dimension. The tensor approximation without predefined multiindices does not reach (9) until the test was stopped. Moreover there remains a problem in the mesh refinement step resulting in an error increase for refined triangulations. Also this problem could not be found in time.

For clarity of Algorithm 4.8 we simplified the setup step such that only $\mathbf{0}$ is the initial multiindex, but for reasons of implementation the `ASGFEM` routines assume nonempty $\operatorname{supp}(\mathcal{I})$, so in fact the initial multiindices must contain also $(1, 0, \ldots)$. This is the reason for the larger error in the first RMS test with predefined multiindices, since for this test we carry out the RMS computation after each new multiindex, rather than after each refinement step which may add more than one multiindex. We see that the RMS errors at rank 2 are quite the same as expected, the minor difference results from choosing different sets of Monte Carlo points.

First we can rule out that the problem of choosing multiindices lies in the `ASGFEM` routines, as they are successfully used in [EGSZ15]. Moreover, we can rule out that the reason for both problems lie in inaccurate quadrature (we chose tolerance 10^{-4}) or that the triangulation $\mathcal{T}$ is too coarse, as we also tested with $2 \cdot 40^2$ mesh cells. One possible reason for the stagnation is that the used test field (of type `monomials` [EZ]) causes the problems, which has not been tested extensively as Dr. Eigel noted. It was not possible to use to original proposed test field `EF-square-cos`, as this test field possesses symmetries which result in vanishing of certain initial differential integrals (5.24) in the right hand side of (4.10). For example, this happens at quadrature level 2 for identical function values $b(p_i) - A(u_r(p_i); p_i)$, $i = 1, 2$, which vanish due to the antisymmetric nature of the second Legendre polynomial. Either a more elaborate method to find the initial non-vanishing differential integral has to be employed, or the test field has to be modified. Still the modification of the test field would not be satisfying, since the method is expected to work for arbitrary (linear) problems. In the following chapter we discuss some possible improvements for the quadrature which overcome this problem.

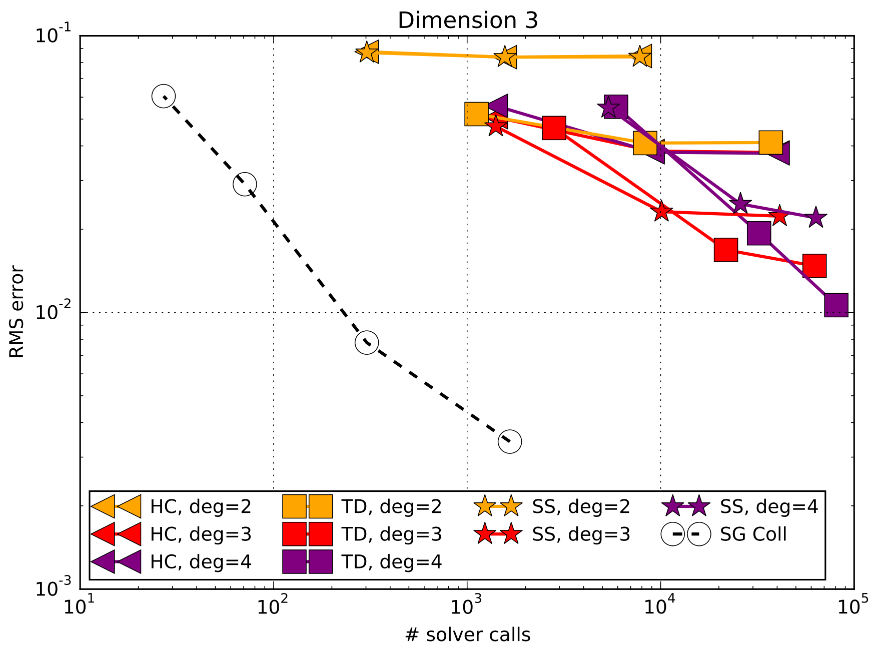


Figure 6.1.: Discrete projection for 3 stochastic dimensions

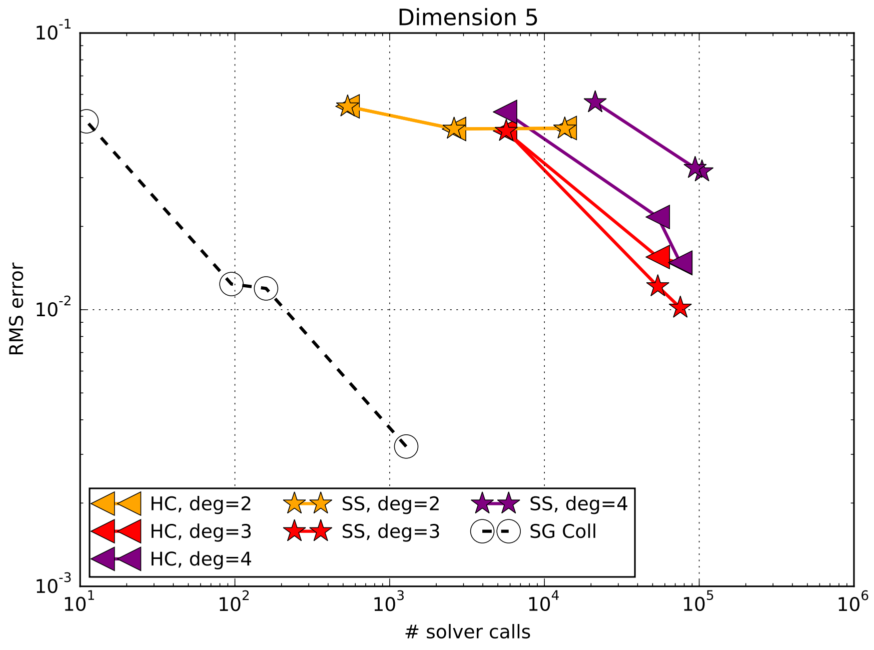


Figure 6.2.: Discrete projection for 5 stochastic dimensions

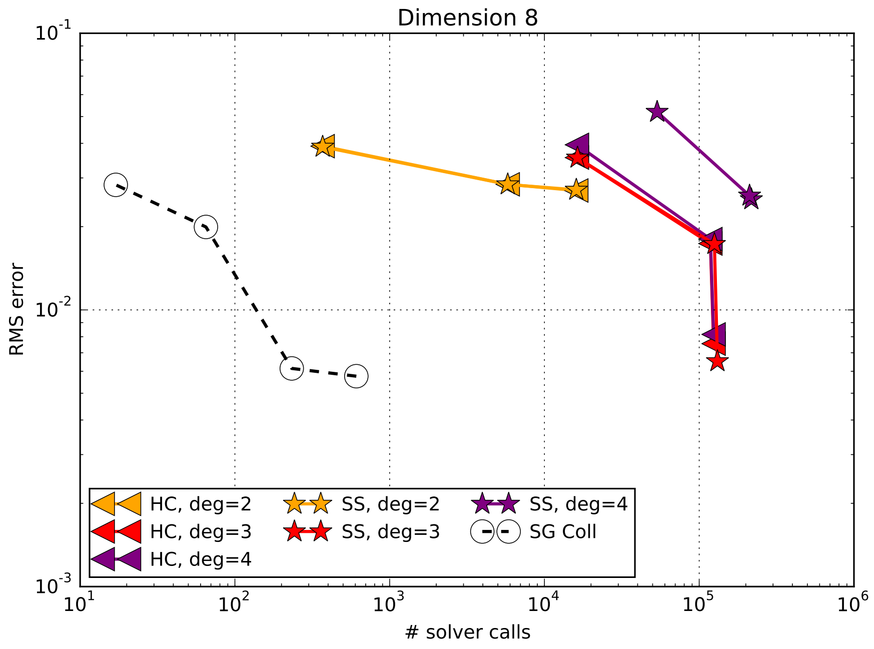


Figure 6.3.: Discrete projection for 8 stochastic dimensions

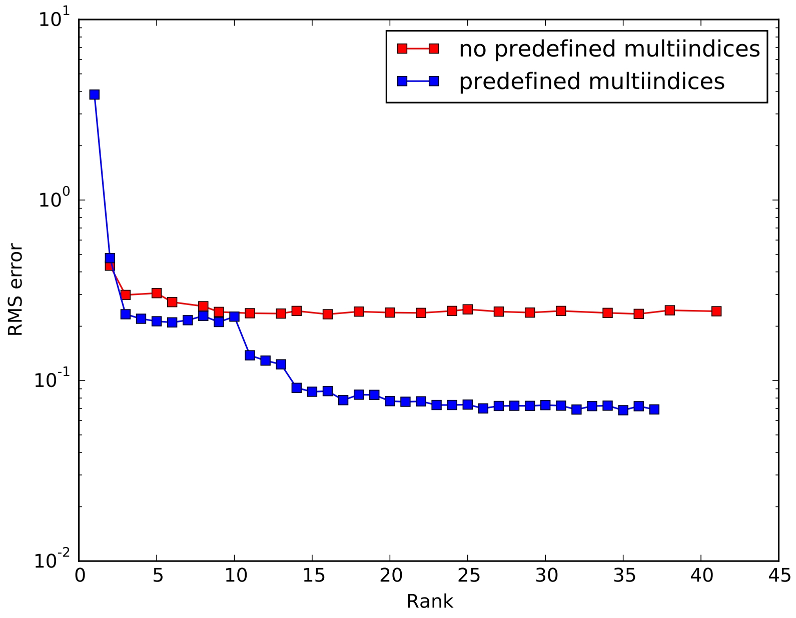


Figure 6.4.: Two tensor approximations, red without further predefined initial multiindices, blue with predefined multiindices in the first dimension

7. Conclusion and outlook

In this thesis we applied the non-intrusive discrete projection method on a real life FEM problem rather than the very simple example from Chapter 3.4. However, due to the usage of rigid predefined index sets (Definition 3.2) the amount of work which has to be invested makes the method impractical for high-dimensional problems. Without a priori knowledge of the problem which enables us to define a better suited predefined index set, either the ansatz degree has to be chosen very low, or an index set which is even sparser than the hyperbolic cross has to be chosen. Both approaches deteriorate the accuracy of the method. Another drawback of the method is that we have to solve a system $Av = b$ for each quadrature point, which is expensive for large systems, although the method allows the reusage of solutions of such solver calls. The interpolation method from Chapter 3.3.2 is basically impracticable, since all integrals corresponding to the index set have to be evaluated for each iteration of the solver.

The non-intrusive tensor approximation coupled with `ASGFEM` overcomes the problem of large index sets by constructing an optimal index set at run time. Clearly this is a major benefit compared to the discrete projection. Moreover, rather that having to solve a system for each quadrature point, we only need evaluations of the matrices $A(p)$ in the left hand side of (4.10) which is inexpensive, and a matrix-vector multiplication $A(p)u_r(p)$ in the right hand side which still requires less work than solving a system as in the discrete projection method. The tensor approximation only requires one solver call for each rank, which is another benefit of this method. Moreover, the method is equipped with an spatial error estimator (and mesh refinement) which none of the other discussed non-intrusive methods provide. The drawback of this method is that it is only applicable for linear problems such as the model problem (2.6),(2.7).

While the BFGS method does not require any solver calls at all, we have to perform for each rank several quadratures: one quadrature $R_y(x^{(k)})$ per BFGS iteration, plus (at least) three quadratures for the linesearch in each BFGS iteration (which may be omitted once the method converges). A benefit of this method is somewhat more general as is may be used to solve non-linear problems.

All discussed non-intrusive methods rely on high-dimensional quadrature. The implemented dimension-adaptive algorithm is crucial, since classical sparse grids still require far to many quadrature points. Unique hashing proves to be a simple method of accessing grid points efficiently, although it limits the quadrature dimension – depending on chosen maximal level – as we encounter hash values exceeding `int64` data types. The maximal levels used in this thesis allow for integration of at least 30 dimensions. The implemented parallelization of the quadrature is essential for run time.

Several improvements of the discussed methods are possible.

From viewpoint of implementation it would be beneficial to use Cython[1], a superset of Python which allows static typing (which Python lacks) and compiles to C, both are beneficial for run time of the application. Although using Cython does not require major changes in the code, this could not be done in time as one needs to identify where static typing proves to be beneficial. We have also noticed that integrals of the discrete projection and especially in the tensor approximation require quite different amounts of work. Hence a stagnation criterion based on (5.12) would be a more elaborate approach to decide when to stop the quadrature, rather than providing a small enough quadrature tolerance and limiting the quadrature with a large enough number of iterations.

From the mathematical viewpoint there are several improvements. For the quadrature, Dr. Eigel noted that it is beneficial to use global basis functions rather than local hat basis functions, in conjunction with non-equidistant quadrature points (compare Remark 5.5).

There is a recent paper improving the implemented dimension-adaptive quadrature algorithm, capable of using non-nested quadrature points and supporting quadrature on unbounded sets [NTTT15]. Moreover, there is a recent (unpublished) paper which deals with multilevel quadrature for PDEs with log-normal distributed random fields, which allows reusage of quadrature points [HPS].

[1]`http://www.cython.org`

A. Functional Analysis

Theorem A.1. *Gaussian theorem [Joh13, Corollary 3.43] Let $D \subset \mathbb{R}^d, d \geq 2$, be a bounded domain with Lipschitz boundary ∂D. Then, the following identity holds for all $u \in W^{1,1}(D)$*

$$\int_D \partial_i u(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = \int_{\partial D} u(\boldsymbol{s}) \boldsymbol{n}_i(\boldsymbol{s}) \,\mathrm{d}\boldsymbol{s}$$

with $\boldsymbol{n}_i$ the unit outer normal vector on ∂D and the i-th partial derivative ∂_i.

Theorem A.2. *First Green's formula [Joh13, Corollary 3.46] Let $D \subset \mathbb{R}^d, d \geq 2$, be a bounded domain with Lipschitz boundary ∂D. Then, the following identity holds for all $u \in H^2(D)$ and $v \in H^1(D)$:*

$$\int_D \nabla u(\boldsymbol{x}) \cdot \nabla v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x} = \int_{\partial D} \frac{\partial n}{\partial \boldsymbol{n}}(\boldsymbol{s}) v(\boldsymbol{s}) \,\mathrm{d}\boldsymbol{s} - \int_D \Delta u(\boldsymbol{x}) v(\boldsymbol{x}) \,\mathrm{d}\boldsymbol{x}.$$

Definition A.3. *Derivatives of functions [Joh14, Definition A.20] For a d-dimensional multi-index $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_d), \mu_j \in \mathbb{N}_0, j = 1, \ldots, d$ we define the derivative*

$$D^{\boldsymbol{\mu}} := \frac{\partial^{|\boldsymbol{\mu}|_1}}{\partial x_1^{\mu_1} \cdots \partial x_d^{\mu_d}}.$$

Definition A.4. *Function spaces [Joh14, Definition A.20], [Joh13, Ch. 3] We recall some common function spaces. We denote by $\boldsymbol{x}$ a point of the domain D.*

$$\begin{aligned}
C^k(D) &:= \{f : f \text{ and all its derivatives up to order } k \text{ are continuous in } D\} \\
L^k(D) &:= \left\{ f : \int_D |f(x)|^k \,\mathrm{d}x < \infty \right\}, \quad k \in [1, \infty) \\
C^\infty(D) &:= \bigcap_{j=0}^{\infty} C^j(D) \\
C_0^\infty(D) &:= \{f : f \in C^\infty(D), \operatorname{supp}(f) \subset D\}, \quad \operatorname{supp}(f) := \overline{\{\boldsymbol{x} \in D : f(\boldsymbol{x}) \neq 0\}} \\
W^{k,p}(D) &:= \{f \in L^p(D) : (\forall \boldsymbol{\mu} : |\boldsymbol{\mu}| \leq k) : D^{\boldsymbol{\mu}} f \in L^p(D)\}, \quad p \in [1, \infty), k \in \mathbb{N}_0 \\
H^k(D) &:= W^{k,2}(D) \\
H_0^1(D) &:= \overline{C_0^\infty(D)}^{\|\cdot\|_{W^{1,2}(D)}}
\end{aligned}$$

B. Computer science

Definition B.1. *Heap [KM14, Chapter 13] A heap is a tree based data structure. It is a binary tree which is complete (except possibly on deepest level), and satisfies the max-heap property such that parent nodes are larger that their children, and the w.l.o.g. left child has smaller value the right child. To maintain this structure, each insertion of an arbitrary element or the removal of the root node requires subsequent heapification - that is a reordering of the tree such that the max-heap property is satisfied. By the heap property, the largest element is always the root node and thus accessible in* $\mathcal{O}(1)$.

A heap is a basic data structure and thus provided by most programming languages.

Definition B.2. *`Axpy` [Ste98, Chapter 3.2] `Axpy` are optimized routines for linear combinations of vectors*

$$a \leftarrow a + kb,$$

for vectors a, b *and scalar* k*, or for linear combinations of matrices* A, B *in the form*

$$A \leftarrow A + kB.$$

The linear algebra backend `uBLAS` which is used in our implementation provides this functionality.

Bibliography

[Ach03] Achatz, S.: *Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten*, Technische Universität München, Diss., 2003

[Bel] Belk, J.: *Archimedes Parabola.* `https://commons.wikimedia.org/wiki/File:Archimedes_Parabola.svg`,

[Bel72] Bellman, R.: *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1972

[Che04] Chen, W.K.: *The Electrical Engineering Handbook.* Elsevier Science, 2004

[Edw12] Edwards, C.H.J.: *The Historical Development of the Calculus.* Springer New York, 2012 (Springer Study Edition)

[EEU07] Eiermann, M. ; Ernst, O. G. ; Ullmann, E.: Computational aspects of the stochastic finite element method. In: *Computing and Visualization in Science* 10 (2007), Nr. 1, S. 3–15

[EGSZ15] Eigel, M. ; Gittelson, C. J. ; Schwab, C. ; Zander, E.: A convergent adaptive stochastic Galerkin finite element method with quasi-optimal spatial meshes. In: *ESAIM: M2AN* 49 (2015), Nr. 5, S. 1367–1398

[Eps10] Epstein, M.: *The Geometrical Language of Continuum Mechanics.* Cambridge University Press, 2010

[EZ] Eigel, M. ; Zander, E.: *`ALEA` - A Python Framwork for Spectral Methods and Low-Rank Approximations in Uncertainty Quantification.* `https://bitbucket.org/aleadev/alea`,

[Feu05] Feuersänger, C.: *Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen*, Universität Bonn, Institut für Numerische Simulation, Diplomarbeit, 2005

[Feu10] Feuersänger, C.: *Sparse grid methods for higher dimensional approximation*, Technische Universität München, Diss., 2010

[GG98] Gerstner, T. ; Griebel, M.: Numerical integration using sparse grids. In: *Numerical Algorithms* 18 (1998), Nr. 3-4, S. 209–232

[GG03] Gerstner, T. ; Griebel, M.: Dimension-Adaptive Tensor-Product Quadrature. In: *Computing* 71 (2003), Nr. 1, S. 65–87

[GLL$^+$14] Giraldi, L. ; Litvinenko, A. ; Liu, D. ; Matthies, H. ; Nouy, A.: To Be or Not to Be Intrusive? The Solution of Parametric and Stochastic Equations—the "Plain Vanilla" Galerkin Case. In: *SIAM Journal on Scientific Computing* 36 (2014), Nr. 6, S. A2720–A2744

[GLMN15] Giraldi, L. ; Liu, D. ; Matthies, H. G. ; Nouy, A.: To be or not to be intrusive? The solution of parametric and stochastic equations — Proper Generalized Decomposition. In: *SIAM Journal on Scientific Computing* 37 (2015), Nr. 1, S. A347–A368

[GS91] Ghanem, R.G. ; Spanos, P.D.: *Stochastic Finite Elements: A Spectral Approach*. Dover Publications, 1991 (Civil, Mechanical and Other Engineering Series)

[GWZ14] Gunzburger, M. D. ; Webster, C. G. ; Zhang, G.: Stochastic finite element methods for partial differential equations with random input data. In: *Acta Numerica* 23 (2014), 5, S. 521–650

[HPS] Harbrecht, H. ; Peters, M. ; Siebenmorgen, M.: *Multilevel Accelerated Quadrature for PDEs with Log-Normal Distributed Random Coefficient*

[Joh13] John, V.: *Numerical Methods for Partial Differential Equations*. Lecture notes, 2013

[Joh14] John, V.: *Numerical methods for incompressible flow problems I*. Lecture notes, 2014

[JR11] Jakeman, J. D. ; Roberts, S. G.: Local and dimension adaptive sparse grid interpolation and quadrature. In: *arXiv preprint arXiv:1110.0010* (2011)

[Kan03] Kantorovitz, S.: *Introduction to Modern Analysis*. OUP Oxford, 2003 (Oxford Graduate Texts in Mathematics, No. 8)

[KM14] Kushwaha, D.S. ; Misra, A.K.: *Data structures - A programming approach with C*. PHI Learning, 2014

[KS10] Kornhuber, R. ; Schütte, C.: *Einführung in die Numerische Mathematik (Numerik I)*. Lecture notes, 2010

[Ler97] Lerner, L.S.: *Physics for Scientists and Engineers*. Jones and Bartlett, 1997 (Physics for Scientists and Engineers Bd. 2)

[LMW12] Logg, A. ; Mardal, K.-A. ; Wells, G.N.: *Automated Solution of Differential Equations by the Finite Element Method*. Springer New York, 2012

[MZ09] Ma, X. ; Zabaras, N.: An Adaptive Hierarchical Sparse Grid Collocation Algorithm for the Solution of Stochastic Differential Equations. In: *J. Comput. Phys.* 228 (2009), Nr. 8, S. 3084–3113

[MZ12] MATTHIES, H.G. ; ZANDER, E.: Solving stochastic systems with low-rank tensor compression. In: *Linear Algebra and its Applications* 436 (2012), Nr. 10, S. 3819 – 3838

[NTTT15] NOBILE, F. ; TAMELLINI, L. ; TESEI, F. ; TEMPONE, R.: An adaptive sparse grid algorithm for elliptic PDEs with lognormal diffusion coefficient / École Polytechnique Fédérale de Lausanne. 2015 (Nr. 04.2015). – Forschungsbericht

[Pfl10] PFLÜGER, D.: *Spatially Adaptive Sparse Grids for High-Dimensional Problems*, Technische Universität München, Diss., 2010

[Saa11] SAAD, Y.: *Numerical Methods for Large Eigenvalue Problems: Revised Edition.* Society for Industrial and Applied Mathematics, 2011 (Classics in Applied Mathematics)

[Smo63] SMOLYAK, S.A.: Quadrature and interpolation formulas for tensor products of certain classes of functions. In: *Dokl. Akad. Nauk SSSR* 4 (1963), S. 240–243

[Ste98] STEWART, G.W.: *Matrix Algorithms: Volume 1: Basic Decompositions.* Society for Industrial and Applied Mathematics, 1998. – ISBN 9781611971408

[Ull08] ULLMANN, E.: *Solution strategies for stochastic finite element discretizations*, Technische Universität Bergakademie Freiberg, Diss., 2008

[Van84] VANMARCKE, E.: *Random Fields: Analysis and Synthesis.* World Scientific, 1984

[Zan12] ZANDER, E.: *Tensor approximation methods for stochastic problems*, Technische Universität Carolo-Wilhelmina zu Braunschweig, Diss., 2012

[Zei95] ZEIDLER, E.: *Applied Functional Analysis: Applications to Mathematical Physics.* Springer New York, 1995 (Applied Mathematical Sciences Bd. 108)

Freie Universität Berlin

<u>Selbstständigkeitserklärung</u>

Name:	(Nur Block- oder Maschinenschrift verwenden.)
Vorname:	
geb.am:	
Matr.Nr.:	

Ich erkläre gegenüber der Freien Universität Berlin, dass ich die vorliegende ______________________________ selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: ___________________ Unterschrift: _____________________________

(_________________________)