Parallel RBF Kernel-Based Stochastic Collocation for Large-Scale Random PDEs

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Peter Zaspel

aus

Siegburg

Bonn 2015

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

1. Gutachter: Prof. Dr. Michael Griebel

2. Gutachter: Prof. Dr. Mario Bebendorf

Tag der Promotion: 24. April 2015

Erscheinungsjahr: 2015

To my parents

Abstract

In this thesis, the solution of large-scale uncertainty quantification problems is considered. *Uncertainty quantification* aims to extract stochastic (moment) information from processes with uncertain input data. These processes are here identified with random partial differential equations (PDEs). They have random input with respect to initial / boundary conditions, forcing terms, coefficients or domains. *Non-intrusive* methods are studied, in order to reuse existing PDE solvers. Applications are (elliptic) model problems and incompressible two-phase flows. The main contribution of this thesis is a new framework to solve these problems in a high-order convergent, scaling, parallel and optimal complexity fashion. To this end, the *radial basis function* (RBF) *kernel-based stochastic collocation method* is introduced. It combines high-order algebraic or even exponential convergence rates of spectral (sparse) tensor-product methods with optimal preasymptotic convergence of kriging and the profound stochastic framework of Gaussian process regression. The new method uses Lagrange bases from special reproducing kernel Hilbert spaces for approximation. Those Hilbert spaces are constructed from RBFs.

Numerical results show up to exponential convergence for model problems with high smoothness. For problems with low-smoothness, algebraic convergence rates are given. A small error in the preasymptotic regime is always achieved. Convergence results of (quasi-)Monte Carlo and (sparse) spectral tensor-product approaches are often clearly outperformed. An empirical error coupling analysis describes the interplay of all approximations, including conditions to balance all error contributions. Runtime complexity is expressed for a target error. Performance measurements show that a stochastic moment analysis for large-scale two-phase flow problems can be solved within a few hours. This excellent preasymptotic runtime becomes possible by parallelizing all relevant numerical methods, including a two-phase flow solver, iterative dense linear algebra solvers and all parts of the stochastic collocation on *graphics processing units* (GPUs). Most approaches scale across clusters of GPUs.

Optimal complexity and profound speedups are achieved by preconditioning of iterative sparse and dense linear solvers. Dense linear systems from interpolation are preconditioned with a localized restricted additive Schwarz method. Thereby, a new perfectly scalable preconditioner on multi-GPU clusters is constructed. Elliptic problems are solved with a newly implemented optimal Ruge-Stüben algebraic multigrid method. It uses CPU-based C/F splittings and parallelizes all remaining parts of the setup and solve phase on one GPU.

The *curse of dimensionality* is weakened or even broken for problems with fast decaying output covariance spectrum. To this end, *anisotropic RBF kernel-based stochastic collocation* is introduced. Optimal weights for two-phase flow problems are approximated by a Karhunen-Loève expansion of the solution flow field, which requires to solve a large-scale dense eigenvalue problem. Greedy optimization is used for optimal sampling in anisotropic space. Numerical experiments give profound (pre-)asymptotic results for elliptic and two-phase flow problems.

Overall, a combined effort of optimal numerical methods and parallel implementations allows to solve even large-scale uncertainty quantification problems in a small amount of time.

Contents

1 Introduction

Robustness and safety are major concerns in many engineering fields. The influence of small material defects on complex dynamical systems is studied in vehicle or architectural design. Potential danger from toxic processes or waste has to be analyzed in environmental engineering. Even more, the complicated interplay between industries and climate change with potential global impact, is studied in climatology. All these missions have the common characteristics that stochastic information is needed of processes, which are subject to uncertain or unreliable data. This idea is recently summarized under the notion of *uncertainty quantification* (UQ) in the fields of mathematics and engineering. More specifically, uncertainty quantification mainly addresses *numerical simulations* of physical and engineering processes. The objective is thus to get robust information and measurements for variations out of simulations.

From a mathematical point of view, the underlying task is to compute the solution of e.g. partial differential equations (PDEs) with coefficients, initial and boundary conditions, domains or forcing terms being dependent on some random input from a stochastic space $(\Omega, \mathcal{F}, P)$. In this thesis, such problems will be called *random PDE problems.* With $\mathcal{L}$ an operator describing a general PDE in space $\bar{\mathcal{D}}$ and time $[0,T]$, and $\boldsymbol{a}, \boldsymbol{b}$ general space-time stochastic processes that describe random parameters, random PDE problems have solutions $\boldsymbol{u} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^r$ such that almost surely

$$\mathcal{L}(\boldsymbol{a}(\omega,\boldsymbol{x},t))\boldsymbol{u}(\omega,\boldsymbol{x},t) = \boldsymbol{f}[\boldsymbol{b}(\omega,\boldsymbol{x},t)](\omega,\boldsymbol{x},t) \qquad \text{in } \Omega \times \bar{\mathcal{D}} \times [0,T] \tag{1.1}$$

holds. Here, the general PDE operator is expected to encode initial and boundary conditions. In uncertainty quantification, the objective is to evaluate statistical moments of the random PDE solution with the most common examples of expectation value $\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x},t)$ and variance $\mathrm{Var}\left[\boldsymbol{u}\right](\boldsymbol{x},t)$. In airplane design, this might be the mean velocity field and its variance. It is often of further interest to compute moments of *quantities of interest* (QOI). These are derived values from solution fields like extremal values or integrals of the velocity.

There are two major approaches to approximately solve random PDE problems, namely *intrusive* and *non-intrusive* methods. An intrusive technique simultaneously discretizes stochastic and physical space with the classical example of stochastic Galerkin approaches [GS91, XK02, BTZ04, FST05, SG11]. Even though this method delivers following [Xiu10, Section 7.4] favorable properties such as small errors with fewer number of equations and potentially small overall run-time, it requires to re-discretize and re-implement existing deterministic PDE solvers which might become rather involved because of the Galerkin reformulation. Non-intrusive techniques (e.g. *stochastic collocation*) reuse existing PDE solvers and generate a series of deterministic solutions which are used to approximate stochastic moments. It is thereby possible to perform uncertainty quantification analysis even for very complex large-scale applications for which a re-implementation of existing solvers is no option. However, this is connected to a higher computational effort, with at least hundreds, thousands or even more deterministic problems that have to be solved. Noting that a *single* high-resolution simulation often requires computational resources in the range of hours to days on a parallel computer, a moment analysis for a large-scale random PDE problem then seems not to be tractable.

Important examples for large-scale PDE problems arise in the analysis of *fluid dynamics*. Fluids are gases, liquids or plasmas. They play a key role in applications like vehicle design, droplet and bubble dynamics or future nuclear fusion plants. Indicators for the complexity of understanding and modeling their underlying continuous laws are e.g. the open questions with regard to the existence and uniqueness of strong solutions of the three-dimensional transient *Navier-Stokes equations*, which are formulated as one of the Millennium Prize Problems [CJW$^+$06]. The Navier-Stokes equations are a mathematical model for the description of incompressible fluids in continuous space. They relate the velocity and pressure of a fluid to its material parameters such as viscosity or density in presence of forces and boundary conditions. While the exact mathematical properties of these and some related equations are still open, their approximate numerical solution has become a main tool in the field of *computational fluid dynamics* (CFD). Flow simulations, thus the numerical solution of the underlying equations, replace real-world experiments in many cases.

In this thesis, uncertainty quantification shall be made feasible for large-scale complex random PDE problems by means of non-intrusive uncertainty quantification. Besides of model problems, large-scale complex uncertainty quantifications applications shall be exemplified by two-phase flow simulations modeled by the three-dimensional two-phase incompressible Navier-Stokes equations. They are given with $i = 1, 2$ as

$$\begin{aligned}
\rho_i \partial_t \boldsymbol{u}_i + \rho_i (\boldsymbol{u}_i \cdot \nabla) \boldsymbol{u}_i &= \nabla \cdot \mu_i (\nabla \boldsymbol{u}_i + \{\nabla \boldsymbol{u}_i\}^T) - \nabla p_i + \rho_i \boldsymbol{g} && \text{in } \mathcal{D}_i \times [0,T]\,, \\
\nabla \cdot \boldsymbol{u}_i &= 0 && \text{in } \mathcal{D}_i \times [0,T]\,, \\
\boldsymbol{u}_1 &= \boldsymbol{u}_2 && \text{on } \Gamma_f \times [0,T]\,, \\
[\mathbf{T}] \cdot \boldsymbol{n}_{\Gamma_f} &= \sigma \kappa \boldsymbol{n}_{\Gamma_f} && \text{on } \Gamma_f \times [0,T]\,,
\end{aligned}$$

skipping initial and boundary conditions. The two-phase Navier-Stokes equations describe the interaction of two non-mixing fluids like water and oil or water and air (at low Mach numbers) in domains $\mathcal{D}_1$ and $\mathcal{D}_2$ with important applications such as fluvial construction analysis or bubble flows in chemical bubble reactors. Approximating a *single* two-phase flow problem is a large computational effort. Here, the application of non-intrusive uncertainty quantification is almost impossible without accomplishing the following tasks:

1. A non-intrusive uncertainty quantification method with **high- to exponential-order error convergence** and **excellent pre-asymptotic error behavior** has to be found.

2. All numerical components including the stochastic part and PDE solvers have to be constructed at **optimal runtime complexity**.

3. The numerical methods have to be optimized for **small pre-asymptotic runtime**.

4. To solve large-scale problems, a parallelization for distributed-memory parallel compute clusters with **optimal parallel scalability** has to be made.

5. If the underlying problem exposes enough structure, the **curse of dimensionality** in stochastic parameter space has to be **broken or weakened**.

This is where this thesis starts. A *multi-disciplinary approach* is proposed, to solve all above problems. It combines optimal numerical methods and empirical error analysis from *mathematics* with algorithmic developments from *computer science* and hardware-aware programming from the field of *high performance computing* (HPC).

RBF kernel-based stochastic collocation for two-phase flows

As motivated before, the approximation of large-scale random PDE problems with existing solver implementations requires non-intrusive uncertainty analysis. To achieve high convergence rates and excellent preasymptotic behavior for uncertainty quantification, *radial basis function (RBF) kernel-based stochastic collocation* is introduced. Starting from the general random PDE problem (1.1), whose solutions are at least in the Bochner space $L^2(\Omega;[0,T];L^2(\mathcal{D}))$, the first important step towards a stochastic collocation approximation is the introduction of a *finite-dimensional noise assumption* [BNT10]. This gives, after measure change, the problem to find $\boldsymbol{u}:\Gamma\times\mathcal{D}\times[0,T]\to\mathbb{R}^r$ such that

$$\mathcal{L}(\boldsymbol{a}(\boldsymbol{y},\boldsymbol{x},t))\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)=\boldsymbol{f}[\boldsymbol{b}(\boldsymbol{y},\boldsymbol{x},t)](\boldsymbol{y},\boldsymbol{x},t) \qquad \text{in } \Gamma\times\bar{\mathcal{D}}\times[0,T] \tag{1.2}$$

for all $\boldsymbol{y}\in\Gamma$ on $(\Gamma,\mathcal{B}^{N_{FN}},\rho d\boldsymbol{y})$. Here, Γ is a finite-dimensional (tensor-product) space, such as $[-r,r]^{N_{FN}}$ with stochastic dimension N_{FN}. Solutions are in the Bochner space $L^2(\Gamma;[0,T];L^2(\mathcal{D}))$. The connection between equations (1.1) and (1.2), thus the finite-dimensional noise assumption, is often fulfilled by an approximation of the random input fields $\boldsymbol{a},\boldsymbol{b}$ by the Karhunen-Loève expansion, e.g.

$$\boldsymbol{a}(\omega,\boldsymbol{x},t)\approx\boldsymbol{a}_{KL}(Y_1^{\boldsymbol{a}}(\omega),\dots,Y_{N_{KL}}^{\boldsymbol{a}}(\omega),\boldsymbol{x},t)=\mathbb{E}\left[\boldsymbol{a}\right](\boldsymbol{x},t)+\sum_{m=1}^{N_{KL}^{\boldsymbol{a}}}\sqrt{\lambda_m^{\boldsymbol{a}}}Y_m^{\boldsymbol{a}}(\omega)\boldsymbol{\psi}_m^{\boldsymbol{a}}(\boldsymbol{x},t)$$

with $\boldsymbol{a}_{KL}(\boldsymbol{y},\boldsymbol{x},t)\cong\boldsymbol{a}_{KL}(Y_1^{\boldsymbol{a}}(\omega),\dots,Y_{N_{KL}}^{\boldsymbol{a}}(\omega),\boldsymbol{x},t)$. The function $\boldsymbol{u}_{KL}\in L^2(\Gamma;[0,T];L^2(\mathcal{D}))$ is the solution of (1.2) for approximated input space-time stochastic processes $\boldsymbol{a}_{KL},\boldsymbol{b}_{KL}$. By this construction, we introduce a *finite noise or Karhunen-Loève error.* It depends on the truncation error of the Karhunen-Loève expansion and is related to the correlation length or covariance spectrum of the random input and the random PDE, cf. [ST06, Theorem 2.7]. In many cases, it is necessary to keep several terms in the expansion. Each term introduces one stochastic dimension, leading very often to a *higher-dimensional problem* in the stochastic parameter $\boldsymbol{y}\in\Gamma\subset\mathbb{R}^{N_{KL}}$ with the well-known *curse of dimensionality* [BG04] for approximation.

Many non-intrusive approaches have been proposed to solve the above random PDE problem. These include, but are not limited to standard Monte Carlo sampling, quasi-Monte Carlo based methods [GKN+11], multi-level Monte Carlo approaches [CGST11, BSZ11, DKS13], (generalized) polynomial chaos [Sud08, Eld09, EMSU12] and stochastic collocation [NTW08b, BNT10]. While Monte Carlo sampling leads to a dimension-independent convergence rate of $O({N_\Gamma}^{-1/2})$ in stochastic space, quasi-Monte Carlo methods try to achieve rates of $({N_\Gamma}^{-1})$ without dimension dependence. In contrast, multi-level Monte Carlo methods stick to standard Monte Carlo sampling but try to reduce the overall computational complexity by using multiple resolution levels in PDE space and stochastic space. Uncertainty quantification by polynomial chaos uses polynomial expansions in stochastic space to approximate the underlying stochastic process. It has close connections to similar stochastic collocation constructions. If applied to appropriately smooth problems, polynomial chaos expansion allows to achieve exponential convergence. An in-depth discussion of these and other methods is well beyond the scope of this thesis. More details on spectral methods are given in [LMK10], while [SG11] reviews some of the methods with special focus on sparse tensor discretizations. In [Xiu09], a broader review is given. As mentioned before, in this thesis, the very general framework of stochastic collocation is used to solve random PDE problems. Therefore, this method is introduced in more detail.

Stochastic collocation In stochastic collocation, deterministic solutions of the (random) PDE problem (1.2) for fixed samples or *collocation points* $X_\Gamma := \left\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_\Gamma}\right\} \subset \Gamma$ of the finite-dimensional stochastic space Γ are computed. The collocation method uses a Lagrange basis $\{L_i\}_{i=1}^{N_\Gamma} \subset \mathcal{P}(\Gamma)$ to interpolate in the full space Γ leading to approximations

$$\boldsymbol{u}_{KL} \approx (\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{KL}) \in \mathcal{P}(\Gamma) \otimes L^2([0,T]; L^2(\mathcal{D})), \quad (\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{KL})(\boldsymbol{y}, \boldsymbol{x}, t) = \sum_{i=1}^{N_\Gamma} \boldsymbol{u}_{KL}(\boldsymbol{y}_i, \boldsymbol{x}, t) L_i(\boldsymbol{y})$$

with a *stochastic collocation approximation error.* Since we numerically solve (1.2) by discretization in space and time, we further introduce the PDE approximation

$$\boldsymbol{u}_{KL} \approx D_{h_\mathcal{D},\delta t}\boldsymbol{u}_{KL}\,,$$

with $D_{h_\mathcal{D},\delta t}$ the operator to project on the finite-dimensional PDE solution space $L^2(\Gamma) \otimes V_{h_\mathcal{D},\delta t}([0,T] \times \mathcal{D})$ and a *PDE approximation error.* Stochastic moments, e.g. the mean, are approximated by applying quadrature with the approximation

$$\mathbb{E}\left[\boldsymbol{u}_{KL}\right](\boldsymbol{x}, t) = \int_\Gamma \boldsymbol{u}_{KL}(\boldsymbol{y}, \boldsymbol{x}, t)\rho(\boldsymbol{y})d\boldsymbol{y} \approx (Q^{l,N_{KL}}\boldsymbol{u}_{KL})(\boldsymbol{x}, t)$$

with an additional *quadrature error.* In Section 4.6, the usual error splitting for the first moment approximation of stationary space-dependent random PDE problems is discussed. It has the structure

$$\left\|\mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right\|_{L^2(\mathcal{D})} \leq \varepsilon_{KL} + \varepsilon_{h_\mathcal{D}} + \varepsilon_{N_\Gamma} + \varepsilon_Q \tag{1.3}$$

with some Karhunen-Loève, PDE discretization, stochastic collocation and quadrature error on the right-hand side.

This discussion motivates to handle the different approximations independently. Therefore, it gives rise to a series of choices to be made for the construction of a stochastic collocation method. Ignoring here the Karhunen-Loève approximation error, these choices are the stochastic approximation space $\mathcal{P}(\Gamma)$ with collocation points X_Γ, the quadrature method $Q^{l,N_{KL}}$ to evaluate moments and the PDE discretization approach.

The currently most widely used approach in stochastic collocation is to use a spectral tensor-product approximation in the stochastic space Γ, thus the approximation space $\mathcal{P}(\Gamma)$ is the tensor product space of polynomials. Increasing the number of collocation points is equivalent to increasing the polynomial degree. Furthermore, collocation points are roots of orthogonal polynomials that correspond to the given distribution. Classical results for this standard approach are discussed in [BNT10]. To overcome the curse of dimensionality stemming from the construction of a higher-dimensional tensor-product grid, sparse approximations by the Smolyak sparse grid technique [BG04] have been introduced to stochastic collocation [NTW08b]. Some literature with comparisons to other methods is available [BNTT11, TPME11, CQR14]. Stochastic collocation is applied to many applications, e.g. [MHZ05, MX09, SM11, BS13, ZLY$^+$13]. More background on this topic is given in overview literature like [LMK10] and recent publications include, but are not limited to [NTW08a, JNX13, TJWG14, ES14].

Approximation in reproducing kernel Hilbert spaces In contrast to the classical spectral stochastic collocation approaches, this thesis introduces reproducing kernel Hilbert spaces (RKHS), cf. Section 4.2, as stochastic collocation approximation spaces with radial-symmetric basis function (RBF) kernels $k(\boldsymbol{y}, \boldsymbol{y}') := \varphi(\|\boldsymbol{y} - \boldsymbol{y}'\|_2)$. Note, that there is related recent work on kernel-based collocation for the approximate solution of stochastic partial differential equations [CFY12, FY13b] and the special-case of an elliptic random PDE [FY13a]. However this approach applies collocation in physical space and therefore is an intrusive method. Other related work is [LWB07] which uses radial basis functions for interpolation in stochastic space without a profound mathematical framework. Therefore, the here given approach is new.

Interpolation in reproducing kernel Hilbert spaces allows for an approximation of a given function $f \in \mathscr{F}$ by

$$f(\boldsymbol{y}) \approx s_{f,X}(\boldsymbol{y}) := \sum_{j=1}^{N_\Gamma} \alpha_j k(\boldsymbol{y}, \boldsymbol{y}_j), \quad s_{f,X}(\boldsymbol{y}_i) = f(\boldsymbol{y}_i) \;\forall i = 1, \dots, N_\Gamma, \quad X_\Gamma = \left\{\boldsymbol{y}_j\right\}_{j=1}^{N_\Gamma}$$

with X_Γ the *collocation points* or *centers*, as before. To formally introduce an *RBF kernel-based stochastic collocation*, the Lagrange basis in the so-called *native space* $\mathcal{N}_k(\Gamma)$ is used. The native space is derived by completion of $F_k(\Gamma) := \text{span}\,\{k(\cdot, \boldsymbol{y}) | \boldsymbol{y} \in \Gamma\}$. It is a reproducing kernel Hilbert space. Throughout this thesis, the stochastic approximation space is

$$\mathcal{P}(\Gamma) := \mathcal{N}_k(\Gamma)\,,$$

for changing *strictly positive definite radial* kernel functions k. Note that the interpolant $s_{f,X}$ of a function $f \in \mathcal{N}_k(\Gamma)$ is always its best approximation in space $\mathcal{N}_k(\Gamma)$ with respect to the native space (semi-)norm $|\cdot|_{\mathcal{N}_k(\Gamma)}$, cf. [Wen04, Chapter 13]. Furthermore, it minimizes this norm for all functions in $\mathcal{N}_k(\Gamma)$ that interpolate f [Wen04, Chapter 13]. These optimality results are not available in many of the standard collocation approaches. To exemplify this, we can state the fact that the native spaces of e.g. *Wendland* and *Matérn* kernels are specific Sobolev spaces. Consequently, kernel interpolation will lead to *best-approximations* with norm-minimality properties *in these Sobolev spaces.*

Besides these optimality properties, the close relationship of kernel-based approximation to *kriging* [Kri51, Mat63, Ste99, vK04] with its well-known behavior of low error for few collocation points makes it an optimal candidate for the target application. Moreover, the given method has similarities to *Gaussian process regression* [RW05] with a profound stochastic framework.

Depending on the smoothness of the approximated function f and the choice of kernel and collocation points, there exist convergence results for kernel-based approximation with higher-order algebraic rates or even exponential rates, cf. Section 4.3.2. Most of these convergence results rely on a very even distribution of the collocation points X_Γ, which is e.g. given for some quasi-Monte Carlo point sequences like the Halton sequence [Hal60]. Even though the implementation of kernel-based stochastic collocation will allow for many samplings including Monte Carlo, quasi-Monte Carlo, tensor product grids or sparse grids, the numerical results will be mainly based on points sets X_Γ constructed by a Halton sequence.

By that way, the existing theoretical results on kernel approximation convergence will be carried over to the new kernel-based stochastic collocation method, allowing to outperform low-order algebraic convergence rates from (quasi-)Monte Carlo techniques in asymptotic convergence, in most cases. Even more, exponential convergence rates known from e.g. stochastic collocation with tensor product or sparse-grid constructions of orthogonal polynomials will be matched in asymptotic behavior and the error behavior is even outperformed in preasymptotics for sufficiently smooth problems. Empirical results will show that kernel-based stochastic collocation achieves higher convergence rates than classical spectral (sparse) tensor-product methods for problems of finite, low smoothness. The mesh-less construction of RBF interpolants further gives the advantage of approximation with arbitrary numbers of collocation points instead of dyadic constructions known from tensor-product methods and sparse grids. This is very important in the preasymptotic regime. Overall, RBF kernel-based stochastic collocation is a perfect fit for the challenge to introduce a high- to exponential-order stochastic approximation with excellent preasymptotic behavior.

Higher-dimensional quadrature Evaluating stochastic moments requires quadrature. It will be possible to reduce the quadrature problem to a weight coefficient approximation requiring only quadrature of kernel functions and products of kernel functions. Thereby, the major part of the stochastic moment evaluation problem complexity is moved to the stochastic collocation or interpolation.

The integrals that have to be approximated are higher-dimensional integrals with stochastic dimension N_{KL}. In some cases, solving these can be done by analytical means. However, usually numerical quadrature is preferred to get flexibility for arbitrary kernel functions. A series of textbook quadrature rules for higher-dimensional problems ranging from (quasi-) Monte Carlo integration [Caf98] up to Clenshaw-Curtis quadrature on sparse grids [GG98] will be implemented. The univariate Clenshaw-Curtis quadrature rule will be implemented with optimal $O(n \log n)$ complexity. Overall, sparse and full tensor-product quadrature with Clenshaw-Curtis rules will be applied such that overall optimal high order convergence and optimal complexity of is achieved.

Discretization of the two-phase Navier-Stokes equations As initially discussed, besides of model problems, the target large-scale complex random PDE problem is the two-phase incompressible Navier-Stokes equation system. It is discretized by finite differences / volumes on a staggered discretization grid. Chorin's projection approach [Cho67, Cho68] allows to solve the equation system over time. Higher-order space (e.g. ENO, WENO, VONOS, QUICK) and time (Runge-Kutta of third order, Adams-Bashforth) discretizations are applied. A level-set function [OS88] helps to distinguish the two flow phases [SSO94]. The existing flow solver package NaSt3DGPF [DGN98, STCE06, CGS09] is applied, featuring these discretizations. It runs on clusters of standard processors by a domain decomposition approach with the message passing interface (MPI). Many applications are covered by this flow solver, including river simulations in presence of hydraulic constructions [STCE06], coating processes, bubble and droplet dynamics [CGS09], porous media flows [VCG$^+$08], rigid body interaction [Cro10], sediment transport [BG13], non-Newtonian flows [Cla08, GR14], contact angle dynamics [GK14] and animation [ZG11]. Other work on the numerical handling of incompressible two-phase flows is proposed e.g. in [KFL00, SSH$^+$07, GR11, TSLV11].

Parallel computing on graphics processing units Small pre-asymptotic runtime will be achieved by implementing *all* relevant numerical methods on *graphics processing units* (GPUs). This highly parallel hardware is very popular in high performance computing, cf. [OLG$^+$07] for an early review. Fully dedicated parallel processing GPU hardware is available with appropriate programming languages (CUDA, OpenCL, ...) [SK10, Far11, MGM$^+$11]. For those algorithms that can be reformulated in a way that many similar instructions are executed on structured data, GPUs tend to outperform equally-priced multi-core standard processors in terms of runtime, because of higher memory bandwidth and peak performance, and in terms of power consumption. This gives rise to the assumption that well-optimized implementations on GPUs are a good choice for optimal performance of the underlying numerical algorithms. Today, compute clusters equipped with GPUs are among the fastest compute systems in the world [Str06]. They are further assumed to be a starting point for next-generation *Exascale* parallel computing clusters with exaFLOP performance, thus 10^{18} floating point operations per second (FLOPS). Adaptation and redesign of new algorithms for such systems is critical. To summarize, the two major reasons to apply GPUs throughout this thesis are their high performance and the requirement of new hardware-aware numerical methods which achieve maximum performance on the next generation of HPC systems.

Parallelization for distributed memory parallel computers with GPUs While all relevant numerical methods are parallelized to run on at least a single GPU, some parts are even further parallelized to scale on a distributed memory compute cluster equipped with GPUs, thus a *multi-GPU cluster*.

A central example is the two-phase flow solver. Its core components will be *re-implemented* and *parallelized* on a multi-GPU cluster to solve largest-scale problems. Optimization techniques for stencil computations and high multi-GPU scalability have to be discussed and applied. In terms of single-GPU performance, a factor three speedup on GPUs over equally priced CPUs, i.e. standard processors, is shown. This is a decent result. Furthermore, improvements in power consumption by a factor of two will be highlighted. A multi-GPU scalability analysis will outline almost optimal weak scaling on up to 48 GPUs. Note that the results on multi-GPU parallelization of NaSt3DGPF are already published by the author in [GZ10, ZG13]. At time of publication, this was the first two-phase incompressible flow solver purely running on GPUs and on multi-GPU clusters. Related work included [TS09, GBWT09, Mic09, Kel09, JTS10, WA11, JS11, CL11, KCLL11]. The results of [GZ10, ZG13] are summarized in Chapter 5 with some recent updates on technological advancements.

In addition to the multi-GPU parallel flow solver, large parts of the kernel-based stochastic collocation method are parallelized for multi-GPU clusters and delivered as modular libraries. As an example, the new multi-GPU parallel iterative solver library for the dense linear systems *parla*, used for kernel interpolation, is implemented. It will be shown to have almost optimal parallel scalability on a GPU cluster.

Model-problems and real-world applications A series of moment convergence results with respect to different problem classes and kernel functions with further comparisons to (sparse) spectral tensor-product approximations are given. Model problems will range from problems with a known analytic solution to elliptic partial differential equations with random coefficients that are either modeled as piece-wise constant random fields or given by a Karhunen-Loève expansion. A major focus is given to the random two-phase incompressible Navier-Stokes equations. Note that there is a large body of literature on uncertainty quantification in computational fluid dynamics with applications in e.g. groundwater flow [CL91, GKW^{+}08], incompressible flows [KL06, Wan11, LMK10, Sch11, PS12, BSS13, DV13, CDBC13, TLN14, SHL14], compressible flows [LB08, BLMS13, CCGC13] and other applications [Roa97, MHZ05], to name a few. However, to the authors knowledge, uncertainty quantification has never been applied to the two-phase incompressible Navier-Stokes equations before.

Three important application problems which base on the random (two-phase) Navier-Stokes equations will be analyzed. The first is the flow over a backward facing step, representing applications in river construction design. It has a random inflow velocity, density and viscosity. The objective is to find stochastic properties in the vortex reattachment length. As second problem, a rising bubble simulation in presence of a domain-dependent random volume force modeled by a Karhunen-Loève expansion is discussed. Finally, the real-world application of stochastic homogenization for chemical bubble reactors is outlined. The last two test cases will be considered large-scale problems. Their run-time per single deterministic solution is in the range of several hours on a 12-core CPU system. Convergence studies would become unfeasible for larger problem sizes.

By using GPU-based stochastic analysis and the multi-GPU parallel flow solver, it will be possible to perform the given studies in a fast and parallel scalable way with high- to exponential-order convergence and excellent preasymptotics. Error convergence results of stochastic collocation by global (sparse) tensor-product approximations will be outperformed in the preasymptotic regime and matched in the asymptotics, in many cases. Higher convergence rates are achieved by the kernel-based methods for random PDE problems of finite, low smoothness.

Empirical error coupling From (1.3), we know that the approximation of stochastic moments of random PDE solutions involves a series of numerical approximation errors. One important step will be to analyze these errors and their coupling, to give indicators for optimal error balancing. In fact, we look for the a-priory unknown relationship between a given target approximation error and the required minimum number of collocation points, quadrature points and PDE discretization points to get below that error. In some applications, it is possible to give some of the necessary error estimates by means of numerical analysis, cf. [GKW^{+}08, NTW08a, BNT10, BSZ11, BNTT14, ES14]. Then, with knowledge on all involved errors, a balancing becomes possible. However, in this thesis, the true application problem are the random three-dimensional two-phase incompressible Navier-Stokes equations in strong form. Their deterministic counterpart has no solution theory. Furthermore, an analysis for coupled stochastic influences in coefficients, boundary conditions and initial conditions is complicated anyway. This is why the error coupling analysis will be done in terms of empirical studies of the different error parts, cf. Section 6.3. As a result, *empirical* conditions will be given, to achieve a given fixed mean approximation error tolerance for the different discretization parameters. Moreover, the computational complexity will be expressed in terms of that error tolerance. Both results will be new in that field.

Parallel preconditioning in uncertainty quantification

In the first part of this thesis, asymptotic computational complexities of the overall stochastic collocation method will be given. For elliptic model problems, we get a complexity of

$$O\left(N_\Gamma {N_\mathcal{D}}^2 + N_\Gamma \left(N_Q^{l,1} \log N_Q^{l,1}\right)^{N_{KL}} + {N_\Gamma}^3\right) ,$$

with N_Γ the number of collocation points, $N_\mathcal{D}$ the number of finite difference discretization grid points, N_Q the number of quadrature points per stochastic dimension and N_{KL} the number of stochastic dimensions corresponding to the number of terms in the Karhunen-Loève approximation. In the two-phase Navier-Stokes case, this extends to

$$O\left(N_\Gamma {N_\mathcal{D}}^2 N_\mathcal{T} + N_\Gamma \left(N_Q^{l,1} \log N_Q^{l,1}\right)^{N_{KL}} + {N_\Gamma}^3\right) .$$

$N_\mathcal{T}$ is the number of time steps. The first term corresponds to the solution of the N_Γ random PDE problems, the second summand characterizes quadrature and the third part of the estimate describes the computational complexity of solving the kernel approximation problem for non-compactly supported kernels such as Gaussian or Matérn kernels. As we will see, the quadrature problem might be solved by a widely problem-independent precomputing step or even analytically. Therefore, it is not considered to be of dominant nature. Clenshaw-Curtis quadrature is fast anyway. However, the computational complexity $O(N_\Gamma {N_\mathcal{D}}^2)$ or $O(N_\Gamma {N_\mathcal{D}}^2 N_\mathcal{T})$ of all PDE runs and the complexity $O({N_\Gamma}^3)$ of kernel approximation has to be reduced, solving the problem of getting an overall optimal complexity method. This is the objective of the second part of this thesis. Both dominant complexities base on numerical linear algebra. Iterative methods with efficient preconditioners will reduce these complexities.

Local preconditioning for kernel approximation Let us start here with the discussion of the optimization for optimal complexity of kernel approximation. Interpolation by non-compactly supported kernels requires to solve a dense linear system, with $O({N_\Gamma}^3)$ complexity for both LU factorizations and (unpreconditioned) iterative methods. This leads to prohibitive runtimes with hundreds of thousands or even millions of collocation points. Also, with growing number of points, the condition of these dense linear systems usually explodes leading to bad numerical approximations even in case of direct solvers. Therefore, *local* kernel *Lagrange basis* functions shall be introduced following [FHN$^+$13]. They give rise to preconditioners for Krylov subspace solvers with *dense* matrices. Local Lagrange basis functions

$$\widetilde{L}_i(\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{y} = \boldsymbol{y}_i \\ 0 & \text{if } \boldsymbol{y} \in \widetilde{X}_i \setminus \boldsymbol{y}_i \end{cases} \qquad \text{with} \qquad \widetilde{L}_i(\boldsymbol{y}) = \sum_{j=1}^{N_i} \widetilde{\alpha}_j^i k(\boldsymbol{y}, \boldsymbol{y}_{i_j}) .$$

approximate standard Lagrange basis functions as they are constructed by using point-wise local point neighborhoods $\widetilde{X}_i$ instead of the full point set. In [FHN$^+$13], it has been shown that the resulting *preconditioner* for the linear system in a kernel interpolation problem, with conditionally positive definite thin splines, is *optimal* for problems on a sphere, thus without boundary. Optimal preconditioners in iterative solvers lead to problem-size independent convergence. As we will see, the proposed preconditioner can also be identified as a special *restricted additive Schwarz* [CS99] preconditioner.

In this thesis, this preconditioning together with a Krylov subspace iterative solver for *dense* matrices is introduced to kernel-based stochastic collocation with strictly positive definite kernels. It will be exemplified for Matérn kernels. As it turns out, applying the preconditioner to approximation problems *with* boundary does only slightly increase the number of iterations in the linear solver. Therefore, it is a strong preconditioner leading to a final complexity of $O(cN_\Gamma{}^2)$ operations with a Krylov subspace solver for *dense* matrices. In fact, following e.g. [BN92, BL97, BCM99, Yin06, Wen06, GD07], it might be even possible to reduce this complexity to $O(cN_\Gamma \log N_\Gamma)$ by replacing the dense matrix-vector product by a fast multipole method. However, this is future work on GPUs in higher-dimensional spaces. In [BCM99, BLB00, LK04], a domain decomposition method gives performance improvements. Another preconditioning technique is used in [SCM12, Che13] with implementations for large CPU clusters in [AS]. Further work on parallel scalable RBF interpolation is [YBK09]. Nevertheless, to the authors knowledge, this thesis realizes the first multi-GPU parallel preconditioned, almost optimal solver based on the local Lagrange approach. The application of this technique in stochastic collocation is new, anyway.

The full preconditioned iterative method will be implemented on a multi-GPU cluster, as said before. Note that the construction of the preconditioner involves to solve many small and local interpolation problems. For this reason, it is an optimal candidate for Exascale algorithms which shall be extremely parallel and local, in the best possible case. Due to the iterative nature of the method, it is also error-resilient. It will be shown that an appropriate domain-decomposition multi-GPU parallelization indeed shows *optimal strong scaling* results. Because of the much better computational complexity, it will be possible to show scalability in the problem size, thus kernel-based stochastic collocation in the range of million collocation points will become feasible. This will be exemplified for an elliptic random PDE problem with a higher-dimensional stochastic space. Overall, this work solves the problems of achieving almost optimal complexity with low runtime at optimal parallel scalability for the stochastic collocation analysis.

Algebraic multigrid for random PDE problems Solving the random PDE problems of this thesis involves $O(N_\Gamma N_\mathcal{D}{}^2 N_\mathcal{T})$ operations for the two-phase Navier-Stokes equations and $O(N_\Gamma N_\mathcal{D}{}^2)$ operations for elliptic problems, as explained before. While the necessary number of collocation points is related to the approximation of the stochastic collocation method, the quadratic complexity in the number of grid points is due to the sparse Krylov subspace solvers for the solution of discretized elliptic problems. These show up in the random elliptic PDEs and in the pressure Poisson equation in Chorin's projection approach to solve the two-phase incompressible Navier-Stokes equations. In the first part of this dissertation, these problems are solved with a Jacobi-preconditioned conjugate gradient (CG) solver with up to $N_\mathcal{D}$ iterations and sparse matrix-vector products at $O(N_{\mathcal{D}})$ operations. It is well-known that multigrid methods [TS01] allow to achieve optimal convergence rates independent of the mesh width of the underlying discretization. By introducing a multigrid method to solve the respective elliptic problems, it is thus expected to get a complexity of $O(N_{\mathcal{D}})$ operations for the iterative solvers promising $O(N_\Gamma N_{\mathcal{D}} N_{\mathcal{T}})$ or $O(N_\Gamma N_{\mathcal{D}})$ operations for the solution all realizations of a given random PDE problem.

Therefore, a GPU-parallel multigrid method for elliptic problems will be implemented. While standard geometric multigrid methods often struggle with discretizations in complex geometries, which are e.g. necessary for the two-phase flow solver, the *algebraic multigrid method* (AMG) is able to handle this special case. In contrast to geometric multigrid methods, algebraic multigrid methods build the multigrid hierarchy by a purely algebraic construction involving only the entries of the underlying matrix. Probably the first implementation of AMG on GPU was [PLW$^+$07, HLDP10]. (Smoothed) aggregation type AMGs have also been implemented for GPUs before, because of their rather simple algorithmic structure, cf. [BDO12, Vra12, EL12, BCHZ13, Luk14]. Another approach is given in [WHCX13]. However, the robust classical Ruge-Stüben AMG [TS01, Appendix A] is hard to parallelize on GPUs, due to the intrinsically sequential nature of the classification process between fine grid and coarse grid points in the original algorithm. Nevertheless, two commercial, non-free implementations with limited public access exist, namely GAMPACK [ENS12] and AmgX [Cor]. They use the PMIS (parallel maximum independent set) classification method [Yan06] requiring time-consuming long-range interpolation. This approach is very parallel, but not always robust, which can be overcome by investing a higher amount of computational work.

The objective of this thesis is to implement a robust *classical* AMG with only the coarse/fine grid classification on CPU and the remaining part on GPU. It will be integrated in the existing GPU linear iterative solver framework CUSP [BG12]. In previous published work on classical AMG on GPU [KF12], only the application of the preconditioner was parallelized on GPU and also used in a multi-GPU setting. Here, in difference, also the construction of interpolation and restriction as well as smoothers is done on GPU, leaving only the coarse/fine grid point classification on CPU. The new AMG solver runs on a single GPU together with one CPU core. It will be shown for some cases that this hybrid approach outperforms the open parallel CPU algebraic multigrid implementation BoomerAMG [HY02], with a performance improvement of up to 50 percent, on almost equally priced hardware, i.e. comparing one GPU to 8/12 CPU cores. Some further performance comparison indicators are given with regard to the commercial GPU AMG implementation GAMPACK [ENS12], which show impressive speedup factors of up to 2.5 on almost equally priced hardware. A comparison towards the parallel GPU AMG implementation AmgX [Cor] is hard, since technical reports are missing and reporting results from the test version is not allowed to the authors knowledge and at the time of writing this thesis. In addition to the very promising performance comparison towards a CPU implementation, the method is also faster by a factor of two for an elliptic random PDE uncertainty quantification problem, comparing to the previous GPU-based Jacobi-preconditioned CG solver. It will thus be possible to get an optimal complexity method for the PDE solver part, which is also faster and runs on GPUs. This concludes the second part of this thesis solving all but one problem. The remaining problem is the curse of dimensionality.

Approximation and empirical analysis of fast decaying random PDEs

We learned before that we have to solve N_Γ deterministic PDE problem realizations. The RBF kernel-based stochastic collocation is able to do this with algebraic to exponential convergence. Standard error estimates in kernel approximation [Wen04] formulate such convergence rates in terms of the *fill distance* $h_{X_\Gamma,\Gamma}$, which replaces the standard mesh width in meshless methods. It is well known [Fas07, Chapter 14] that we have the relationship

$$N_\Gamma \sim \left(\frac{1}{h_{X_\Gamma,\Gamma}}\right)^{N_{KL}} ,$$

thus the number of collocation points to approximate e.g. the stochastic space $\Gamma \subset \mathbb{R}^{N_{KL}}$ with the same mesh width grows exponentially in the dimension of that space. This is the *curse of dimensionality*, as introduced before.

In the third part of this thesis, the curse of dimensionality will be weakened or broken for problems with fast decaying spectrum of the random PDE solution covariance function. Indeed, such problems are often represented in stochastic space by many stochastic dimensions N_{KL} whereas only few of these dimensions are dominant for a small overall error. The currently existing standard approach in stochastic collocation to break the curse of dimensionality in presence of a few dominant stochastic dimensions, thus a fast decay of the solution's covariance spectrum, is the anisotropic sparse grid stochastic collocation [NTW08a] with extensions to dimension-adaptive methods in [JR13, ZLY$^+$13, NJ14, SL14, KHRVed].

However, a suitable analogous technique maintaining all good convergence properties of the RBF kernel-based approximation does not exist. Therefore, *anisotropic RBF stochastic collocation* with Matérn and Gaussian kernels will be introduced to approximate stochastic moments at almost *constant* convergence rates in the number of collocation points while increasing the stochastic dimension. This weakens or even breaks the curse of dimensionality. Two important constructions lead to this numerical result. The first is an empirical study of the covariance spectrum of the solution of a random PDE. The second is the *anisotropic RBF kernel* construction with *optimal sampling* in the associated anisotropic native space.

A-posteriori Karhunen-Loève convariance spectrum decay analysis For some model problems, such as elliptic random PDEs or groundwater flows, it is possible to give information on the relationship between the covariance spectrum of the *random input* and the covariance spectrum of the *random PDE solution*, cf. [NTW08a, BNT10, BNTT14]. In those cases, a fast decaying input covariance eigenmode structure often leads to a fast decay of the spectrum of the solution or output covariance. Let us note here that the number of large eigenmodes in the output covariance spectrum is closely related to the number of important stochastic dimensions. With knowledge on the covariance input-output relationship, it is possible to give *a-priori estimates* for the importance of a stochastic dimension. However, the lack of solution theory for the three-dimensional two-phase incompressible Navier-Stokes equations in strong formulation makes this theoretical analysis hard or even impossible. Therefore, a kernel-based method to analyze the input-output behavior of covariance fields for the random two-phase incompressible Navier-Stokes equations is proposed. It will be a purely snapshot based numerical approach to measure the solution covariance spectrum for a given random input. This is done by approximating what will be called the *a-posteriori* or *output Karhunen-Loève expansion.*

Let us assume a known covariance spectrum of the stochastic field $\boldsymbol{a}(\boldsymbol{\omega}, \boldsymbol{x})$, which describes input parameters of a stationary stochastic parameter PDE problem. Therefore, we know the truncated Karhunen-Loève expansion

$$\boldsymbol{a}(\omega, \boldsymbol{x}) \approx \mathbb{E}\left[\boldsymbol{a}\right](\boldsymbol{x}) + \sum_{k=1}^{N_{KL}^{\boldsymbol{a}}} \sqrt{\lambda_k^{\boldsymbol{a}}} Y_k^{\boldsymbol{a}}(\boldsymbol{\omega}) \psi_k^{\boldsymbol{a}}(\boldsymbol{x})$$

with $(\lambda_k^{\boldsymbol{a}}, \psi_k^{\boldsymbol{a}})$ the eigenpairs of a Fredholm integral equation of second kind with

$$\int_{\mathcal{D}} \operatorname{Cov}\left[\boldsymbol{a}\right](\boldsymbol{x}, \boldsymbol{x}') \psi_k^{\boldsymbol{a}}(\boldsymbol{x}') d\boldsymbol{x}' = \lambda_k^{\boldsymbol{a}} \psi_k^{\boldsymbol{a}}(\boldsymbol{x}) \,. \tag{1.4}$$

Then, the truncated a-posteriori Karhunen-Loève expansion is

$$\boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}) \approx \mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x}) + \sum_{k=1}^{N_{KL}^{\boldsymbol{u}}} \sqrt{\lambda_k^{\boldsymbol{u}}} Y_k^{\boldsymbol{u}}(\boldsymbol{y}) \psi_k^{\boldsymbol{u}}(\boldsymbol{x}) \,.$$

with similar definitions for the eigenpairs $(\lambda_k^{\boldsymbol{u}}, \psi_k^{\boldsymbol{u}})$. It contains all information of the covariance structure of the solution random field. Recent results on the approximation of Karhunen-Loève expansions are [ST06] with a fast multipole expansion, [EEU07] with $\mathcal{H}$-matrices and [HPS14] with low-rank approximations by the Pivoted Cholesky Decomposition. However, to the authors knowledge, this has not been done for two-phase flow problems, so far. Furthermore, a (multi-)GPU based approach is missing and the approximation of stochastic moments by RBF kernel-based stochsastic collocation is new, anyway.

In this thesis, a Nyström discretization [Hac95] of the continuous solution covariance operator is used to approximate the output Karhunen-Loève expansion based on moment approximations by the RBF kernel-based stochastic collocation method. This leads to a large-scale eigenvalue problem with *dense* square matrices in the range of tens to hundreds of thousands rows (thus the number of grid points of the PDE problem). As e.g. proposed in [Kie08], this problem will be efficiently solved by a Lanczos iterative method [Saa03]. In contrast to a direct eigenvalue solver, this dense iterative approach reduces the computational time to compute the eigenmodes to a very small fraction. It is further parallelized in a highly scalable way on multiple GPUs achieving best possible performance. Remember that it is thus possible to approximate the solution covariance spectrum even in those cases in which no theory is known. Note that the output Karhunen-Loève expansion can be also used for model reduction. In fact, its truncated versions are best $N_{KL}^{\boldsymbol{u}}$-term approximations to the original continuous solution field in the underlying stochastic space [ST06].

Anisotropic RBF stochastic collocation The a-posteriori Karhunen-Loève covariance decay analysis is used as a tool to introduce anisotropic RBF stochastic collocation. This will give an alternative to anisotropic full tensor-product or sparse grid constructions. Anisotropic RBF stochastic collocation will follow ideas on anisotropic RBF interpolation [CMM07, BDL10, FHW12, GLS13] and Gaussian process regression [RW05, Section 5.1] replacing the RBF kernels $k(\boldsymbol{y}, \boldsymbol{y}') := \varphi(\|\boldsymbol{y} - \boldsymbol{y}'\|_2)$ by anisotropic kernels with a weighted norm $\|\cdot\|_\gamma$ as

$$k^{\gamma}(\boldsymbol{y},\boldsymbol{y}') = \varphi(\|\boldsymbol{y}-\boldsymbol{y}'\|_{\gamma}), \quad \text{with} \quad \|\boldsymbol{y}\|_{\gamma} := \left(\boldsymbol{y}^{\top}\mathrm{diag}(\boldsymbol{\gamma})^2\boldsymbol{y}\right)^{1/2}, \quad \boldsymbol{\gamma} := (\gamma_1,\ldots,\gamma_{N_{FN}})^{\top}.$$

Vector $\boldsymbol{\gamma}$ will encode the anisotropy or importance of different stochastic dimensions. In fact, it will be motivated to choose $\gamma_i = \sqrt{\lambda_i^{\boldsymbol{u}}}$ or to use a-priory estimates for $\boldsymbol{\gamma}$ as in [NTW08a]. This construction leads to a new stochastic collocation approximation with

$$\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t) \approx (\mathcal{I}_{N_\Gamma}^{\gamma}\boldsymbol{u})(\boldsymbol{y},\boldsymbol{x},t) := \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)L_i^{\gamma}(\boldsymbol{y})$$

that approximates in the weighted kernel native space $\mathcal{N}_k^{\gamma}$ such that we get

$$\mathcal{I}_{N_\Gamma}^{\gamma}\boldsymbol{u} \in \mathcal{N}_k^{\gamma}(\Gamma) \otimes L^2([0,T];L^2(\mathcal{D})).$$

While the classical sampling method for radial basis functions uses very regular collocation point sets stemming e.g. from a quasi-Monte Carlo Halton sequence, optimal point sets for anisotropic kernel-based stochastic collocation are unknown. To overcome this problem, a greedy adaptive method for good point sets in anisotropic weighted native spaces is introduced. It uses the algorithm proposed in [DMSW05]. This minimizes the power function

$$[P_{k,X}(\boldsymbol{y})]^2 := Q[\boldsymbol{L}(\boldsymbol{y})](\boldsymbol{y}), \quad Q[\boldsymbol{a}](\boldsymbol{y}) = k(\boldsymbol{y},\boldsymbol{y}) - 2\sum_{j=1}^{N} a_j k(\boldsymbol{y},\boldsymbol{y}_j) + \sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(\boldsymbol{y}_i,\boldsymbol{y}_j)$$

for which we know

$$|f(\boldsymbol{y}) - s_{f,X}(\boldsymbol{y})| \le P_{k,X}(\boldsymbol{y})\|f\|_{\mathcal{N}_k(\Gamma)}$$

for some classes of functions. By that way, a purely approximation space-dependent error reducing point set is generated. It will turn out that such an almost optimal point set comes close to an anisotropic tensor product grid [NTW08a] with arbitrary numbers of points, in case of anisotropic Matérn kernels. Note that there are close connections between this optimal point sampling with its generalized version in [Sch13], Adaptive Cross Approximation [Beb11, BMS14], Leja and Fekete sequences [NJ14] and low-rank approximations by e.g. the pivoted Cholesky decomposition [HPS12].

Optimal point sets will be precomputed in Matlab and used in a new anisotropic RBF stochastic collocation GPU implementation. The approach is first tested with an elliptic model problem and weights known from literature [NTW08a]. In the final two-phase bubble flow application problem, weights are derived by an a-posteriori Karhunen-Loève analysis for a low-resolution simulation of the same problem. Most numerical results will give hints towards *dimension-independent* convergence rates for fixed fast-decaying covariance structure problems, with clear error improvements in the pre-asymptotic regime. In some way, this work complements non-constructive results for dimension-independent convergence for anisotropic Gaussian kernels [FHW12]. Overall, the last part of this thesis gives hints towards the solution of the remaining problem of weakening or even *breaking the curse of dimensionality.*

Summary of own contributions

Numerical methods

- The RBF kernel-based stochastic collocation method is introduced. It is a framework to solve random PDEs at high- to exponential error convergence order with excellent pre-asymptotic error behavior, outperforming Monte Carlo and (sparse) spectral tensor-product approximations in many important application cases.
- For the first time, stochastic collocation is applied to two-phase flow problems modeled by the two-phase incompressible Navier-Stokes equations.
- An empirical error coupling analysis for this framework is outlined, giving indicators for optimal error balancing between the different numerical approximations.
- Optimal computational complexity is achieved for the kernel based stochastic collocation by the introduction of an local Lagrange basis preconditioner for kernel approximation. It has an intrinsic perfect parallel scalability and optimal properties for Exascale computing. This allows to solve stochastic collocation problems in the range of hundreds of thousands to potentially millions of collocation points.
- All discussed deterministic realizations of random PDE problems can be solved at optimal computational complexity by a classical Ruge-Stüben algebraic multigrid method.
- A Nyström-discretization based approximation of the a-posteriori Karhunen-Loève expansion, using kernel-based moment approximation, is introduced. This allows, for the first time, to empirically analyze the solution covariance structure of the random two-phase Navier-Stokes equations, for which no theory is known.
- The new asymptotic RBF kernel-based stochastic collocation approach is presented. It is empirically shown that it weakens or breaks the curse of dimensionality for random PDE problems with fast decay in the covariance spectrum of the random solution field. Lower pre-asymptotic error is clearly achieved.
- A large number of numerical results for model problems and real-world applications are given.

Software

- (Multi-)GPU based libraries for stochastic space sampling, higher-dimensional quadrature, RBF kernel-based approximation and the new RBF kernel-based stochastic collocation are implemented with optimal runtime complexity and low preasymptotic runtime.
- Dense iterative linear solvers are implemented in the stand-alone multi-GPU parallel library *parla* with almost optimal parallel scalability.
- All core components of the originally MPI-parallel CPU-based solver NaSt3DGPF for the two-phase Navier-Stokes equations are parallelized to run on a multi-GPU cluster with profound speedups and almost excellent weak scalability. This is the first multi-GPU solver of this kind.

- A multi-GPU parallel preconditioner for dense linear systems from kernel interpolation with excellent strong scaling is implemented, resulting in the first iterative multi-GPU solver with local Lagrange preconditioning for kernel problems.

- The classical algebraic multigrid method is implemented and parallelized in a hybrid GPU version, including major parts of the *setup phase* on GPU delivering up to a 50 percent speedup over optimal multi-core CPU AMG implementations on almost equally priced hardware and a two-fold speedup over naive GPU Jacobi-preconditioned solvers. This is the first non-commercial code of this kind.

- Based on *parla*, the first multi-GPU iterative method for large-scale dense eigenvalue problems with algorithms to compute the a-posteriori Karhunen-Loève expansion is developed.

- The isotropic RBF kernel-based approximation and isotropic stochastic collocation library is extended to support the new anisotropic RBF approximation and new anisotropic stochastic collocation in a (multi-)GPU fashion.

Outline

The remainder of this thesis is organized as follows. In the first part, the basic RBF kernel-based stochastic collocation method with applications to two-phase flows is discussed. Therein, Chapter 2 summarizes basic concepts of uncertainty quantification. Chapter 3 discusses the different model and application problems. Thereafter, Chapter 4 introduces the RBF kernel-based stochastic collocation method. Some discretization details and the multi-GPU parallelization with numerical results for the two-phase Navier-Stokes solver are given in Chapter 5. Chapter 6 concludes the first part of this thesis by numerical studies and empirical error coupling results. In the second part, parallel preconditioning techniques are discussed with local preconditioning for kernel approximation in Chapter 7 and algebraic multigrid in Chapter 8. The last part is dedicated to the objective to break the curse of dimensionality in RBF-based stochastic collocation. Here, Chapter 9 describes the numerical a-posteriori Karhunen-Loève expansion analysis and Chapter 10 introduces the anisotropic RBF stochastic collocation with numerical results. This thesis is concluded in Chapter 11 with a short summary and outlook.

Acknowledgements

At this point, I would like to thank all the people that have contributed to the completion of this thesis. First of all, I would like to express my gratitude to my advisor Prof. Dr. Michael Griebel for introducing me to this topic. His important ideas and many inspiring discussions as well as the provided excellent working conditions were the important foundations for this work. Furthermore, I would like to thank Prof. Dr. Mario Bebendorf for providing his opinion as the second referee.

Sincere thanks go to all my colleagues at the Institute for Numerical Simulations (INS) for the nice atmosphere throughout the years. I am especially grateful to Dr. Christian Rieger for endless hours of fruitful discussions and countless inspirations. Also, I thank Markus Burkow, Patrick Diehl, Dr. Chrisian Rieger, Alexander Rüttgers and Ralph Thesen for proofreading parts of this thesis. All the computational work could only be done because of the enormous efforts of those, who are responsible for the IT at the INS, Dr. Bram Metsch and Ralph Thesen and more recently Markus Burkow and Patrick Diehl and all the great students.

Major parts of the numerical results were generated on a GPU cluster of the institute SCAI which is part of the Fraunhofer Society, while some of the two-phase flow solver scalability studies were done on a GPU cluster of the RWTH Aachen University via the Ressourcenverbund – Nordrhein-Westfalen (RV-NRW). This support is grateful acknowledged.

On a personal level, I will always be thankful for the patience, encouragement and unconditional understanding of my parents and siblings.

Part I

RBF kernel-based stochastic collocation for two-phase flows

2 Uncertainty quantification

In this chapter, the mathematical framework to model and formulate *uncertainty quantification* problems is considered. Solving these problems is the main objective of this thesis. Therefore, this chapter forms the base of this thesis and also prepares for the subsequent chapter of model and application problems.

Uncertainty quantification will be introduced as the evaluation of stochastic moments of (derived quantities from) solutions of *random PDE problems*. Throughout this thesis, random PDE problems are understood as partial differential equations which are parametrized by stochastic parameters. This parametrization is not restricted to random coefficients. General influences on initial and boundary conditions, forcing terms and the domain shape are also possible. To achieve a maximum of flexibility, generic operator-type equations shall be used to introduce random PDE problems. It will turn out that a technical assumption has to be made to allow a numerical treatment of the described problems. This is the *finite noise assumption* which can, e.g., be fulfilled by a truncated Karhunen-Loève expansion.

The chapter starts with a short review of some probability theory definitions and results. Thereafter, general random PDE problems are derived from deterministic parametric problems. It follows the introduction of the finite noise assumption by the Karhunen-Loève expansion or piecewise constant random fields leading to a finite-dimensional reformulation of the random PDE problem. Finally, basic terms in uncertainty quantification are discussed.

2.1 Primer on probability theory

In the following, to fix notation and to give a quick reminder, some basics in probability theory are shortly reviewed. This overview is an excerpt from [LMK10, Appendix A] and [Gri02, Chapter 2] with notation similar to [BNT10, NTW08b]. Note that some definitions contain statements, which would require a proof. These can be found e.g. in [Gri02, Chapter 2] and are not reproduced. The same holds for the proofs for all lemmata in this section.

Our discussion starts with the *sample space* Ω, which is the set of possible outcomes of a random experiment. Its elements are often called ω. In the target application Ω is uncountable. We can then introduce *σ-fields* by

Definition 2.1 (σ-field [Gri02, Section 2.2.2][LMK10, Appendix A.1.1])**.** *Shall be $\mathcal{F}$ a non-empty set of subsets of Ω. Then, $\mathcal{F}$ is called* σ-field*, if*

1. $\emptyset \in \mathcal{F}$,

2. $A \in \mathcal{F} \Rightarrow A^c \in \mathcal{F}$,

3. $A_i \in \mathcal{F}, i \in I, I$ *countable* $\Rightarrow \bigcup_{i\in I} A_i \in \mathcal{F}$,

with A^c the complement of A.

A special class of σ-fields are *Borel σ-fields*. They are generated by all open sets of a topological set. Furthermore, $\mathcal{B}(\mathbb{R}^d) = \mathcal{B}^d$ is the Borel σ-field on $\mathbb{R}^d$. In the special case of $d = 1$, we have $\mathcal{B} = \mathcal{B}^1 = \mathcal{B}(\mathbb{R})$ and $\mathcal{B}$ is the set of open intervals. We can now introduce *measurable spaces*, *measures* and *measure spaces* in two definitions.

Definition 2.2 (Measurable space [Gri02, Section 2.2.2][LMK10, Appendix A.1.1])**.** *With $\mathcal{F}$ a σ-field, elements $A \in \mathcal{F}$ are called* event *or* $\mathcal{F}$-measurable *and* $(\Omega, \mathcal{F})$ *is a* measurable space.

Definition 2.3 (Measures and measure spaces [Gri02, Section 2.2.3][LMK10, Appendix A.1.2])**.** *A set function $\mu : \mathcal{F} \to [0, \infty]$ is called* measure *on $\mathcal{F}$ if*

$$\mu\left(\bigcup_{i=1}^{\infty} A_i\right) = \sum_{i=1}^{\infty} \mu(A_i), \quad A_i \in \mathcal{F}, A_i \cap A_{i \neq j} = \emptyset\,,$$

thus it is countably additive. Then $(\Omega, \mathcal{F}, \mu)$ is called measure space. *A* finite measure *has to fulfill $\mu(\Omega) < \infty$.*

Of important nature for this thesis are *probability spaces* given by

Definition 2.4 (Probability space [Gri02, Section 2.2.3][LMK10, Appendix A.1.3])**.** *A* probability measure *or* probability *is a finite measure P with*

1. $P : \mathcal{F} \to [0, 1]$
2. $P(\Omega) = 1$

With the beforehand definitions, the triple $(\Omega, \mathcal{F}, P)$ is a probability space. *Furthermore, it is* complete *if for all $A \subset B$ with $B \in \mathcal{F}$ and $P(B) = 0$, it holds $A \in \mathcal{F}$ which fulfills $P(A) = 0$. In the following, all probability spaces are expected to be complete.*

Lemma 2.1 (Properties of probability spaces [Gri02, Section 2.2.3][LMK10, Appendix A.1.3])**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space. For $A, B \in \mathcal{F}$, it holds that*

1. $P(A) \leq P(B), \quad$ *if* $A \subseteq B$,
2. $P(A) = 1 - P(A^c)$,
3. $P(A \cup B) = P(A) + P(B) - P(A \cap B)$.

Next, we need to comment on functions and *induced probabilities* on measurable spaces which eventually allow to introduce the important notions of *random variables* and *random vectors*.

Definition 2.5 (Measurable function [Gri02, Section 2.4][LMK10, Appendix A.2.1])**.** *For two measurable spaces $(\Omega, \mathcal{F})$ and $(\Psi, \mathcal{G})$, a function $h : \Omega \to \Psi$ is called* measurable from $(\Omega, \mathcal{F})$ to $(\Psi, \mathcal{G})$ *if it holds*

$$h^{-1}(B) = \{\omega : h(\omega) \in B\} \in \mathcal{F}, \quad \forall B \in \mathcal{G}\,.$$

Definition 2.6 (Induced probability [Gri02, Section 2.4.1][LMK10, Appendix A.2.1])**.** *Let $h : \Omega \to \Psi$ be measurable from $(\Omega, \mathcal{F})$ to $(\Psi, \mathcal{G})$ and furthermore a probability P is given such that we have a probability space $(\Omega, \mathcal{F}, P)$, then the function $Q : \mathcal{G} \to [0, 1]$ with*

$$Q(B) := P(h^{-1}(B)), \quad \forall B \in \mathcal{G}\,,$$

is a probability measure on $(\psi, \mathcal{G})$ *which is called* probability induced by h *or* distribution of h*.*

Definition 2.7 (Random variables and vectors [Gri02, Section 2.4.2][LMK10, Appendix A.2.2])**.** *With* $(\Omega, \mathcal{F}, P)$ *a probability space, a function* $X : \Omega \to \mathbb{R}$*, which is measurable from* $(\Omega, \mathcal{F})$ *to* $(\mathbb{R}, \mathcal{B})$*, is called* $\mathbb{R}$-valued random variable *or simply* random variable*. The* $\mathbb{R}^d$*-valued function* $\boldsymbol{X} : \Omega \to \mathbb{R}^d$ *being measurable from* $(\Omega, \mathcal{F})$ *to* $(\mathbb{R}^d, \mathcal{B}^d)$ *is a* random vector*, if all its coordinates are random variables.*

The $\mathbb{R}$-valued random variable $X : \Omega \to \mathbb{R}$ is sometimes also denoted as $X(\omega)$. We can of course also define the induced probability of a random vector $\boldsymbol{X}$ by

Definition 2.8 (Distribution [Gri02, Section 2.4.2][LMK10, Appendix A.2.2])**.** *The* distribution *of a random vector* $\boldsymbol{X}$*, given as before, is its induced probability, which is defined by*

$$Q(B) := P(\boldsymbol{X}^{-1}(B)), \quad \forall B \in \mathcal{B}^d .$$

To define the expectation operator, the concept of integrals and integrability is necessary. Avoiding an in-depth discussion of this wide topic, we restrict ourselves to the following simple definition.

Definition 2.9 (Integrability [Gri02, Section 2.5.1][LMK10, Appendix A.3.1])**.** *Let* $(\Omega, \mathcal{F}, P)$ *be a probability space and* X *a random variable measurable from* $(\Omega, \mathcal{F})$ *to* $(\mathbb{R}, \mathcal{B})$*, the integral of* X *with respect to* P *over the event* $A \in \mathcal{F}$ *is*

$$\int_\Omega I_A(\omega) X(\omega) dP(\omega) = \int_A X(\omega) dP(\omega) ,$$

with I_A *the classic indicator function for* A*.* X *is called* P-integrable over A*, if the integral exists and is finite.*

This definition can be extended to random vectors $\boldsymbol{X}$ requiring that all components are individually P-integrable over A. Following the lines of [LMK10], it can be then stated that for $A = \Omega$ the above integral corresponds to the *expectation operator* $E[\cdot]$, thus it holds

$$\mathbb{E}[X] = \int_\Omega X(\omega) dP(\omega) .$$

We collect properties of this operator in

Lemma 2.2 (Properties of the expectation operator [Gri02, Section 2.5.2][LMK10, Appendix A.3.2])**.** *For* X, Y $\mathbb{R}$*-valued random variables defined on the probability space* $(\Omega, \mathcal{F}, P)$ *and* P*-integrable over* Ω *and assuming that the expectation operator exists and is finite, we have*

1. $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y], \quad \forall a, b \in \mathbb{R}$ *(linearity),*
2. *if* $X \geq 0$ *holds almost surely, then* $\mathbb{E}[X] \geq 0$*,*
3. *if* $Y \leq X$ *holds almost surely, then* $\mathbb{E}[Y] \leq \mathbb{E}[X]$*,*
4. $|\mathbb{E}[X]| \leq \mathbb{E}[|X|]$*.*

Note that a statement holds *almost surely*, if it has probability one. Now, it is possible to introduce function spaces for random variables.

Definition 2.10 ($L_q(\Omega, \mathcal{F}, P)$ spaces [Gri02, Section 2.6][LMK10, Appendix A.3.3])**.** *With* $(\Omega, \mathcal{F}, P)$ *a probability space, the* $L_q(\Omega, \mathcal{F}, P)$ space*,* $q \geq 1$*, is the set of random variables* X *on that probability space, for which holds*

$$\mathbb{E}\left[|X|^q\right] < \infty \,.$$

Those elements X *are then denoted by* $X \sim L_q(\Omega, \mathcal{F}, P)$ *and we have the abbreviation* L_q *for* $L_q(\Omega, \mathcal{F}, P)$*.*

Lemma 2.3 (L_2 space [Gri02, Section 2.6][LMK10, Appendix A.3.3])**.** *The* $L_2(\Omega, \mathcal{F}, P)$ *space or short* L_2 *space is a vector space. With the inner product* $\langle X, Y\rangle = \mathbb{E}\left[XY\right]$ *for elements* $X, Y \in L_2$*,* L_2 *becomes a Hilbert space with norm* $\|X\|_{L_2} = \left(\mathbb{E}\left[X^2\right]\right)^{1/2}$*.*

Another important concept are *distribution functions* for which the definition and some properties are collected.

Definition 2.11 (Distribution function [Gri02, Section 2.10.1][LMK10, Appendix A.4.1])**.** *Let* X *be a random variable* $X : \Omega \to \mathbb{R}$ *as before. Its* (cumulative) distribution function *is given as*

$$F(x) := P(X^{-1}((-\infty, x])) = P(\{\omega : X(\omega) \leq x\}) = P(X \leq x)\,.$$

Lemma 2.4 (Properties of distribution functions [Gri02, Section 2.10.1][LMK10, Appendix A.4.1])**.** *For* X *a random variable and* F *its distribution function, we have*

1. F *is right-continuous, increasing and has the range* $[0, 1]$*,*
2. F *can have a countable number of jump discontinuities,*
3. $\lim_{x\to\infty} F(x) = 1$ *and* $\lim_{x\to-\infty} F(x) = 0$*,*
4. $P(a < X \leq b) = F(b) - F(a) \geq 0$ *for* $a \leq b$*,*
5. $P(a \leq X < b) = F(b) - F(a) + P(X = a) - P(X = b)$ *for* $a \leq b$*.*

From distribution functions, we can derive *density functions* by

Definition 2.12 (Density function [Gri02, Section 2.10.2][LMK10, Appendix A.4.2])**.** *With the additional requirement to the distribution function* F *to be absolutely continuous in* $\mathbb{R}$*, there is an integrable function* ρ*, for which holds*

$$F(b) - F(a) = \int_a^b \rho(x)dx, \quad a \leq b\,.$$

That function ρ *is called* (probability) density function *of* X*.*

Lemma 2.5 (Properties of density functions [Gri02, Section 2.10.2][LMK10, Appendix A.4.2])**.** *A density function* ρ*, as before, has the properties*

1. *$\rho(x) = F'(x)$ so that $\int_{-\infty}^{x} \rho(y)dy = F(x)$,*
2. *$\rho \geq 0$,*
3. *ρ is no probability measure,*
4. *$\int_{-\infty}^{\infty} \rho(x)dx = 1$.*

For $(\Omega, \mathcal{F}, P)$ a probability space and $X : \Omega \to \mathbb{R}$ a random variable on that space, we have some well-known density functions with the

- *normal or Gaussian density function: $\rho_{\mu,\sigma^2}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$* and the
- *uniform density function: $\rho_{a,b} = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & \text{otherwise} \end{cases}$.*

Random variables X with Gaussian or normal density functions are called Gaussian or normally distributed random variables and are denoted by $X \sim \mathcal{N}(\mu, \sigma^2)$. Moreover, random variables X with uniform density function are called uniformly distributed random variables and are denoted by $X \sim \mathcal{U}(a,b)$.

For practical computations, it is desirable to express the evaluation of the expectation operator of a random variable as an integral over $\mathbb{R}$. This can be realized by introducing *measurable transformations.*

Definition 2.13 (Measurable transformations [Gri02, Section 2.4.3][LMK10, Appendix A.2.3])**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space with a random vector $\boldsymbol{X}$, which is a measurable function from $(\Omega, \mathcal{F})$ to $(\mathbb{R}^d, \mathcal{B}^d)$. We call a measurable function g from $(\mathbb{R}^d, \mathcal{B}^d)$ to $(\mathbb{R}, \mathcal{B})$ a* transformation *on the random variable.*

It is possible to show that the composed mapping $Y = g \circ \boldsymbol{X}$ is a random variable which is measurable from (Ω, σ) to $(\mathbb{R}, \mathcal{B})$, cf. [Gri02, Section 2.4.3],[LMK10, Appendix A.2.3]. This allows to formulate

Lemma 2.6 (Expectation of transformed random variables [Gri02, Section 2.10.2][LMK10, Appendix A.4.2])**.** *With the random vector $\boldsymbol{X}$ for $d = 1$, thus the random variable X, and the composed mapping $Y = g \circ \boldsymbol{X}$ both integrable and further $Q(B) = P(X^{-1}(B)), B \in \mathcal{B}$ and F the density function of X, we have*

$$\mathbb{E}[Y] = \int_{\Omega} Y(\omega)dP(\omega) = \int_{\Omega} g(X(\omega))dP(\omega) \textit{ and}$$
$$\mathbb{E}[Y] = \int_{\mathbb{R}} g(x)Q(dx) = \int_{\mathbb{R}} g(x)dF(x) = \int_{\mathbb{R}} g(x)\rho(x)dx\,.$$

Distribution and density functions can also be generalized to random vectors in

Definition 2.14 (Joint distribution and density functions [Gri02, Section 2.11.1][LMK10, Appendix A.5.1])**.** *For random vectors $\boldsymbol{X}$, the* joint distribution function $F : \mathbb{R}^d \to [0,1]$ *is given by*

$$F(\boldsymbol{x}) = P\left(\cap_{i=1}^{d}\{X_i \leq x_i\}\right)$$

with $\boldsymbol{x} = (x_1, \ldots, x_d) \in \mathbb{R}^d$. *If it exists, the* joint density function $\rho : \mathbb{R}^d \to [0,1]$ *is given by*

$$\rho(\boldsymbol{x}) := \frac{\partial^d F(\boldsymbol{x})}{\partial x_1 \cdots \partial x_d} .$$

In many cases, *independence* of random variables allows to simplify the underlying computations. Therefore, we give

Definition 2.15 (Independence of random variables [Gri02, Sec. 2.7.1, Sec. 2.11.2][LMK10, Appendix A.5.2])**.** *Given a probability space* $(\Omega, \mathcal{F}, P)$ *and a collection of sub-*σ*-fields* $\mathcal{F}_i$, $i \in I$ *of* $\mathcal{F}$*, the sub-fields are called independent if*

$$P\left(\cap_{i \in I} A_i\right) = \prod_{i \in I} P(A_i), \quad \forall A_i \in \mathcal{F}_i \, ,$$

in case of I *being finite. For infinite* I*, the above equation has to hold for all finite subsets of* I*. Having a family of random variables* X_i*,* $i \in I$ *from the probability space with* I *finite or infinite, the* X_i *are* independent random variables*, if the sigma-fields* $\sigma(X_i)$*, generated by the random variables, are independent.*

Lemma 2.7 (Independence of random variables [Gri02, Sec. 2.7.1, Sec. 2.11.2][LMK10, Appendix A.5.2])**.** *The above family of random variables* X_i *is independent if and only if for all* $J \subset I$ *finite, it holds*

$$P(X_i \leq x_i, i \in J) = \prod_{i \in J} P(X_i \leq x_i), \quad x_i \in \mathbb{R} \, .$$

The desired quantities that shall be computed throughout this thesis are *moments.* We start with their definition for random vectors.

Definition 2.16 (Moments of random vectors[Gri02, Section 2.11.4][LMK10, Appendix A.5.3])**.** *Let the probability space* $(\Omega, \mathcal{F}, P)$ *be given. The random vector* $\boldsymbol{X}$ *shall be defined on that space. Furthermore, the function* $g(\boldsymbol{x}) = \prod_{i=1}^d x_i^{s_i}$ *with* $s_i \geq 0$ *integers shall be given. Then* $g(\boldsymbol{X})$ *is a real-valued random variable. With the additional requirement of* $\boldsymbol{X} \sim L_s$*, thus for all* $i = 1, \ldots, d$ *we have* $X_i \sim L_s$*, it is possible to define the* moments of order s of X*,* $s = \sum_{i=1}^d s_i$ *by*

$$\mu(s_1, \ldots, s_d) = \mathbb{E}\left[g(\boldsymbol{X})\right] = \mathbb{E}\left[\prod_{i=1}^d X_i^{s_i}\right] .$$

They exist and are finite. Corresponding choices of the s_i *lead to the well-known moments*

- Mean *of* X_i*:* $\mu_i = \mathbb{E}\left[X_i\right]$,
- Correlation *of* (X_i, X_j)*:* $r_{i,j} = \mathbb{E}\left[X_i X_j\right]$,
- Covariance *of* (X_i, X_j)*:* $c_{i,j} = \mathbb{E}\left[(X_i - \mu_i)(X_j - \mu_j)\right] = r_{i,j} - \mu_i \mu_j$,
- Variance *of* X_i*:* $\sigma_i^2 = c_{i,i} = \mathbb{E}\left[(X_i - \mu_i)^2\right] = r_{i,i} - \mu_i^2$.

However, since we often want to analyze space- and time-dependent problems, further generalizations of random variables are necessary. They are given by the following three definitions.

Definition 2.17 (Stochastic process [Gri02, Section 3.2])**.** *With* $(\Omega, \mathcal{F}, P)$ *a probability space,* $\boldsymbol{X} : \Omega \times I \to \mathbb{R}^d$ *a two argument function and* $I \subset \mathbb{R}$ *or* $I = [0, \infty)$*, we call* $\boldsymbol{X}$ *an* $\mathbb{R}^d$*-valued* stochastic process*, if for each* $t \in I$*,* $\boldsymbol{X}(t)$ *is an* $\mathbb{R}^d$*-valued random variable on* $(\Omega, \mathcal{F}, P)$*, thus* $\boldsymbol{X}(t) \in \mathcal{F}$*.*

Definition 2.18 (Random field [Gri02, Section 3.2])**.** *With* $(\Omega, \mathcal{F}, P)$ *a probability space,* $\boldsymbol{X} : \Omega \times \mathcal{D} \to \mathbb{R}^d$ *a two argument function and* $\mathcal{D} \subset \mathbb{R}^q$*,* $q \geq 1$ *integer, we call* $\boldsymbol{X}$ *an* $\mathbb{R}^d$*-valued* random field*, if for each* $\boldsymbol{x} \in \mathcal{D}$*,* $\boldsymbol{X}(\boldsymbol{x})$ *is an* $\mathbb{R}^d$*-valued random variable on* $(\Omega, \mathcal{F}, P)$*, thus* $\boldsymbol{X}(\boldsymbol{x}) \in \mathcal{F}$*.*

Definition 2.19 (Space-time stochastic process [Gri02, Section 3.1])**.** *With* $(\Omega, \mathcal{F}, P)$ *a probability space and* $\boldsymbol{X} : \Omega \times \mathcal{D} \times I \to \mathbb{R}^d$ *a three argument function,* $\mathcal{D} \subset \mathbb{R}^q$ *and* $I \subset \mathbb{R}$ *or* $I = [0, \infty)$*,* $q \geq 1$ *integer, we call* $\boldsymbol{X}$ *an* $\mathbb{R}^d$*-valued* space-time stochastic process*, if for each* $t \in I$*,* $\boldsymbol{X}(\cdot, t)$ *is a random field and for each* $\boldsymbol{x} \in \mathcal{D}$*,* $\boldsymbol{X}(\boldsymbol{x}, \cdot)$ *is a stochastic process.*

Measurabilty extends naturally to these objects by

Definition 2.20 (Measurable stochastic processes and random fields [Gri02, Section 3.2])**.** *Using notation from Definitions 2.17 and 2.18, a stochastic process* $\boldsymbol{X}(t)$ *or random field* $\boldsymbol{X}(\boldsymbol{x})$ *is* measurable *if the function* $\boldsymbol{X} : \Omega \times I \to \mathbb{R}^d$ *or* $\boldsymbol{X} : \Omega \times \mathcal{D} \to \mathbb{R}^d$ *is measurable from* $(\Omega \times I, \mathcal{F} \times \mathcal{B}(I))$ *to* $(\mathbb{R}^d, \mathcal{B}^d)$ *or from* $(\Omega \times \mathcal{D}, \mathcal{F} \times \mathcal{B}(\mathcal{D}))$ *to* $(\mathbb{R}^d, \mathcal{B}^d)$*.*

An analogous definition holds for space-time stochastic processes. Finally, we can introduce moments for stochastic processes and random fields. Their evaluation will be the main objective application in this thesis.

Definition 2.21 (Moments of stochastic processes and random fields [Gri02, Section 3.7])**.** *For* $\boldsymbol{X}$ *an* $\mathbb{R}^d$*-valued stochastic process or random field in* $L_2(\Omega, \mathcal{F}, P)$*, it is possible to define* mean*,* correlation *and* covariance functions $\boldsymbol{\mu}$*,* $\boldsymbol{r}$ *and* $\boldsymbol{c}$ *as*

- $\boldsymbol{\mu}(t) = \mathbb{E}\left[\boldsymbol{X}(t)\right]$ *or* $\boldsymbol{\mu}(\boldsymbol{x}) = \mathbb{E}\left[\boldsymbol{X}(\boldsymbol{x})\right]$*,*
- $\boldsymbol{r}(t, s) = \mathbb{E}\left[\boldsymbol{X}(t)\boldsymbol{X}(s)^\top\right]$ *or* $\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}\left[\boldsymbol{X}(\boldsymbol{x})\boldsymbol{X}(\boldsymbol{x})^\top\right]$*,*
- $\boldsymbol{c}(t, s) = Cov\left[\boldsymbol{X}\right](t, s) = \mathbb{E}\left[(\boldsymbol{X}(t) - \boldsymbol{\mu}(t))(\boldsymbol{X}(s) - \boldsymbol{\mu}(s))^\top\right]$ *or*
  $\boldsymbol{c}(\boldsymbol{x}, \boldsymbol{x}') = Cov\left[\boldsymbol{X}\right](\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}\left[(\boldsymbol{X}(\boldsymbol{x}) - \boldsymbol{\mu}(\boldsymbol{x}))(\boldsymbol{X}(\boldsymbol{x}') - \boldsymbol{\mu}(\boldsymbol{x}'))^\top\right]$*.*

We say that an R^d-valued stochastic process or random field $\boldsymbol{X}$ is in $L^2(\Omega, \mathcal{F}, P)$, if $\boldsymbol{X}(t) \in L^2(\Omega, \mathcal{F}, P)$ $(\forall t)$ or $\boldsymbol{X}(\boldsymbol{x}) \in L^2(\Omega, \mathcal{F}, P)$ $(\forall \boldsymbol{x})$.

Later, we will use the notion of *correlation length.* To exemplify it, we take exponential kernel correlation functions: For some $\mathbb{R}$-valued stochastic processes, the correlation function is given by the exponential kernel [LMK10, Section 2.1.3], thus

$$r(t, s) = \sigma^2 e^{-\frac{|t-s|}{L_c}},$$

with σ^2 the variance. Then, L_c is the so-called *correlation length.* In this context, the correlation length implies that the smaller the correlation length the higher the correlation between close values of t, s. In other words, the correlation length is a measure for the "distance" for which the influence between t, s decreases.

2.2 Random PDE problems

With the beforehand discussed basic terminology in probability theory, it is now the aim to introduce the concept of *random PDE problems*. Starting from very general parametric PDE problems, we will move over to parameter functions, which are random fields or space-time stochastic processes. These ultimately lead to what will be called random PDE problems, observing that the setting of standard random-*coefficient* PDE problems is not powerful enough to describe all model and application problems in Chapter 3. Since this thesis is concerned with several different random PDE problems, the intention is further to introduce a unifying nomenclature for all problems using a rather generalized framework of operators.

2.2.1 Parametrized PDEs

Our starting point are general parametrized initial-boundary value problem PDEs of the form

$$\mathcal{L}(\boldsymbol{a})\boldsymbol{u} = \boldsymbol{f}[\boldsymbol{b}] \qquad \text{in } \bar{\mathcal{D}} \times [0,T] \tag{2.1}$$

with $\mathcal{D} \subset \mathbb{R}^d$, $d \in \mathbb{N}$, the underlying domain, $\boldsymbol{u} : \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^r$ ($r \in \mathbb{N}$, $T \in \mathbb{R}^{\geq 0}$) the space- and time-dependent solution of the PDE problem, $\boldsymbol{a} : \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^{s_1}, \boldsymbol{b} : \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^{s_2}$, $s_1, s_2 \in \mathbb{N}$ some potentially time- and/or space-dependent parameter functions, $\mathcal{L}$ some PDE operator including boundary and initial value handling and $\boldsymbol{f}$ the right-hand sides of the PDE problem, the boundary conditions and the initial values. Note that, wherever it will be necessary, this hand-waving notation will be more rigorously identified with mathematical objects. However in many cases, this notation will be sufficient. As an example, we can set

$$T = 0, \quad r = s_1 = s_2 = 1, \quad \mathcal{D} = [0,1]^2, \quad \boldsymbol{a} \equiv a(\boldsymbol{x}), \quad \boldsymbol{b} \equiv 1,$$

$$\mathcal{L}(\boldsymbol{a}) := \begin{pmatrix} -\nabla \cdot \boldsymbol{a} \nabla \\ \gamma \end{pmatrix}, \quad \boldsymbol{f} := \begin{pmatrix} 0 \\ \boldsymbol{b} \end{pmatrix},$$

with $\gamma : \boldsymbol{u} \mapsto \boldsymbol{u}|_{\partial\mathcal{D}}$ the trace operator, to describe the elliptic problem

$$\begin{aligned} -\nabla \cdot a(\boldsymbol{x}) \nabla \boldsymbol{u} &= 0 && \text{in } \mathcal{D}\,, \\ \boldsymbol{u} &= 1 && \text{on } \partial\mathcal{D}\,. \end{aligned}$$

Here, by $T = 0$, the problem becomes stationary, thus the time component is dropped. Describing a full initial-boundary value problem is of course a bit more involved, but possible. Also, the notation in (2.1) allows parametrizations in coefficients, boundary conditions and even in domain shapes by appropriate choices of $\mathcal{L}$ and $\boldsymbol{f}$.

2.2.2 Introducing randomness

To be able to transform the fully deterministic, but parametrized problem (2.1) to what will be called random PDE, the parametrization has to be modified to a stochastic setting. Thus, we replace the original parameter functions $\boldsymbol{a}$ and $\boldsymbol{b}$ by random fields or even space-time stochastic processes. After fixing some stochastic space $(\Omega, \mathcal{F}, P)$, we hence introduce space-time stochastic processes $\boldsymbol{a} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^{s_1}$ and $\boldsymbol{b} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^{s_2}$ which become the new parameters for the general PDE problem. As a consequence of introducing space-time stochastic processes as parameters, the right-hand side $\boldsymbol{f}[\boldsymbol{b}]$ and the solution $\boldsymbol{u}$ become dependent on $\omega \in \Omega$, too. We especially have $\boldsymbol{u} : \Omega \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^r$. At the same time, we require the operators $\mathcal{L}$ and $\boldsymbol{f}$ to remain identical to the classical parametrized PDE case.

Throughout this thesis, we moreover have to make the assumption on the solution $\boldsymbol{u}(\omega, \boldsymbol{x}, t)$ to exist for all $\omega \in \Omega$ and on the mapping $\boldsymbol{g} : (\boldsymbol{a}, \boldsymbol{b}) \mapsto \mathcal{L}(\boldsymbol{a})^{-1}\boldsymbol{f}[\boldsymbol{b}]$ to be measurable on the appropriate spaces. Then, the solution $\boldsymbol{u}$ itself is a space-time stochastic process and the evaluation of e.g. stochastic moments of the solution becomes possible. Note however, that these assumptions are not necessarily fulfilled for all to be discussed model and application problems, since, e.g., the existence and uniqueness of strong solutions of the three-dimensional Navier-Stokes equations is still an open question. To be prepared for stochastic collocation techniques described in Section 4.1, we finally also require the solution $\boldsymbol{u}$ to be point evaluable with respect to the stochastic parameter.

Overall we can now state the general *random PDE* problem by

Definition 2.22 (Random PDE problem)**.** *For a given stochastic space $(\Omega, \mathcal{F}, P)$ find a solution $\boldsymbol{u} : \Omega \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^r$ such that*

$$\mathcal{L}(\boldsymbol{a})\boldsymbol{u} = \boldsymbol{f}[\boldsymbol{b}] \qquad \text{in } \Omega \times \bar{\mathcal{D}} \times [0, T] \tag{2.2}$$

holds almost surely. Here, we have $\mathcal{D} \subset \mathbb{R}^d$ the domain, $T \in \mathbb{R}^{\geq 0}$ the end time, $d, r, s_1, s_2 \in \mathbb{N}^+$, $\boldsymbol{a} : \Omega \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^{s_1}$, $\boldsymbol{b} : \Omega \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^{s_2}$ space-time stochastic processes on $(\Omega, \mathcal{F}, P)$ as parameters and $\mathcal{L}, \boldsymbol{f}$ the general PDE and right-hand side operators including boundary and initial condition handling from parametric PDEs, cf. Sec. 2.2.1. Furthermore the solution is expected to exist for all $\omega \in \Omega$ and to be point evaluable. The mapping $\boldsymbol{g} : (\boldsymbol{a}, \boldsymbol{b}) \mapsto \mathcal{L}(\boldsymbol{a})^{-1}\boldsymbol{f}[\boldsymbol{b}]$ from parameter space to the solutions has to be measurable.

Note once again that the above definition covers, as in the parametrized PDE case, much more than pure random-*coefficient* PDE problems. Instead initial conditions, boundary conditions and even the domain (by transformation) might be considered dependent on the space-time-stochastic processes $\boldsymbol{a}$ and $\boldsymbol{b}$. This is why the more general terminology of *random PDEs* is used, covering all these cases. Throughout this thesis, this greater flexibility will be used to define and solve highly complex PDE problems with stochastic influence.

2.3 Function spaces of random PDE solutions

In the following, function spaces for the solutions of random PDE problems are discussed. As usual in the literature, cf. e.g. [Ruz04, Chapter 2], we introduce for a measure space $(\Sigma, \mathcal{G}, \mu)$ and a Banach space $(X, \|\cdot\|_X)$ the linear space $L^p(\Sigma; X)$ $(1 \leq p < \infty)$ of all strongly measurable functions $f : \Omega \to X$ with finite integral $\int_\Sigma \|f\|_X^p d\mu$, thus

$$L^p(\Sigma; X) := \left\{ f : \Omega \to X \middle| \|f\|_{L^p(\Sigma;X)} < \infty \right\}, \quad \|f\|_{L^p(\Sigma;X)} := \left(\int_\Sigma \|f\|_X^p d\mu \right)^{1/p}.$$

This function space is often called *(Lebesgue-)Bochner space* and is in fact a space of equivalence classes of functions. For $p = \infty$, we also get the space

$$L^\infty(\Sigma; X) := \left\{ f : \Omega \to X \middle| \|f\|_{L^\infty(\Sigma;X)} < \infty \right\}, \quad \|f\|_{L^\infty(\Sigma;X)} := \operatorname{ess\,sup}_{\sigma\in\Sigma} \|f\|_X < \infty\,.$$

Before we look at Bochner spaces, to describe the solutions of *random* PDEs, we should remember that we started from *deterministic time-dependent* PDE problems. We do not assume strict regularity on these solutions. Furthermore, we express the deterministic solutions as fields in $\mathcal{D}$ that are functions in time. Therefore, they are elements of a Bochner space

$$L^2\left([0,T]; L^2(\mathcal{D})\right) := \left\{ f : [0,T] \to L^2(\mathcal{D}) \middle| \|f\|_{L^2([0,T];L^2(\mathcal{D}))} < \infty \right\},$$

$$\|f\|_{L^2([0,T];L^2(\mathcal{D}))} := \left(\int_{[0,T]} \|f\|^2_{L^2(\mathcal{D})} dt \right)^{1/2} .$$

In case of $\boldsymbol{u} : \mathcal{D} \times [0,T] \to \mathbb{R}^r$ with $r = 1$, $L^2(\mathcal{D})$ is the usual ($\mathbb{R}$-valued) Lebesgue function space on domain $\mathcal{D}$. However, for $r > 1$, the solution $\boldsymbol{u}$ would be $\mathbb{R}^r$-valued, which requires to introduce a vector-valued Lebesgue function space, which would be again some Bochner space

$$L^2\left(\mathcal{D}; L^2(\mathbb{R}^r)\right) := \left\{ f : \Omega \to \mathbb{R}^r \middle| \|f\|_{L^2(\mathcal{D};L^2(\mathbb{R}^r))} < \infty \right\}$$

$$\|f\|_{L^2(\mathcal{D};L^2(\mathbb{R}^r))} := \left(\int_{\mathcal{D}} \|f\|^2_{L^2(\mathbb{R}^r)} d\boldsymbol{x} \right)^{1/2} .$$

To be concise, this technical detail is skipped, in the following. Wherever suitable, we assume that

$$L^2(\mathcal{D}) := L^2(\mathcal{D}; L^2(\mathbb{R}^r))\,.$$

Now, we move over to random PDE problems as given by Definition 2.22. Let us for now ignore the time-dependence of the solutions, thus looking only at stationary random PDE problems. Then we have a similar construction. With $(\Omega, \mathcal{F}, P)$ the underlying probability space, the solution (Bochner) function space $L^2(\Omega; L^2(\mathcal{D}))$ is thus assumed to be

$$L^2\left(\Omega; L^2(\mathcal{D})\right) := \left\{ f : \Omega \to L^2(\mathcal{D}) \middle| \|f\|_{L^2(\Omega;L^2(\mathcal{D}))} < \infty \right\},$$

$$\|f\|_{L^2(\Omega;L^2(\mathcal{D}))} := \left(\int_{\Omega} \|f\|^2_{L^2(\mathcal{D})} dP(\omega) \right)^{1/2} .$$

A combination of the space $L^2([0,T]; L^2(\mathcal{D}))$ for time dependent deterministic PDE solutions and the space $L^2(\Omega; L^2(\mathcal{D}))$ of solutions of stationary random PDE problems, finally leads to the function space of solutions to the random PDE problem given in Definition 2.22. In this case, it is usual (cf. e.g. [BNT10, BSZ11]) to assume $\boldsymbol{u} \in L^2(\Omega; [0,T]; L^2(\mathcal{D}))$, with

$$L^2\left(\Omega; [0,T]; L^2(\mathcal{D})\right) := L^2\left(\Omega; L^2\left([0,T]; L^2(\mathcal{D})\right)\right),$$

$$L^2\left(\Omega; L^2\left([0,T]; L^2(\mathcal{D})\right)\right) := \left\{ f : \Omega \to L^2\left([0,T]; L^2(\mathcal{D})\right) \middle| \|f\|_{L^2(\Omega;[0,T];L^2(\mathcal{D}))} < \infty \right\},$$

$$\|f\|_{L^2(\Omega;[0,T];L^2(\mathcal{D}))} := \left(\int_\Omega \|f\|^2_{L^2([0,T];L^2(\mathcal{D}))} dP(\omega)\right)^{1/2} .$$

It is well-known that the Bochner space $L^2(\Omega; L^2(\mathcal{D}))$ is isometric isomorph to the space $L^2(\Omega) \otimes L^2(\mathcal{D})$. Also, the time-dependent random PDE problem solution space $L^2(\Omega; [0,T]; L^2(\mathcal{D}))$ is isometric isomorph to $L^2(\Omega) \otimes L^2([0,T]; L^2(\mathcal{D}))$.

2.4 Finite-dimensional noise assumption

Let us remember the problem statement from Definition 2.22, in which, given a probability space $(\Omega, \mathcal{F}, P)$, one looks for a random function or more specifically for a space-time stochastic process $\boldsymbol{u} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^r$ with $\mathcal{D} \subset \mathbb{R}^d$, such that almost surely it holds

$$\mathcal{L}(\boldsymbol{a}(\omega, \boldsymbol{x}, t))\boldsymbol{u}(\omega, \boldsymbol{x}, t) = \boldsymbol{f}[\boldsymbol{b}(\omega, \boldsymbol{x}, t)](\omega, \boldsymbol{x}, t) \qquad \text{in } \Omega \times \bar{\mathcal{D}} \times [0,T] . \tag{2.3}$$

In this and the following chapters, the notation proposed in standard stochastic collocation papers like [NTW08b, BNT10] is widely applied.

To be able to handle the above problem numerically, it is necessary to transform the infinite dimensional stochastic variable ω to some finite-dimensional representation. The typical approach e.g. in the above mentioned literature is to introduce a *finite-dimensional noise assumption*, cf. [BNT10], for the stochastic parameter functions $\boldsymbol{a}$ and $\boldsymbol{b}$. This basically assumes the stochastic dependence of the parameter functions $\boldsymbol{a}, \boldsymbol{b}$ to be represented by a finite number of N_{FN} random variables $\{Y_m(\omega)\}_{m=1}^{N_{FN}}$ on $(\Omega, \mathcal{F}, P)$ with $Y_m : \Omega \to \mathbb{R}$, $\mathbb{E}[Y_m] = 0$ and $\mathrm{Var}[Y_m] = 1$. Thus we seek for approximations of the form

$$\boldsymbol{a}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{a}_{FN}(Y_1(\omega), \dots, Y_{N_{FN}}(\omega), \boldsymbol{x}, t) , \tag{2.4}$$

$$\boldsymbol{b}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{b}_{FN}(Y_1(\omega), \dots, Y_{N_{FN}}(\omega), \boldsymbol{x}, t) . \tag{2.5}$$

In the following paragraphs, the Karhunen-Loève expansion and finite-dimensional random models are discussed motivating the above approximation. Thereafter, the operator equation for the general random PDE problem (2.3) is adapted according to the finite noise assumption, leading to a finite-dimensional random PDE problem, which can be solved numerically.

2.4.1 Karhunen-Loève expansion for stochastic parameter functions

One motivation for the finite-dimensional noise assumption typically comes from the Karhunen-Loève decomposition [Loe78, Chapter XI], which gives a series expansion of a stochastic process $a(t)$ or a random field $a(\boldsymbol{x})$ based on an eigenvalue decomposition of the associated covariance function. We follow [ST06] with adapted notation and obtain for random fields

Theorem 2.1 (Karhunen-Loève expansion [ST06, Prop. 2.8],[Xiu10, Section 4.2.1],[LMK10, Section 2.1.1])**.** *Let $(\Omega, \mathcal{F}, P)$ be a probability space and $a : \Omega \times \mathcal{D} \to \mathbb{R}$ a random field with $a(\cdot, \omega) \in L^2$ for all $\omega \in \Omega$ and $a \in L^2(\Omega, \mathcal{F}, P)$, thus $a \in L^2(\Omega \times \mathcal{D})$. There exists a sequence of random variables $Y_m \in L^2(\Omega, \mathcal{F}, P)$, $m \geq 1$, with*

$$\mathbb{E}[Y_m] = \int_\Omega Y_m(\omega) dP(\omega) = 0, \quad \mathbb{E}[Y_n Y_m] = \int_\Omega Y_n(\omega) Y_m(\omega) dP(\omega) = \delta_{nm}, \quad \forall n, m \geq 1 ,$$

such that we can describe the random field a in $L^2(\Omega \times \mathcal{D})$ by the Karhunen-Loève expansion

$$a(\omega, \boldsymbol{x}) = \mathbb{E}\left[a\right](\boldsymbol{x}) + \sum_{m=1}^{\infty} \sqrt{\lambda_m} Y_m(\omega) \psi_m(\boldsymbol{x}) \,. \tag{2.6}$$

Here, (λ_m, ψ_m), $m \geq 1$ denotes the sequence of eigenvalues and eigenfunctions of the Carleman operator

$$C_{Cov[a]} : L^2(\mathcal{D}) \to L^2(\mathcal{D}), \quad C_{Cov[a]} : \psi \mapsto \int_{\mathcal{D}} Cov\left[a\right](\cdot, \boldsymbol{x}) \psi(\boldsymbol{x}) d\boldsymbol{x}, \quad \forall \psi \in L^2(\mathcal{D}) \,,$$

thus the (λ_m, ψ_m) pairs solve the integral equation

$$\int_{\mathcal{D}} Cov\left[a\right](\boldsymbol{x}, \boldsymbol{x}') \psi_m(\boldsymbol{x}') d\boldsymbol{x}' = \lambda_m \psi_m(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in \mathcal{D} \,.$$

It further holds that the eigenvalues λ_m are non-negative with $\sum_{m \geq 1} {\lambda_m}^2 < +\infty$. The random variables Y_m are derived as

$$Y_m(\omega) := \frac{1}{\sqrt{\lambda_m}} \int_{\mathcal{D}} \left(a(\omega, \boldsymbol{x}) - \mathbb{E}\left[a\right](\boldsymbol{x})\right) \psi_m(\boldsymbol{x}) d\boldsymbol{x}, \quad \forall m \geq 1 \,.$$

Proofs for this fundamental result can be found e.g. in [ST06] or [Loe78, Chapter XI]. Since the eigenvalues λ_m are non-negative and obey the given finite summability condition, they have to decay to zero for $m \to \infty$. This motivates to truncate the expansion (2.6) after the first N_{KL} terms as

$$a(\omega, \boldsymbol{x}) \approx a_{KL}(\omega, \boldsymbol{x}) = \mathbb{E}\left[a\right](\boldsymbol{x}) + \sum_{m=1}^{N_{KL}} \sqrt{\lambda_m} Y_m(\omega) \psi_m(\boldsymbol{x}) \,. \tag{2.7}$$

Indeed, we know from [LMK10, Section 2.1.2] that the truncated Karhunen-Loève expansion (2.7), is the best N_{KL}-term approximation of the given random field with respect to the error norm $\mathbb{E}\left[\|\cdot\|_{\mathcal{D}}^2\right]^{1/2}$.

With obvious extensions to $\mathbb{R}^s$-valued space-time stochastic processes, it is now possible to introduce approximations to the parameter functions $\boldsymbol{a}, \boldsymbol{b}$ by

$$\boldsymbol{a}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{a}_{KL}(Y_1^{\boldsymbol{a}}(\omega), \ldots, Y_{N_{KL}^{\boldsymbol{a}}}^{\boldsymbol{a}}(\omega), \boldsymbol{x}, t) = \mathbb{E}\left[\boldsymbol{a}\right](\boldsymbol{x}, t) + \sum_{m=1}^{N_{KL}^{\boldsymbol{a}}} \sqrt{\lambda_m^{\boldsymbol{a}}} Y_m^{\boldsymbol{a}}(\omega) \boldsymbol{\psi}_m^{\boldsymbol{a}}(\boldsymbol{x}, t) \,,$$

$$\boldsymbol{b}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{b}_{KL}(Y_1^{\boldsymbol{b}}(\omega), \ldots, Y_{N_{KL}^{\boldsymbol{b}}}^{\boldsymbol{b}}(\omega), \boldsymbol{x}, t) = \mathbb{E}\left[\boldsymbol{b}\right](\boldsymbol{x}, t) + \sum_{m=1}^{N_{KL}^{\boldsymbol{b}}} \sqrt{\lambda_m^{\boldsymbol{b}}} Y_m^{\boldsymbol{a}}(\omega) \boldsymbol{\psi}_m^{\boldsymbol{b}}(\boldsymbol{x}, t) \,.$$

To simplify notation, it is assumed to have a common sequence of random variables Y_m with a common truncation N_{KL} for both $\boldsymbol{a}$ and $\boldsymbol{b}$ such that

$$\boldsymbol{a}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{a}_{KL}(Y_1(\omega), \ldots, Y_{N_{KL}}(\omega), \boldsymbol{x}, t) \,,$$

$$\boldsymbol{b}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{b}_{KL}(Y_1(\omega), \ldots, Y_{N_{KL}}(\omega), \boldsymbol{x}, t)\,,$$

with $\mathbb{E}[Y_m] = 0$ and $\mathbb{E}[Y_m Y_n] = \delta_{mn}$ as required by Theorem 2.1 and $\mathrm{Var}[Y_m] = 1$. This is exactly what we are expecting in the finite noise assumption.

2.4.2 Finite-dimensional random models

The finite-dimensional noise assumption also holds for other examples. One of these is e.g. mentioned in [NTW08b], where a piecewise constant random field is assumed. In that case, a random function

$$a_N(\omega, \boldsymbol{x}) = a_{min} + \sum_{i=1}^{N_{FN}} \sigma_i Y_i(\omega) 1_{\mathcal{D}_i}(\boldsymbol{x})$$

is the coefficient field in a general elliptic problem in which the physical domain $\mathcal{D}$ is the union of non-overlapping subdomains $\mathcal{D}_i$ and the σ_i are domain-dependent diffusion coefficients, which are piecewise constant with stochastic dependence on random variables Y_i with $\mathbb{E}[Y_i] = 0$ and $\mathrm{Var}[Y_m] = 1$. Here, $1_{\mathcal{D}_i}$ is the indicator function for subdomain $\mathcal{D}_i$.

Other constructions of this kind are also possible, e.g. if the stochastic dependence of a given PDE coefficient is assumed to be very simple.

2.4.3 Finite-dimensional stochastic space in random PDEs

Using the finite-dimensional noise assumption, we can approximate the space-time stochastic processes $\boldsymbol{a}, \boldsymbol{b}$ which are parameter functions as

$$\boldsymbol{a}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{a}_{FN}(Y_1(\omega), \ldots, Y_{N_{FN}}(\omega), \boldsymbol{x}, t)\,,$$
$$\boldsymbol{b}(\omega, \boldsymbol{x}, t) \approx \boldsymbol{b}_{FN}(Y_1(\omega), \ldots, Y_{N_{FN}}(\omega), \boldsymbol{x}, t)\,.$$

Following the lines of [BNT10], we set the images of the Y_m to be $\Gamma_m \equiv Y_m(\Omega) \subseteq \mathbb{R}$ and the tensor product of these spaces is $\Gamma := \prod_{m=1}^{N_{FN}} \Gamma_m \subseteq \mathbb{R}^{N_{FN}}$. We further can define the joint probability density function $\rho : \Gamma \to \mathbb{R}_+$ with $\rho \in L^\infty(\Gamma)$ for the finite set of random variables and usually assume the Y_m to be independent. Then we have also $\rho := \prod_{n=1}^{N_{FN}} \rho_m$ with ρ_m the density functions of the Y_m.

With this in mind, we reformulate the random parameter functions to operate on the tensor product space of images of the $Y_m(\omega)$ as

$$\hat{\boldsymbol{a}}_{FN}, \hat{\boldsymbol{b}}_{FN} : \Gamma \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^{s_1, s_2}\,.$$

For $\boldsymbol{y} = (Y_1(\omega), \ldots, Y_{N_{FN}}(\omega))^\top$, we thus introduce the approximations

$$\boldsymbol{a}(\omega, \boldsymbol{x}, t) \approx \hat{\boldsymbol{a}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t), \quad \boldsymbol{b}(\omega, \boldsymbol{x}, t) \approx \hat{\boldsymbol{b}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t)\,. \tag{2.8}$$

It is then possible to rewrite the random PDE problem statement from Definition 2.22 as:

Find function $\hat{\boldsymbol{u}}_{NF} : \Gamma \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^r$, such that it holds for each $\boldsymbol{y} \in \Gamma$ that

$$\mathcal{L}\left(\hat{\boldsymbol{a}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t)\right) \hat{\boldsymbol{u}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t) = \boldsymbol{f}\left[\hat{\boldsymbol{b}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t)\right](\boldsymbol{y}, \boldsymbol{x}, t) \qquad \text{in } \Gamma \times \bar{\mathcal{D}} \times [0, T]\,. \tag{2.9}$$

In fact, the above problem is (after measure change) again a random PDE problem with the stochastics-related parameter $\boldsymbol{y}$. Following [BNT10], the technically correct formulation would be to require (2.9) to hold ρ-almost-everywhere in Γ, noting that for fixed $\boldsymbol{x} \in \mathcal{D}, t \in [0,T]$ we have that e.g. $\hat{\boldsymbol{a}}_{FN}(\cdot, \boldsymbol{x}, t)$ would be a random variable on $(\Gamma, \mathcal{B}^{N_{FN}}, \rho\, d\boldsymbol{y})$. But this technical detail is skipped in the following.

The finite-dimensional formulation now allows for the use of standard numerical approximation techniques, avoiding the usual rather technically and notationally complex formulation of the problems with respect to some probability space. However, it is necessary to understand the implications of the evaluation of stochastic moments of quantities like the solution field $\hat{\boldsymbol{u}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t)$. In fact, these evaluations are still with respect to the original probability space $(\Omega, \mathcal{F}, P)$. For the mean of e.g. $\boldsymbol{u}$, we thus have by applying Lemma 2.6 and (2.8) that

$$
\begin{aligned}
\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x}, t) &= \int_\Omega \boldsymbol{u}(\omega, \boldsymbol{x}, t) P(d\omega) && (2.10)\\
&\approx \int_\Omega \hat{\boldsymbol{u}}_{FN}\left((Y_1(\omega), \dots, Y_{N_{FN}}(\omega))^\top, \boldsymbol{x}, t \right) P(d\omega) && (2.11)\\
&= \int_\Gamma \hat{\boldsymbol{u}}_{FN}(\boldsymbol{y}, \boldsymbol{x}, t) \rho(\boldsymbol{y}) d\boldsymbol{y}\,, && (2.12)
\end{aligned}
$$

assuming all involved compound functions to be measurable.

To try to keep notation at a readable level, less strict formulations shall be used throughout the rest of this thesis. Following similar approaches from [Xiu10, Section 4.3] or [BNT10], the finite-dimensional problem formulation after the finite noise assumption will be taken as standard without the additional indicators of the involved quantities to be approximations. We thus seek for functions $\boldsymbol{u} : \Gamma \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^r$ such that for each $\boldsymbol{y} \in \Gamma$, it holds

$$
\mathcal{L}\left(\boldsymbol{a}(\boldsymbol{y}, \boldsymbol{x}, t)\right) \boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}, t) = \boldsymbol{f}\left[\boldsymbol{b}(\boldsymbol{y}, \boldsymbol{x}, t)\right](\boldsymbol{y}, \boldsymbol{x}, t) \qquad \text{in } \Gamma \times \bar{\mathcal{D}} \times [0,T]\,. \tag{2.13}
$$

As a side effect, a simplified notation for, e.g., the mean is

$$
\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x}, t) := \int_\Gamma \boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}, t) \rho(\boldsymbol{y}) d\boldsymbol{y}\,.
$$

Analogous abbreviations for correlation, covariance and variance will be used.

2.4.4 Solution function space

The beforehand construction leads to a different solution function space for the random PDE problem. In fact, we now have for the probability space $(\Gamma, \mathcal{B}^{N_{FN}}, \rho d\boldsymbol{y})$ the solution $\hat{\boldsymbol{u}}_{FN} : \boldsymbol{\Gamma} \times \mathcal{D} \times [0,T] \to \mathbb{R}^r$, which is given in the Bochner function space $L^2(\Gamma; [0,T]; L^2(\mathcal{D}))$, with

$$
L^2\left(\Gamma; [0,T]; L^2(\mathcal{D})\right) := L^2\left(\Gamma; L^2\left([0,T]; L^2(\mathcal{D})\right)\right)\,,
$$

$$
L^2\left(\Gamma; L^2\left([0,T]; L^2(\mathcal{D})\right)\right) := \left\{ f : \Gamma \to L^2([0,T]; L^2(\mathcal{D})) \middle| \|f\|_{L^2(\Gamma;[0,T];L^2(\mathcal{D}))} < \infty \right\}\,,
$$

$$
\|f\|_{L^2(\Gamma;[0,T];L^2(\mathcal{D}))} := \left(\int_\Gamma \|f\|^2_{L^2([0,T];L^2(\mathcal{D}))} \rho(\boldsymbol{y}) d\boldsymbol{y} \right)^{1/2}\,.
$$

$$\begin{tikzcd}
\boxed{\begin{array}{c}\text{parameters}\\ \boldsymbol{a}(\boldsymbol{y},\boldsymbol{x},t)\\ \boldsymbol{b}(\boldsymbol{y},\boldsymbol{x},t)\end{array}} \arrow[r] \arrow[rr, bend left=40, "\substack{\boldsymbol{g}(\boldsymbol{a},\boldsymbol{b}) := \mathcal{L}(\boldsymbol{a})^{-1}\boldsymbol{f}[\boldsymbol{b}] \\ \text{response surface}}"] \arrow[rrr, bend right=30, "\boldsymbol{\pi}\circ\boldsymbol{g}"'] & \boxed{\begin{array}{c}\text{random}\\ \text{PDE}\\ \mathcal{L}(\boldsymbol{a})\boldsymbol{u}=\boldsymbol{f}[\boldsymbol{b}]\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{solution}\\ \text{field}\\ \boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)\end{array}} \arrow[r] \arrow[r, bend left=50, "\boldsymbol{\pi}"] & \boxed{\begin{array}{c}\text{quantity of}\\ \text{interest}\\ \boldsymbol{\pi}(\boldsymbol{u})(\boldsymbol{y},\boldsymbol{x},t)\end{array}} \arrow[r] & \boxed{\begin{array}{c}\text{moments}\\ \mathbb{E}\left[\boldsymbol{\pi}(\boldsymbol{u})\right]\\ \mathrm{Var}\left[\boldsymbol{\pi}(\boldsymbol{u})\right]\\ \mathrm{Cov}\left[\boldsymbol{\pi}(\boldsymbol{u})\right]\end{array}}
\end{tikzcd}$$

Figure 2.1: Uncertainty quantification for random PDEs aims at evaluating stochastic moments of quantities of interest, which are extracted from the random PDE solution field.

Moreover, it is well-known that we have the an isometric isomorphism

$$L^2\left(\Gamma;[0,T];L^2(\mathcal{D})\right) \cong L^2(\Gamma)\otimes L^2\left([0,T];L^2(\mathcal{D})\right) .$$

2.5 Uncertainty quantification for random PDEs

After having developed a formalism to describe random PDE problems in a finite-dimensional setting, it is necessary to give an idea of the concepts of uncertainty quantification that shall be discussed.

In this thesis, the quantification of uncertainties is understood to be the evaluation of stochastic moments either of the solution of a random PDE problem or of some derived quantity of the random PDE problem solution. Figure 2.1 outlines the basic idea. Starting from the (random) input parameter fields, the random PDE problem (2.13) is solved. Its solution field $\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)$ is still a space-time stochastic process. From a deterministic point of view, the solution is a function $\boldsymbol{g}(\boldsymbol{a},\boldsymbol{b})$ of the input parameter fields thus,

$$\boldsymbol{u} = \boldsymbol{g}(\boldsymbol{a},\boldsymbol{b}) := \mathcal{L}(\boldsymbol{a})^{-1}\boldsymbol{f}[\boldsymbol{b}]\,.$$

In the literature, the image of $\boldsymbol{g}$ is often called *response surface* and the function itself describes the response of a system (here the random PDE) to some input. Usually, the response function $\boldsymbol{g}$ is defined with respect to some input vector, which could be $\boldsymbol{y}\in\Gamma$ in this case. However, the intention here is to stick to the formulation as some dependent function on $\boldsymbol{a},\boldsymbol{b}$. This shall underline that the parameter functions could contain hidden deterministic parameters which are not subject to stochastic influence but for which it makes also sense to define a response function.

The next step in uncertainty quantification for random PDEs, as discussed in this thesis, cf. Figure 2.1, is the mapping of the solution field $\boldsymbol{u}$ to a *quantity of interest* (QOI). This is done with the mapping function $\boldsymbol{\pi}$. Very often, a quantity of interest is a single number describing e.g. some averaged solution field property, a single value extracted from the solution field or extremal values. Also, more complex quantities, like forces or travel times, which might even require the solution of another PDE, are possible. Of course, quantities of interest might also be full vector fields, in which case the mapping function $\boldsymbol{\pi}$ picks a part of the solution field or might be the identity. Overall, the definition of the mapping $\boldsymbol{\pi}$ is highly problem-specific.

In the final step of uncertainty quantification for random PDEs, stochastic moments of the quantity of interest (with respect to the random input quantities) are evaluated. These give important characterizations of the overall stochastic properties with a rather simple description method. Even though Figure 2.1 suggests a sequential numerical treatment of the described "steps", this does not have to be the case.

Numerical methods to evaluate stochastic moments of random PDEs are usually divided in two groups, *intrusive* and *non-intrusive methods*. Intrusive methods require to implement new solvers which discretize physical space and stochastic space at the same time. A prominent example for intrusive methods is the stochastic Galerkin approach [GS91],[LMK10, Chapter 4]. On the other hand, non-intrusive methods rely on (approximate) solutions from existing deterministic PDE solvers. A standard example for non-intrusive methods is the stochastic collocation method [NTW08b, BNT10]. For a numerical comparison of both exemplified methods see e.g. [BNTT11]. In this thesis, the non-intrusive approach is chosen, since the target applications are large-scale random PDE problems with existing (deterministic) solvers. To exemplify this, the existing (multi-GPU ported) two-phase flow solver NaSt3DGPF will be used. An in-depth discussion of the non-intrusive RBF kernel-based stochastic collocation method will be given after having introduced the model and application problems in the next chapter.

3 Model and application problems

This chapter collects the model and application problems, which will be discussed throughout this thesis. It starts by introducing two model problems, for which analytic solutions are known. These will facilitate error measurements. The second pair of problems is used to analyze and to compare the convergence properties of the applied kernel-based uncertainty quantification method to well-known numerical results from the literature. Both problems are random-coefficient elliptic problems with two space dimensions and several stochastic dimensions. They are well-studied in the literature, cf. [BNT10, BNTT11, van14], and require some computational effort. The third set of problems is based on a random version of the two-phase incompressible Navier-Stokes equations. These problems will be the main objective applications in this thesis. They feature all the limitations of classical engineering-oriented problems in this domain. These are high to extreme computational costs to evaluate a single deterministic PDE, widely unknown mathematical properties and many parameters (initial values, physical properties, domain shape, boundary conditions ...) which might be considered uncertain in real-world.

3.1 Problems with analytic solution

Let us start by defining two model problems for which analytic solutions are known. The idea is here, to have an optimal setting for empirical error measurements.

3.1.1 Random-coefficient elliptic problem with known solution

The first problem is a random PDE, which is motivated by an example proposed in [TPME11]. Given a complete probability space $(\Omega, \mathcal{F}, P)$, the objective is to find a random field $u : \Omega \times [-0.5, 0.5]^2 \to \mathbb{R}$, such that

$$\begin{aligned} -\nabla \cdot (a(\omega, \boldsymbol{x}) \nabla u(\omega, \boldsymbol{x})) &= \; f(\boldsymbol{x}) \quad \text{in } \Omega \times (-0.5, 0.5)^2 \,, \\ u(\omega, \boldsymbol{x}) &= \; 0 \quad \text{on } \Omega \times \partial(-0.5, 0.5)^2 \,. \end{aligned}$$

holds almost surely. The finite noise assumption is fulfilled by introducing a one-term Karhunen-Loève expansion of the diffusion coefficient $a(\omega, \boldsymbol{x})$ resulting in the finite-dimensional problem to find $\boldsymbol{u} : \Gamma \times [-0.5, 0.5]^2 \to \mathbb{R}$, such that

$$\begin{aligned} -\nabla \cdot (a(\boldsymbol{y}, \boldsymbol{x}) \nabla u(\boldsymbol{y}, \boldsymbol{x})) &= \; f(\boldsymbol{x}) \quad \text{in } \Gamma \times (-0.5, 0.5)^2 \,, \\ u(\boldsymbol{y}, \boldsymbol{x}) &= \; 0 \quad \text{on } \Gamma \times \partial(-0.5, 0.5)^2 \,. \end{aligned}$$

with $\Gamma \subseteq \mathbb{R}$ and the random diffusion coefficient

$$a(\boldsymbol{y}, \boldsymbol{x}) = 1 + \sigma \frac{1}{\pi^2} y_1 \cos\left(\frac{\pi}{2}\left(x_1^2 + x_2^2\right)\right) ,$$

thus we have $\boldsymbol{y} = y_1$. The right-hand side term is given as

$$\begin{aligned} f(\boldsymbol{y}, \boldsymbol{x}) =& 32\left(1 + \sigma + \frac{y_1 \cos(\frac{1}{2}\pi(x_1^2 + x_2^2))}{\pi^2}\right) e^{-y_1^2}\left(x_2^2 - \frac{1}{2} + x_1^2\right) \\ & - \frac{32}{\pi} y \sin\left(\frac{1}{2}\pi(x_1^2 + x_2^2)\right)\left(x_1^2 e^{-y_1^2}\left(x_2^2 - \frac{1}{4}\right) + x_2^2 e^{-y_1^2}\left(x_1^2 - \frac{1}{4}\right)\right) . \end{aligned}$$

With this construction, it is possible to derive an exact solution of the random PDE problem as

$$u(\boldsymbol{y}, \boldsymbol{x}) = 16\, e^{-y_1^2}\left(x_1^2 - \frac{1}{4}\right)\left(x_2^2 - \frac{1}{4}\right) .$$

According to [TPME11], this problem is only well-posed if $|y_1| < \frac{\pi^2}{\sigma}$. In difference to that reference, the variable y_1 shall here correspond to the random variable $Y_1(\omega) \sim \mathcal{U}(-\sqrt{3}, \sqrt{3})$, thus we employ the density function $\rho(\boldsymbol{y}) = \frac{1}{2\sqrt{3}}$. It is possible to derive the exact solution for the first stochastic moment as

$$\mathbb{E}[u] = \frac{1}{6} \operatorname{erf}\left(\sqrt{3}\right) \sqrt{3}\sqrt{\pi}\left(16x_1^2x_2^2 - 4x_1^2 - 4x_2^2 + 1\right) .$$

The exact solution for the second moment is also available. This easily allows an analysis of the first and second moment of the quantity of interest $\pi \equiv Id$, thus e.g. $\mathbb{E}[\pi(u)] = \mathbb{E}[u]$ shall be approximated, cf. Section 2.5.

3.1.2 g function

To be able to asses the discussed numerical methods also for a higher-dimensional stochastic space Γ, the second problem with analytic solution is a non-PDE test problem, the g function, cf. [DR84]. It allows to validate the involved numerical methods without solving any equation. Adapted to the notation of this thesis, it reads as

$$u_{N_{FN}}(\boldsymbol{y}) = \prod_{m=1}^{N_{FN}} \frac{|4y_m - 2| + a_m}{1 + a_m} .$$

Following [SB13], we assume $a_m = \frac{m-2}{2}$. It is well-known, that for y_m corresponding to independent random variables $Y_m(\omega) \sim \mathcal{U}(0, 1)$, the mean is given by

$$\mathbb{E}[u_{N_{FN}}] = 1$$

for arbitrary dimension $N_{FN} \geq 1$ of the space Γ. The density function is $\rho(\boldsymbol{y}) = 1$. Throughout this thesis, the first stochastic moment of the g function with quantity of interest $\pi \equiv Id$ is studied. Note that the above problem has finite smoothness in stochastic space, because of the absolute value operator.

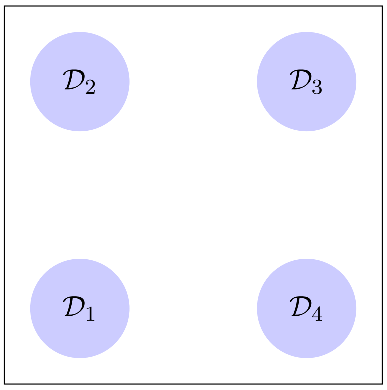


Figure 3.1: The two-dimensional elliptic model problem with piecewise constant random diffusion field has four circle-shaped subdomains with constant random diffusion coefficients.

3.2 Two-dimensional random-coefficient elliptic problems

The second set of model problems is probably the most studied problem in the field. It is the random-coefficient elliptic problem, with domain $\mathcal{D} = [0,1]^2$ and a complete probability space $(\Omega, \mathcal{F}, P)$, as usual. The objective is to find a random field $u : \Omega \times \bar{\mathcal{D}} \to \mathbb{R}$, such that almost surely

$$\begin{aligned} -\nabla \cdot (a(\omega, \boldsymbol{x}) \nabla u(\omega, \boldsymbol{x})) = &\quad f(\boldsymbol{x}) \quad \text{in } \Omega \times \mathcal{D}\,, \\ u(\omega, \boldsymbol{x}) = &\quad 0 \quad \text{on } \Omega \times \partial\mathcal{D}\,. \end{aligned}$$

Here, $f : \mathcal{D} \to \mathbb{R}$ is some deterministic forcing term. Boundary conditions are also deterministic with a homogeneous Dirichlet boundary.

After introducing the finite noise assumption with respect to $a(\omega, \boldsymbol{x})$, it is possible to reformulate the given problem in the finite-dimensional sense requiring to find a function $\boldsymbol{u} : \Gamma \times \bar{\mathcal{D}} \to \mathbb{R}$, such that

$$-\nabla \cdot (a(\boldsymbol{y}, \boldsymbol{x}) \nabla u(\boldsymbol{y}, \boldsymbol{x})) = f(\boldsymbol{x}) \quad \text{in } \Gamma \times \mathcal{D}\,, \tag{3.1}$$

$$u(\boldsymbol{y}, \boldsymbol{x}) = 0 \quad \text{on } \Gamma \times \partial\mathcal{D}\,. \tag{3.2}$$

Using that general random PDE, two sub-problems are defined to reflect different standard constructions found in the literature.

3.2.1 Piecewise constant random diffusion field

In the first sub-problem, taken from [BNT10, BNTT11], the finite noise assumption is fulfilled by a piecewise constant random field as motivated in Section 2.4.2. The domain-dependent diffusion coefficient is thus given as

$$a(\boldsymbol{y}, \boldsymbol{x}) = 1 + \sum_{m=1}^{4} \lambda_m y_m \chi_m(\boldsymbol{x}) ,$$

with χ_m the indicator function for the four circle-shaped subdomains $\mathcal{D}_1, \ldots, \mathcal{D}_4 \subset \mathcal{D}$ with radius 0.13 each, cf. Figure 3.1. The centers of the subdomains are at coordinates $(0.2, 0.2)$, $(0.2, 0.8)$, $(0.8, 0.8)$ and $(0.8, 0.2)$ and their parameters λ_m are given as

$$\lambda_1 = 1.0, \quad \lambda_2 = 0.9, \quad \lambda_3 = 0.75, \quad \lambda_4 = 0.6 .$$

Furthermore, the y_m correspond to random variables $Y_m(\omega) \sim \mathcal{U}(-0.99, 0)$, thus $\rho(\boldsymbol{y}) = \left(\frac{1}{0.99}\right)^4$, and $f \equiv 1$. The first stochastic moment shall be evaluated for the quantity of interest $\pi \equiv Id$.

3.2.2 Karhunen-Loève expansion-based random diffusion field

The second sub-problem bases on [BNT10, van14]. It uses $f \equiv 1$ in equation (3.1). Here, the diffusion coefficient $a \equiv a_{N_{KL},L_c}$ is given by a Karhunen-Loève expansion as

$$\log\left(a_{N_{KL},L_c}(\boldsymbol{y}, \boldsymbol{x}) - 0.5\right) = 1 + y_1 \left(\frac{\sqrt{\pi} L_c}{2}\right)^{1/2} + \sum_{m=2}^{N_{KL}} \lambda_m \phi_m(x_1) y_m ,$$

with the $\{y_m\}_{m=1}^{N_{KL}}$ corresponding to independent random variables $\{Y_m(\boldsymbol{\omega})\}_{m=1}^{N_{KL}}$ with each $Y_m \sim \mathcal{U}(-\sqrt{3}, \sqrt{3})$, thus $\rho(\boldsymbol{y}) = \left(\frac{1}{2\sqrt{3}}\right)^{N_{KL}}$ and

$$\lambda_m := (\sqrt{\pi} L_c)^{1/2} \exp\left(\frac{-(\lfloor \frac{m}{2} \rfloor \pi L_c)^2}{8}\right), \quad \phi_m(x_1) := \begin{cases} \sin(\lfloor \frac{m}{2} \rfloor \pi x_1) & \text{if } m \text{ even,} \\ \cos(\lfloor \frac{m}{2} \rfloor \pi x_1) & \text{if } m \text{ odd} \end{cases}, \quad m > 1$$

This construction approximates a one-dimensional random variable a with covariance

$$\mathrm{Cov}\left[\log(a - 0.5)\right](x, x') = e^{-\frac{(x-x')^2}{L_c^2}} .$$

N_{KL} is the number of terms of the Karhunen-Loève expansion. Furthermore, L_c is the correlation length. Both parameters can be modified in the numerical experiments. The quantity of interest $\pi \equiv Id$ is again used and the first two stochastic moments shall be approximated.

3.3 Random two-phase incompressible Navier-Stokes equations

The major application problem motivating the work presented in this thesis is a random version of the two-phase incompressible Navier-Stokes equations. They model the interaction of two incompressible fluids which do not mix but remain disjoint with a common interface. Classical examples for such fluid-fluid systems in real world are oil and water or water and air, noting that it is usual in the field to assume air to be incompressible for the discussed test cases,

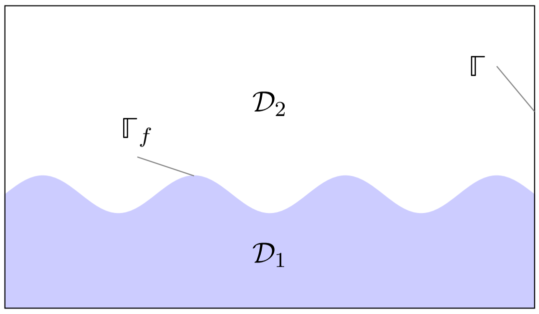


Figure 3.2: The domain $\mathcal{D}$ with boundary $\mathbb{\Gamma}$ is subdivided into two distinct fluid phase domains $\mathcal{D}_1$ and $\mathcal{D}_2$ and the fluid-fluid interface $\mathbb{\Gamma}_f$ in the two-phase Navier-Stokes equations.

cf. e.g. [SSH$^+$07]. The deterministic two-phase Navier-Stokes equations are modeled similar to e.g. [CGS09]. Further references are [SSO94, GR11, TSLV11].

3.3.1 Deterministic model

The deterministic two-phase Navier-Stokes equations are given in three dimensions, thus $\mathcal{D} \subset \mathbb{R}^3$ with $\mathcal{D}$ a connected domain with $\mathbb{\Gamma} = \partial\mathcal{D}$ its boundary. Two sub-domains $\mathcal{D}_1, \mathcal{D}_2$ identify the two fluid phases. Technically, they are time-dependent and thus functions $\mathcal{D}_1, \mathcal{D}_2 : [0,T] \to 2^{\mathcal{D}}$ with $\mathcal{D}_1(t) \cap \mathcal{D}_2(t) = \emptyset$ for all $t \in [0,T]$ and $T \in \mathbb{R}_+$ the final simulation time. The full domain $\mathcal{D}$ is covered by $\mathcal{D}_1$, $\mathcal{D}_2$ and the time-dependent fluid-fluid separation interface $\mathbb{\Gamma}_f(t)$, thus $\mathcal{D} = \mathcal{D}_1(t) \cup \mathcal{D}_2(t) \cup \mathbb{\Gamma}_f(t)$, cf. Figure 3.2. In each of the two sub-domains, thus $i = 1, 2$, the deterministic system of the two-phase Navier-Stokes equations reads as

$$\begin{aligned}
\rho_i \partial_t \boldsymbol{u}_i + \rho_i(\boldsymbol{u}_i \cdot \nabla)\boldsymbol{u}_i &= \nabla \cdot \mu_i(\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T) - \nabla p_i + \rho_i \boldsymbol{g} && \text{in } \mathcal{D}_i \times [0,T]\,, && (3.3)\\
\nabla \cdot \boldsymbol{u}_i &= 0 && \text{in } \mathcal{D}_i \times [0,T]\,, && (3.4)\\
\boldsymbol{u}_i &= \boldsymbol{u}_{0_i} && \text{in } \mathcal{D}_i \times \{0\}\,, && (3.5)\\
\mathcal{B}[\boldsymbol{a}_{\mathbb{\Gamma}}]\boldsymbol{u}_i &= \boldsymbol{b}_{\mathbb{\Gamma}} && \text{on } \mathbb{\Gamma} \times [0,T], && (3.6)\\
\frac{\partial p_i}{\partial \boldsymbol{n}_{\mathbb{\Gamma}}} &= 0 && \text{on } \mathbb{\Gamma} \times [0,T]\,, && (3.7)\\
\boldsymbol{u}_1 &= \boldsymbol{u}_2 && \text{on } \mathbb{\Gamma}_f \times [0,T]\,, && (3.8)\\
[\mathbf{T}] \cdot \boldsymbol{n}_{\mathbb{\Gamma}_f} &= \sigma\kappa \boldsymbol{n}_{\mathbb{\Gamma}_f} && \text{on } \mathbb{\Gamma}_f \times [0,T]\,. && (3.9)
\end{aligned}$$

It is solved for the velocity fields $\boldsymbol{u}_i : \mathcal{D}_i \times [0,T] \to \mathbb{R}^3$ $[m/s]$ and pressure fields $p_i : \mathcal{D}_i \times [0,T] \to \mathbb{R}$ $[kg/(m \cdot s^2)]$ with given initial conditions for the velocity field by $\boldsymbol{u}_{0_i} : \mathcal{D}_i \to \mathbb{R}^3$ and boundary conditions (3.6) and (3.7) for velocity and pressure. Velocity boundary conditions are for now denoted by some general boundary operator $\mathcal{B}$ with parameter function $\boldsymbol{a}_{\mathbb{\Gamma}} : \mathbb{\Gamma} \times [0,T] \to \mathbb{R}^s$ and the space-time-dependent right-hand side function $\boldsymbol{b}_{\mathbb{\Gamma}} : \mathbb{\Gamma} \times [0,T] \to \mathbb{R}^s$, which will be replaced by some more complex boundary domain dependent mixed-type boundary conditions in the application examples. Equation (3.7) describes homogeneous Neumann boundary conditions for the pressure field. The two important material properties for incompressible fluids are the subdomain-wise constant densities ρ_i $[kg/m^3]$ and viscosities μ_i $[kg/(m \cdot s)]$. Both fluids interact with respect to a volume force $\boldsymbol{g} \in \mathbb{R}^3$, e.g. gravity. At the fluid-fluid interface Γ_f, there exists a jump condition (3.9) for the surface stress tensor $\boldsymbol{T}_i := -p_i\boldsymbol{I} + (\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T) \in \mathbb{R}^3 \times \mathbb{R}^3$, with $[\boldsymbol{T}]$ the jump $(\boldsymbol{T}_1 - \boldsymbol{T}_2)$ across the interface. The other coupling condition between the velocities $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ is given in (3.8). Finally, $\sigma \in \mathbb{R}$ is the surface tension coefficient, $\kappa \in \mathbb{R}$ is curvature of Γ_f and $\boldsymbol{n}_{\Gamma_f} \in \mathbb{R}^3$ is the surface normal of the interface.

The *main equations* of the two-phase Navier-Stokes equations are the *momentum equation* (3.3) and the *continuity equation* (3.4). While the first one models the major part of the dynamics with the *transport term* $\rho_i(\boldsymbol{u}_i \cdot \nabla)\boldsymbol{u}_i$ and the *viscosity* or *diffusion term* $\nabla \cdot \mu_i(\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T)$, the second represents the *incompressibility constraint* for both fluids. Note that it is common to have no initial conditions for the pressure, since the pressure is usually understood as a Lagrange multiplier and the solution method applied in this thesis does not require initial conditions for pressure, cf. Section 5.1.2.

3.3.2 Velocity field boundary conditions

Boundary conditions for the velocity field were given by some general operator $\mathcal{B}$ before. These shall be replaced by a set of problem-dependent conditions. The full dynamics of transient flow simulations is usually influenced by a series of boundary conditions and their combination on different boundary subsets. The most important shall be shortly outlined here. Homogeneous Dirichlet boundary conditions

$$\boldsymbol{u}_i|_\Gamma = 0 \qquad \text{in } [0,T]$$

impose the artificial condition on the fluids to have no slip at the boundary and no mass transport over the boundary interface. (Note, that we use here Γ to indicate some subset of the full boundary, to shorten notation.) Changing the above boundary condition to infinite slip at a non-penetrated boundary leads to mixed boundary conditions

$$(\boldsymbol{u}_i \cdot \boldsymbol{n}_\Gamma)|_\Gamma = 0, \quad \left.\frac{\partial(\boldsymbol{u}_i \cdot \boldsymbol{s}_\Gamma)}{\partial \boldsymbol{n}_\Gamma}\right|_\Gamma = 0, \quad \left.\frac{\partial(\boldsymbol{u}_i \cdot \boldsymbol{t}_\Gamma)}{\partial \boldsymbol{n}_\Gamma}\right|_\Gamma = 0 \qquad \text{in } [0,T]$$

with $(\boldsymbol{u}_i \cdot \boldsymbol{n}_\Gamma)$ the velocity component normal to the boundary and $(\boldsymbol{u}_i \cdot \boldsymbol{s}_\Gamma)$, $(\boldsymbol{u}_i \cdot \boldsymbol{t}_\Gamma)$ the two respective tangential velocity field components. Furthermore, mass transport over the boundary can be implied by non-homogeneous Dirichlet boundary conditions

$$\boldsymbol{u}_i|_\Gamma = \boldsymbol{u}_{\Gamma i} \qquad \text{in } [0,T]\,.$$

Finally, Neumann-type boundary conditions in boundary normal direction, thus

$$\left.\frac{\partial \boldsymbol{u}_i}{\partial \boldsymbol{n}_\Gamma}\right|_\Gamma = 0 \qquad \text{in } [0,T]$$

impose a fluid behavior somewhat similar to the one expected at an open box boundary.

As it is possible to combine these boundary conditions in an arbitrary way, it is necessary to respect the compatibility condition [CGS09]

$$\int_\Gamma \boldsymbol{u} \cdot \boldsymbol{n}_\Gamma ds = 0$$

to have a total mass flux of zero over the boundary with respect to the combined velocity fields $\boldsymbol{u}$. For more information on the above types of boundary conditions, see also e.g. [DGN98].

3.3.3 Random model

The random model of the deterministic two-phase incompressible Navier-Stokes equations, as proposed in this thesis, introduces uncertainties to a series of parameters. For a given probability space $(\Omega, \mathcal{F}, P)$ and $i = 1, 2$, one looks for the functions $\boldsymbol{u_i} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^3$ and $p_i : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}$ such that it holds P-almost surely

$$\begin{aligned}
\rho_i(\omega)\frac{D\boldsymbol{u}_i}{Dt} &= & \nabla \cdot \mu_i(\omega)\boldsymbol{S}_i - \nabla p_i + \rho_i(\omega)\boldsymbol{g}(\omega,\boldsymbol{x}) \quad & \text{in } \Omega \times \mathcal{D}_i(\omega) \times [0,T], & (3.10)\\
\nabla \cdot \boldsymbol{u}_i &= & 0 \quad & \text{in } \Omega \times \mathcal{D}_i(\omega) \times [0,T], & (3.11)\\
\boldsymbol{u}_i &= & \boldsymbol{u}_{0_i}(\omega,\boldsymbol{x}) \quad & \text{in } \Omega \times \mathcal{D}_i(\omega) \times \{0\}, & (3.12)\\
\mathcal{B}[\boldsymbol{a}_{\mathbb{F}}(\omega)]\boldsymbol{u}_i &= & \boldsymbol{b}_{\mathbb{F}}(\omega,\boldsymbol{x}) \quad & \text{on } \Omega \times \mathbb{\Gamma} \times [0,T], & (3.13)\\
\frac{\partial p_i}{\partial \boldsymbol{n}_{\mathbb{F}}} &= & 0 \quad & \text{on } \Omega \times \mathbb{\Gamma} \times [0,T]\,, & (3.14)\\
\boldsymbol{u}_1 &= & \boldsymbol{u}_2 \quad & \text{on } \Omega \times \mathbb{\Gamma}_f(\omega) \times [0,T], & (3.15)\\
[\mathbf{T}] \cdot \boldsymbol{n} &= & \sigma\kappa\boldsymbol{n} \quad & \text{on } \Omega \times \mathbb{\Gamma}_f(\omega) \times [0,T], & (3.16)
\end{aligned}$$

with the abbreviations $\frac{D\boldsymbol{u}_i}{Dt} := \partial_t \boldsymbol{u}_i + (\boldsymbol{u}_i \cdot \nabla)\boldsymbol{u}_i$ and $\boldsymbol{S}_i := (\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T)$. Note, that the above notation shall highlight those parts of the two-phase Navier-Stokes equations, which might become subject to stochastic influence. These are the densities, viscosities, domain-dependent volume forces, velocity initial conditions, velocity boundary conditions and the distribution of the two phases over the whole domain. Of course, the velocity fields, pressure fields and derived quantities from the phase distribution become ω-dependent by this construction, too. To avoid redundancies, modeling the ω-influence on the involved quantities will be done for specific applications after introducing the finite noise assumption.

The model problem for two-phase flows can be transformed to a finite-dimensional model by introducing Karhunen-Loève expansions for the involved random fields. By doing that, we look for solution functions $\boldsymbol{u_i} : \Gamma \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^3$ and $p_i : \Gamma \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}$ such that it holds for all $\boldsymbol{y} \in \Gamma$ that

$$\begin{aligned}
\rho_i(\boldsymbol{y})\frac{D\boldsymbol{u}_i}{Dt} &= & \nabla \cdot \mu_i(\boldsymbol{y})\boldsymbol{S}_i - \nabla p_i + \rho_i(\boldsymbol{y})\boldsymbol{g}(\boldsymbol{y},\boldsymbol{x}) \quad & \text{in } \Gamma \times \mathcal{D}_i(\boldsymbol{y}) \times [0,T], & (3.17)\\
\nabla \cdot \boldsymbol{u}_i &= & 0 \quad & \text{in } \Gamma \times \mathcal{D}_i(\boldsymbol{y}) \times [0,T], & (3.18)\\
\boldsymbol{u}_i &= & \boldsymbol{u}_{0_i}(\boldsymbol{y},\boldsymbol{x}) \quad & \text{in } \Gamma \times \mathcal{D}_i(\boldsymbol{y}) \times \{0\}, & (3.19)\\
\mathcal{B}[\boldsymbol{a}_{\mathbb{F}}(\boldsymbol{y})]\boldsymbol{u}_i &= & \boldsymbol{b}_{\mathbb{F}}(\boldsymbol{y},\boldsymbol{x}) \quad & \text{on } \Gamma \times \mathbb{\Gamma} \times [0,T], & (3.20)\\
\frac{\partial p_i}{\partial \boldsymbol{n}_{\mathbb{F}}} &= & 0 \quad & \text{on } \Gamma \times \mathbb{\Gamma} \times [0,T]\,, & (3.21)\\
\boldsymbol{u}_1 &= & \boldsymbol{u}_2 \quad & \text{on } \Gamma \times \mathbb{\Gamma}_f(\boldsymbol{y}) \times [0,T], & (3.22)\\
[\mathbf{T}] \cdot \boldsymbol{n} &= & \sigma\kappa\boldsymbol{n} \quad & \text{on } \Gamma \times \mathbb{\Gamma}_f(\boldsymbol{y}) \times [0,T], & (3.23)
\end{aligned}$$

with the same abbreviations and assumptions as above. For the given equation system, three application problems will be defined.

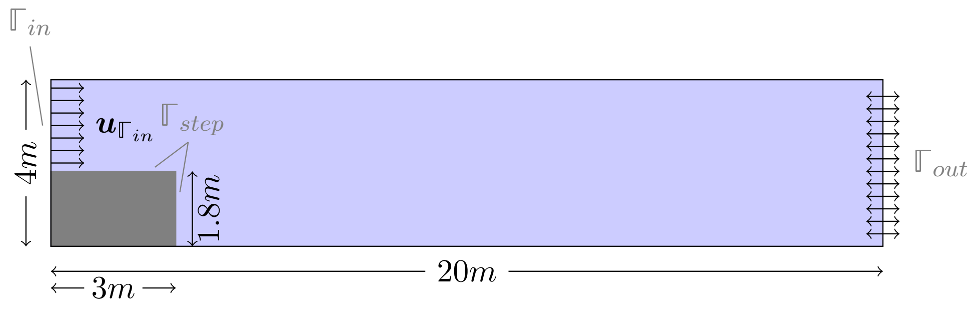


Figure 3.3: Two-dimensional side view of the three-dimensional backward facing step application.

Flow over a backward-facing step

In the design of hydraulic constructions for river systems, e.g. bridges or dams, it is important to know the structure of vortices that form close to these constructions, cf. [Dep13, EG13]. In this thesis, vortex formation shall be studied for a simplified small-scale problem in presence of uncertain inflow velocity, density and viscosity. The uncertainty might e.g. model the changing river behavior over a year, in real world. This first Navier-Stokes application is a flow over a backward-facing step. Figure 3.3 highlights the basic setup. The domain $\bar{\mathcal{D}} = [0, 20] \times [0, 4] \times [0, 2]$ is chosen, with one *meter* as basic unit for length. Inflowing fluid passes over a step with the sizes outlined in Figure 3.3 and usually forms a vortex behind that step. This is done over a period of 10 seconds, thus $T = 10$. In this simplified application, only one fluid phase is simulated. Due to $\mathcal{D}_1(\omega) = \mathcal{D}$, the sub-domains do not depend on any stochastic influence and we have $\mathcal{D}_2 = \emptyset$. Also, one-phase flows do not require the coupling conditions (3.22) and (3.23). Volume force $\boldsymbol{g}$ is set to gravity, thus $\boldsymbol{g} = (0, -9.81, 0)^\top$. With $\boldsymbol{u} \equiv \boldsymbol{u}_1$ and $\mathbb{\Gamma}_{walls} = \mathbb{\Gamma} \setminus (\mathbb{\Gamma}_{in} \cup \mathbb{\Gamma}_{out} \cup \mathbb{\Gamma}_{step})$, cf. Figure 3.3, the velocity boundary conditions replacing (3.20) are

$$
\begin{aligned}
\boldsymbol{u} =& \quad \boldsymbol{u}_{\mathbb{\Gamma}_{in}}(\boldsymbol{y}) && \text{on } \Gamma \times \mathbb{\Gamma}_{in} \times [0, T]\,, \\
\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{n}_{\mathbb{\Gamma}_{out}}} =& \quad 0 && \text{on } \Gamma \times \mathbb{\Gamma}_{out} \times [0, T]\,, \\
\boldsymbol{u} \cdot \boldsymbol{n}_{\mathbb{\Gamma}_{step}} =& \quad 0 && \text{on } \Gamma \times \mathbb{\Gamma}_{step} \times [0, T]\,, \\
\frac{\partial (\boldsymbol{u} \cdot \boldsymbol{s}_{\mathbb{\Gamma}_{step}})}{\partial \boldsymbol{n}_{\mathbb{\Gamma}_{step}}} =& \quad 0 && \text{on } \Gamma \times \mathbb{\Gamma}_{step} \times [0, T]\,, \\
\frac{\partial (\boldsymbol{u} \cdot \boldsymbol{t}_{\mathbb{\Gamma}_{step}})}{\partial \boldsymbol{n}_{\mathbb{\Gamma}_{step}}} =& \quad 0 && \text{on } \Gamma \times \mathbb{\Gamma}_{step} \times [0, T]\,, \\
\boldsymbol{u} =& \quad 0 && \text{on } \Gamma \times \mathbb{\Gamma}_{walls} \times [0, T]\,,
\end{aligned}
$$

thus the inflow condition on the boundary part $\mathbb{\Gamma}_{in}$ is under stochastic influence by the random variable $\boldsymbol{u}_{\mathbb{\Gamma}_{in}} : \Gamma \to \mathbb{R}^3$. Furthermore, we have

$$\boldsymbol{u}_{0_1} \equiv \boldsymbol{u}_{\mathbb{\Gamma}_{in}}(\boldsymbol{y})$$

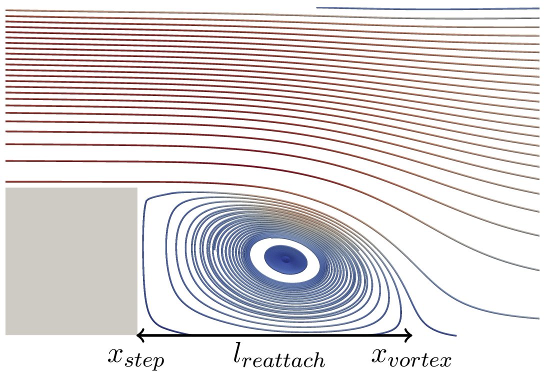


Figure 3.4: The reattachment length is the distance between the step and the end of the vortex, here shown in the visualization of a velocity field.

for the initial condition, leading to an identical inflow and uniform initial condition for the velocity field.

Finally, it is necessary to describe the random variables $\boldsymbol{u}_{\Gamma_{in}}(\boldsymbol{y})$, $\mu_1(\boldsymbol{y})$ and $\rho_1(\boldsymbol{y})$. The random inflow velocity is simplified by

$$\boldsymbol{u}_{\Gamma_{in}} = (u_{\Gamma_{in}}(\boldsymbol{y}), 0, 0)^\top \ .$$

Note that this construction differs from the one in the previous model problem in the way that no longer one single parameter is under stochastic influence. However, we have to introduce a Karhunen-Loève expansion for each of the three involved parameter functions. By truncating these after the first stochastic term, we get

$$\begin{aligned} u_{\Gamma_{in}}(\boldsymbol{y}) =& 0.55 + \frac{0.45}{\sqrt{3}} y_1^u \,, \\ \mu_1(\boldsymbol{y}) =& 0.5005 + \frac{0.4995}{\sqrt{3}} y_1^\mu \,, \\ \rho_1(\boldsymbol{y}) =& 750 + \frac{250}{\sqrt{3}} y_1^\rho \,. \end{aligned}$$

Collecting all parameters under stochastic influence leads to $\boldsymbol{y} = (y_1^u, y_1^\mu, y_1^\rho)$ and $\Gamma \subseteq \mathbb{R}^3$. Note that truncation of the random parameters after the second stochastic term would e.g. result in a six-dimensional stochastic parameter space Γ. All three variables y_1^u, y_1^μ, y_1^ρ shall correspond to independent random variables

$$Y_1^u(\omega) \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right), \quad Y_1^\mu(\omega) \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right), \quad Y_1^\rho(\omega) \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right) ,$$

thus the stochastic density function becomes $\rho(\boldsymbol{y}) = \left(\frac{1}{2\sqrt{3}}\right)^3$. Furthermore, we are either interested in the quantity of interest $\pi_1\left((\boldsymbol{u}\ \boldsymbol{p})^\top\right) = \boldsymbol{u}$ or in the vortex reattachment length $\pi_2 \equiv l_{reattach}(\boldsymbol{y}, t)$, cf. Figure 3.4. It can be formalized by assuming x_{step} to be the first component of the end point coordinate of the step and $x_{vor}(\boldsymbol{y}, t)$ the first component of the vortex' end

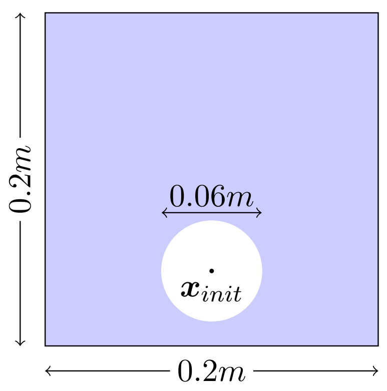


Figure 3.5: Two-dimensional side view of the three-dimensional rising bubble application setup.

point. The reattachment length $l_{reattach}$ is then given as $l_{reattach}(\boldsymbol{y},t) := x_{vor}(\boldsymbol{y},t) - x_{step}$. To be more specific, $x_{vor}(\boldsymbol{y},t)$ shall be defined as the first component of the maximum coordinate of a point on the zero level set of the first velocity field component,

$$x_{vor}(\boldsymbol{y},t) = \text{argmax}_{\boldsymbol{x}\in R(\boldsymbol{y},t)}\, x_1, \quad \text{with} \quad R(\boldsymbol{y},t) = \left\{\boldsymbol{x} \in \mathcal{D} | u_1 = 0, (u_1,u_2,u_3)^T = \boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)\right\} .$$

For $\pi_1\left((\boldsymbol{u}\ \boldsymbol{p})^\top\right) = \boldsymbol{u}$ and $\pi_2 \equiv l_{reattach}$, first and second stochastic moments might be considered.

Karhunen-Loève based random volume force in bubble flow

The second application for the Navier-Stokes equations is a two-phase flow example, namely a rising air bubble in water. To be able to study stochastic influence of a random input field with known covariance spectrum, the volume force is assumed to be random. It is approximated by a truncated Karhunen-Loève expansion. All other parameters remain deterministic. The objective is to study the mean velocity field and the center of mass of the bubble after some time.

Figure 3.5 outlines the basic setup of the two-phase flow problem. The domain is given as $\bar{\mathcal{D}} = [0, 0.2]^3$ and the fluid flow is studied until $T = 0.35$ seconds. Since an air-water system shall be analyzed, we have the densities $\rho_1 = 1000$ and $\rho_2 = 1$ and viscosities $\mu_1 = 1.002 \cdot 10^{-3}$ and $\mu_2 = 1.72 \cdot 10^{-5}$. The initial conditions are set to $\boldsymbol{u}_{0_i} = (0,0,0)^\top$. Furthermore the surface tension coefficient is set to the fixed amount of $\sigma = 0.0728$ which reflects the parameter of a water-air interface. Boundary conditions of the velocity field are introduced by replacing (3.20) with

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{n}_{\mathbb{\Gamma}} &= 0 \qquad \text{on } \Gamma \times \mathbb{\Gamma} \times [0,T]\,, \\ \frac{\partial(\boldsymbol{u} \cdot \boldsymbol{s}_{\mathbb{\Gamma}})}{\partial \boldsymbol{n}_{\mathbb{\Gamma}}} &= 0 \qquad \text{on } \Gamma \times \mathbb{\Gamma} \times [0,T]\,, \\ \frac{\partial(\boldsymbol{u} \cdot \boldsymbol{t}_{\mathbb{\Gamma}})}{\partial \boldsymbol{n}_{\mathbb{\Gamma}}} &= 0 \qquad \text{on } \Gamma \times \mathbb{\Gamma} \times [0,T]\,, \end{aligned}$$

thus an infinite slip is assumed on the boundary. Furthermore, the initial position of the bubble is $\boldsymbol{x}^{init} = (0.1, 0.06, 0.1)^\top$. The phase-wise sub-domains $\mathcal{D}_1, \mathcal{D}_2$ are given by

$$\mathcal{D}_i(t) = \Phi\left[\mathcal{D}_i^0\right](t)\,.$$

Here, Φ describes the transformation of the initial domains $\mathcal{D}_i^0 := \mathcal{D}_i(t=0)$ under fluid flow. These initial domains are defined such that the gas phase is a sphere of radius $0.03\,m$, thus

$$\mathcal{D}_1^0 := \{\boldsymbol{x} \in \mathcal{D} | \|\boldsymbol{x} - \boldsymbol{x}_{init}\| > 0.03\}\ ,$$

$$\mathcal{D}_2^0 := \{\boldsymbol{x} \in \mathcal{D} | \|\boldsymbol{x} - \boldsymbol{x}_{init}\| < 0.03\}\ .$$

The initial free surface is $\Gamma_f(t=0) = \mathcal{D} \setminus (\mathcal{D}_1^0 \cup \mathcal{D}_2^0)$.

Modeling of the random volume force $\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{x})$ is done by a truncated Karhunen-Loève expansion. The volume force is set to

$$\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{x}) := (0, g_{N_{KL}, L_c}(\boldsymbol{y}, x_2), 0)^\top\ .$$

Then, similar to the elliptic problem with Karhunen-Loève based random coefficient field, we assume g_{N_{KL}, L_c} to be the approximation of a random variable g with

$$\mathrm{Cov}\left[\log(g - (-9.81))\right](x, x') = e^{-\frac{(x-x')^2}{L_c^2}}\ ,$$

leading to the truncated Karhunen-Loève expansion

$$\log\left(g_{N_{KL}, L_c}(\boldsymbol{y}, \boldsymbol{x}) + 9.81\right) = 1 + y_1 \left(\frac{\sqrt{\pi} L_c}{2}\right)^{1/2} + \sum_{m=2}^{N_{KL}} \lambda_m \phi_m(x_2) y_m\,, \tag{3.24}$$

with truncation after N_{KL} expansion terms. The $\{y_m\}_{m=1}^{N_{KL}}$ correspond to independent random variables $\{Y_m(\boldsymbol{\omega})\}_{m=1}^{N_{KL}}$ with each $Y_m \sim \mathcal{U}(-\sqrt{3}, \sqrt{3})$, thus $\rho(\boldsymbol{y}) = \left(\frac{1}{2\sqrt{3}}\right)^{N_{KL}}$. Furthermore, the eigenvalues and eigenfunctions are given as

$$\lambda_m := (\sqrt{\pi} L_c)^{1/2} \exp\left(\frac{-(\lfloor \frac{m}{2} \rfloor \pi L_c)^2}{8}\right),\quad \phi_m(x_2) := \begin{cases} \sin(\lfloor \frac{m}{2} \rfloor \pi x_2) & \text{if } m \text{ even,} \\ \cos(\lfloor \frac{m}{2} \rfloor \pi x_2) & \text{if } m \text{ odd} \end{cases},\quad m > 1\,.$$

Usually, the correlation length is assumed to be $L_c = 2.0$.

As said before, the quantities of interest shall be $\pi_1\left((\boldsymbol{u}\ \boldsymbol{p})^\top\right) = \boldsymbol{u}$ and the center of mass of the air bubble at time t which is $\pi_{center}(t)$. Following [BS02, Section 3.1.2], with adapted notation, we have

$$\pi_{center}(t) := \frac{1}{\mathrm{Vol}(\mathcal{D}_2(t))} \int_{\mathcal{D}_2(t)} \boldsymbol{x}\, d\boldsymbol{x}\,.$$

First and second stochastic moments shall be determined.

Stochastic homogenization for rising bubbles

The third application for the Navier-Stokes equations uses a very similar setup as the previous flow example. It covers a rising air bubble in some liquid. However, now, the density, viscosity and initial bubble position are under stochastic influence. Volume forces are deterministically given. The objective is to study the mean velocity field with applications in stochastic homogenization of bubbles in a (chemical) bubble column reactor, cf. [KBU05, Jak08].

Remember that Figure 3.5 outlines the basic setup of this two-phase flow problem. The domain is given as $\bar{\mathcal{D}} = [0, 0.2]^3$ and we have $T = 0.35$. Density ρ_2 and viscosity μ_2 of the gas phase are constants set again to the properties of air, thus $\rho_2 = 1$, $\mu_2 = 1.72 \cdot 10^{-5}$. Volume force $\boldsymbol{g}$ is given as standard gravity. The initial conditions are also set to $\boldsymbol{u}_{0_i} = (0,0,0)^\top$. Furthermore, the surface tension coefficient is $\sigma = 0.0728$. The boundary conditions remain as in the previous test case.

Since the initial position of the air bubble shall be a random quantity, the phase-wise subdomains $\mathcal{D}_1, \mathcal{D}_2$ have to be interpreted as $2^{\mathcal{D}}$-valued stochastic processes. Therefore, these domains are given for time t by $\mathcal{D}_i(\boldsymbol{y}, t)$ with

$$\mathcal{D}_i(\boldsymbol{y}, t) = \Phi[\mathcal{D}_i^0(\boldsymbol{y})](t), \quad \mathcal{D}_i^0(\boldsymbol{y}) := \mathcal{D}_i(\boldsymbol{y}, 0), \quad y \in \Gamma\,,$$

where Φ again describes the transformation of the initial domains $\mathcal{D}_i^0(\boldsymbol{y})$ under fluid flow. It deterministically depends on the velocities $\boldsymbol{u}_1, \boldsymbol{u}_2$, but the initially given domains are subject to random perturbations. We define the initial liquid and gas phase domains $\mathcal{D}_1^0(\boldsymbol{y})$ and $\mathcal{D}_2^0(\boldsymbol{y})$ such that the gas phase domain is a sphere of radius $0.03\,m$ around some random initial center $\boldsymbol{x}_{init}(\boldsymbol{y})$ at the beginning, thus

$$\mathcal{D}_1^0(\boldsymbol{y}) := \{\boldsymbol{x} \in \mathcal{D} | \|\boldsymbol{x} - \boldsymbol{x}_{init}(\boldsymbol{y})\| > 0.03\}\ ,$$

$$\mathcal{D}_2^0(\boldsymbol{y}) := \{\boldsymbol{x} \in \mathcal{D} | \|\boldsymbol{x} - \boldsymbol{x}_{init}(\boldsymbol{y})\| < 0.03\}\ .$$

The initial free surface is $\Gamma_f(\boldsymbol{y}, 0) = \mathcal{D} \setminus (\mathcal{D}_1^0(\boldsymbol{y}) \cup \mathcal{D}_2^0(\boldsymbol{y}))$. Moreover, the random parameter functions $x_1^{init}(\boldsymbol{y})$, $x_2^{init}(\boldsymbol{y})$ and $x_3^{init}(\boldsymbol{y})$ with $\boldsymbol{x}_{init}(\boldsymbol{y}) = (x_1^{init}, x_2^{init}, x_2^{init})$ as well as the material parameters for the liquid phase, $\mu_1(\boldsymbol{y})$ and $\rho_1(\boldsymbol{y})$, are modeled by truncated Karhunen-Loève expansions. Truncation is done after the first stochastic term. Overall, these functions are given as

$$\begin{aligned}
x_1^{init}(\boldsymbol{y}) &= 0.1 + \frac{0.06}{\sqrt{3}} y_1^{x_1}(\omega)\,,\\
x_2^{init}(\boldsymbol{y}) &= 0.06 + \frac{0.01}{\sqrt{3}} y_1^{x_2}(\omega)\,,\\
x_3^{init}(\boldsymbol{y}) &= 0.1 + \frac{0.06}{\sqrt{3}} y_1^{x_3}(\omega)\,,\\
\mu_1(\boldsymbol{y}) &= 0.5005 + \frac{0.4995}{\sqrt{3}} y_1^{\mu_1}(\omega)\,,\\
\rho_1(\boldsymbol{y}) &= 750 + \frac{250}{\sqrt{3}} y_1^{\rho_1}(\omega)\,.
\end{aligned}$$

All parameters are collected in the five-dimensional vector $\boldsymbol{y}$ as $\boldsymbol{y} = \left(y_1^{x_1}, y_1^{x_2}, y_1^{x_3}, y_1^{\mu_1}, y_1^{\rho_1}\right)^\top$ assuming them to correspond to independent random variables for which holds

$$Y_1^{x_1} \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right), \quad Y_1^{x_2} \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right), \quad Y_1^{x_3} \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right),$$

$$Y_1^{\mu_1} \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right), \quad Y_1^{\rho_1} \sim \mathcal{U}\left(-\sqrt{3}, \sqrt{3}\right).$$

Consequently, the density function becomes $\rho(\boldsymbol{y}) = \left(\frac{1}{2\sqrt{3}}\right)^5$. Here, the quantity of interest is $\pi\left((\boldsymbol{u}\ \boldsymbol{p})^\top\right) = \boldsymbol{u}$ and we want to compute the first and second stochastic moment of that output.

4 RBF kernel-based stochastic collocation

In this chapter, the numerical method to solve the given large-scale uncertainty quantification problems is outlined. The method of choice in this thesis is non-intrusive *stochastic collocation*. Non-intrusive methods are designed to use deterministic PDE solvers as black-box to approximate random PDE problems.

Stochastic collocation methods are optimal candidates to solve random PDE problems, if the underlying deterministic PDE problem shall be accessed by point evaluations in stochastic space only. They approximate the random solution field by Lagrange interpolation between a set of deterministic solutions. In this thesis, stochastic collocation applies Lagrange bases of a reproducing kernel Hilbert space which is constructed from radial-symmetric (kernel) basis functions (RBF). Classical literature (e.g. [NTW08b, BNT10]) on stochastic collocation methods assumes a very smooth dependence of the random PDE solution field in the stochastic parameter space. In this case, RBF kernel methods with smooth kernels for interpolation promise to achieve very small approximation errors with only few point evaluations, cf. *kriging* [Mat63]. Keeping in mind that each point evaluation might be a full PDE solver run with potentially millions of unknowns, this is exactly what we need to achieve.

In the following, we start by introducing the stochastic collocation method, which delivers an approximation of the original stochastic problem by a set of deterministic problems. Since radial-symmetric kernel functions are chosen as basis, we have a short look at the underlying reproducing kernel Hilbert spaces and so-called *native spaces* in which we construct the Lagrange basis functions. This construction and popular positive definite radial kernels with standard error estimates will be discussed. Afterwards, approximations for stochastic moments of random PDE problems are derived for the new method. Collocation point sampling, quadrature and linear solvers are important ingredients to compute these approximations. Therefore, several quadrature methods including exponentially convergent tensor-product and sparse grid constructions and exact quadrature are investigated. Since quadrature methods already require sampling methods for the stochastic space, only a brief summary will be given for possible choices of collocation points. Note that by combining a given point choice with quadrature on the kernel basis functions, new quadrature formulas are constructed. Also, linear iterative and direct solvers applied to the kernel interpolation problem are considered. Since multi-GPU parallel numerical methods are important to achieve an efficient large-scale stochastic collocation method, this chapter is concluded by these implementation details.

4.1 Stochastic collocation

In stochastic collocation, we seek to numerically solve the finite-dimensional random PDE problem from (2.13). We thus look for $\boldsymbol{u} : \Gamma \times \bar{\mathcal{D}} \times [0, T] \to \mathbb{R}^r$ such that it holds for each $\boldsymbol{y} \in \Gamma$

that

$$\mathcal{L}\left(\boldsymbol{a}(\boldsymbol{y},\boldsymbol{x},t)\right)\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t) = \boldsymbol{f}\left[\boldsymbol{b}(\boldsymbol{y},\boldsymbol{x},t)\right](\boldsymbol{y},\boldsymbol{x},t) \qquad \text{in } \Gamma \times \bar{\mathcal{D}} \times [0,T]\,.$$

Following e.g. [BNT10], this is done in stochastic collocation by introducing a sampling of the stochastic space Γ with a finite set of *collocation points*

$$X_\Gamma := \left\{\boldsymbol{y}_1,\dots,\boldsymbol{y}_{N_\Gamma}\right\} \subset \Gamma\,.$$

For each of the collocation points $\boldsymbol{y}_i$, the deterministic solution $\boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)$ is evaluated, thus we solve

$$\mathcal{L}(a(\boldsymbol{y}_i,\boldsymbol{x},t))\boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t) = \boldsymbol{f}(\boldsymbol{y}_i,\boldsymbol{x},t) \qquad \text{in } \bar{\mathcal{D}} \times [0,T],\ \forall i = 1,\dots,N_\Gamma\,. \tag{4.1}$$

Then, the full solution $\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)$ is approximated by interpolating between the deterministic solution fields $\boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)$ with a Lagrange basis

$$\{L_i\}_{i=1}^{N_\Gamma}, \quad L_i \in \mathcal{P}(\Gamma), \quad L_i : \Gamma \to \mathbb{R}, \quad \text{with } L_i(\boldsymbol{y}_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

associated to the collocation points $\boldsymbol{y}_i$. Therefore, we get

$$\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t) \approx \left(\mathcal{I}_{N_\Gamma}\boldsymbol{u}\right)(\boldsymbol{y},\boldsymbol{x},t) := \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)L_i(\boldsymbol{y}) \qquad \forall \boldsymbol{y} \in \Gamma\,. \tag{4.2}$$

The approximation space, thus the space $\mathcal{P}(\Gamma)$, in which the basis functions L_i are given, will be identified with the *native space* of some radial-symmetric kernel basis function. It will be a reproducing kernel Hilbert space. Therefore, we have for the approximate solution $\mathcal{I}_{N_\Gamma}\boldsymbol{u}$ of stochastic collocation that

$$\mathcal{I}_{N_\Gamma}\boldsymbol{u} \in \mathcal{P}(\Gamma) \otimes L^2([0,T];L^2(\mathcal{D}))\,.$$

4.2 Reproducing kernel Hilbert spaces and native spaces

In order to understand the basic approximation properties of the kernel-based RBF stochastic collocation method and the applied approximation space $\mathcal{P}(\Gamma)$, a brief introduction into reproducing kernel Hilbert spaces is given. To do that, we will closely follow the lines of [Wen04] and cite important definitions and results with an adapted notation

Theory of *reproducing kernel Hilbert spaces* starts with an underlying Hilbert space

$$\mathscr{F} \subseteq \left\{ f : \Gamma \to \mathbb{R} \,\middle|\, \emptyset \neq \Gamma \subseteq \mathbb{R}^d \right\}$$

of functions that we like to approximate. We can define reproducing kernels by

Definition 4.1 (Reproducing kernels [Wen04, Definition 10.1])**.** *Let $\mathscr{F}$ be a Hilbert space of functions $f : \Gamma \to \mathbb{R}$. A function $k : \Gamma \times \Gamma \to \mathbb{R}$ is called* reproducing kernel *for $\mathscr{F}$ if*

1. $k(\cdot, \boldsymbol{y}) \in \mathscr{F}$ *for all* $\boldsymbol{y} \in \Gamma$,
2. $f(\boldsymbol{y}) = (f, k(\cdot, \boldsymbol{y}))_{\mathscr{F}}$ *for all* $f \in \mathscr{F}$ *and all* $\boldsymbol{y} \in \Gamma$.

The second requirement, the reproduction formula, tells us that all functions can be pointwise evaluated by kernel k.

Definition 4.2 (Reproducing kernel Hilbert spaces)**.** *A Hilbert space* $\mathscr{F}$ *of functions* $f : \Gamma \to \mathbb{R}$ *is called* reproducing kernel Hilbert space *if it has a reproducing kernel* $k : \Gamma \times \Gamma \to \mathbb{R}$.

Next we have to understand the concept of positive definiteness for kernels, which is closely related to the usual definition in the matrix case.

Definition 4.3 (Positive (semi-)definite kernels [Wen04, Definition 6.24])**.** *A continuous kernel* $k : \Gamma \times \Gamma \to \mathbb{R}$ *is called positive semi-definite on* $\Gamma \subseteq \mathbb{R}^d$ *if for all* $N \in \mathbb{N}$, *all pairwise distinct* $X = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subseteq \Gamma$, *and all* $\alpha \in \mathbb{R}^N \setminus \{0\}$ *we have*

$$\sum_{j=1}^{N} \sum_{k=1}^{N} \alpha_j \alpha_k \, k(\boldsymbol{y}_j, \boldsymbol{y}_k) \geq 0.$$

It is called positive definite if the left-hand side of the equation is additionally non-zero.

Until now, we only have very little information on the actual structure of the kernels to be applied. However, it is desirable to have kernels with some invariance properties. As earlier said, radial basis functions / radial kernels shall be applied here. They can be derived by a simple construction outlined in [Wen04]. One starts by formally introducing the concept of invariance.

Definition 4.4 ([Wen04, Definition 10.5])**.** *Let* $\mathcal{T}$ *be a group of transformations* $T : \Gamma \to \Gamma$. *We say* $\mathscr{F}$ *is invariant under the group* $\mathcal{T}$ *if*

1. $f \circ T \in \mathscr{F}$ *for all* $f \in \mathscr{F}$ *and* $T \in \mathcal{T}$,
2. $(f \circ T, g \circ T)_{\mathscr{F}} = (f, g)_{\mathscr{F}}$ *for all* $f, g \in \mathscr{F}$ *and all* $T \in \mathcal{T}$.

Taking this definition, [Wen04] states the invariance property for the kernel in invariant kernel Hilbert spaces as

Theorem 4.1 ([Wen04, Theorem 10.6])**.** *Suppose that the reproducing kernel Hilbert function space is invariant under the transformations of* $\mathcal{T}$, *then the reproducing kernel* k *satisfies*

$$k(T\boldsymbol{y}, T\boldsymbol{y}') = k(\boldsymbol{y}, \boldsymbol{y}')$$

for all $\boldsymbol{y}, \boldsymbol{y}' \in \Gamma$ *and all* $T \in \mathcal{T}$.

The following example formally constructs appropriate radial kernels.

Example 4.1 (Radial kernels [Wen04, Section 10.1])**.** *Let* $\Gamma = \mathbb{R}^d$ *and* $\mathcal{T}$ *is the set of translations and orthogonal transformations. Further,* $A \in \mathbb{R}^{d \times d}$ *are orthogonal transformations such that* $A\xi = ||\xi||_2 \boldsymbol{e}_1, \xi = \boldsymbol{y} - \boldsymbol{y}'$, *with* $\boldsymbol{e}_1$ *the first unit vector. With the above theorem, we get*

$$k(\boldsymbol{y}, \boldsymbol{y}') = k(A\boldsymbol{y}, A\boldsymbol{y}') = k(0, A(\boldsymbol{y} - \boldsymbol{y}')) =: k_0(A(\boldsymbol{y} - \boldsymbol{y}')) = k_0(||\boldsymbol{y} - \boldsymbol{y}'||_2 \boldsymbol{e}_1) =: \varphi(||\boldsymbol{y} - \boldsymbol{y}'||_2)\,.$$

Thus, we can see that k is actually radial under the above assumptions.

Later on, one might use problem dependent norms instead of $||\cdot||_2$, but the actual construction will be the same. This motivates us to stick to this norm throughout this chapter. Note that we have implicitly used the notion of a *radial* function which we can formalize in

Definition 4.5 (Radial function [Wen04, Definition 6.15])**.** *A function $\Phi : \mathbb{R}^d \to \mathbb{R}$ is said to be radial if there exists a function $\varphi : [0, \infty) \to \mathbb{R}$ such that $\Phi(\boldsymbol{y}) = \varphi(\|\boldsymbol{y}\|_2)$ for all $\boldsymbol{y} \in \mathbb{R}^d$.*

After having introduced reproducing kernel Hilbert spaces in general, we will now have a look at the concept of so-called *native spaces*. In fact, we are looking for Hilbert spaces that are constructed from the span of a kernel k function. It will turn out that these spaces are again reproducing kernel Hilbert spaces and the natural choice of space to perform our approximations in.

The starting point for the native space construction is a given symmetric, positive definite kernel $k : \Gamma \times \Gamma$ for which we construct the pre-Hilbert space

$$F_k(\Gamma) := \operatorname{span}\{k(\cdot, \boldsymbol{y}) \,|\, \boldsymbol{y} \in \Gamma\}$$

with scalar product

$$\left(\sum_{j=1}^{N} \alpha_j \, k(\cdot, \boldsymbol{y}_j) \ , \ \sum_{k=1}^{N} \beta_k \, k(\cdot, \boldsymbol{y}'_k) \right)_k := \sum_{j=1}^{N} \sum_{k=1}^{N} \alpha_j \beta_k \, k(\boldsymbol{y}_j, \boldsymbol{y}'_k)$$

as stated in the following theorem.

Theorem 4.2 ([Wen04, Theorem 10.7])**.** *If $k : \Gamma \times \Gamma \to \mathbb{R}$ is a symmetric positive definite kernel, then $(\cdot, \cdot)_k$ defines an inner product on $F_k(\Gamma)$. Furthermore, $F_k(\Gamma)$ is a pre-Hilbert space with reproducing kernel k.*

$F_k(\Gamma)$ can now be extended to a full Hilbert space $\mathscr{F}_k(\Gamma)$ as a completation of $F_k(\Gamma)$ with respect to the norm $|| \cdot ||_k$ induced by the given scalar product. For a formal evaluation of function values even in the completion, a mapping

$$R : \mathscr{F}_k(\Gamma) \to C(\Gamma) , \qquad R(f)(\boldsymbol{x}) := (f, k(\cdot, \boldsymbol{x}))_k$$

can be introduced. With that, we have everything to formally define a native space by

Definition 4.6 (Native space [Wen04, Definition 10.9])**.** *The native Hilbert function space corresponding to the symmetric positive definite kernel $k : \Gamma \times \Gamma \to \mathbb{R}$ is defined by*

$$\mathcal{N}_k(\Gamma) := \overline{R(\mathscr{F}_k(\Gamma))} \, .$$

It carries the inner product

$$(f, g)_{\mathcal{N}_k(\Gamma)} := \left(R^{-1} f, R^{-1} g \right)_k .$$

As initially proposed, these native spaces are reproducing kernel Hilbert spaces with the reproducing kernel from which they were generated:

Theorem 4.3 ([Wen04, Theorem 10.10]). *Suppose that $k : \Gamma \times \Gamma \to \mathbb{R}$ is a symmetric positive definite kernel. Then its associated native space $\mathcal{N}_k(\Gamma)$ is a Hilbert function space with reproducing kernel k.*

With these definitions and results at hand, will in the following require that the approximation space $\mathcal{P}(\Gamma)$ for the stochastic collocation approximation (4.2) is given as

$$\mathcal{P}(\Gamma) := \mathcal{N}_k(\Gamma) .$$

The kernel function k will be chosen appropriately. Remembering the stochastic collocation problem from (4.1) with its general solution space $\mathcal{P}(\Gamma) \otimes L^2([0,T]; L^2(\mathcal{D}))$, we now choose to compute approximations with respect to a kernel native space. Therefore, we get

$$\mathcal{I}_{N_\Gamma} \boldsymbol{u} \in \mathcal{N}_k(\Gamma) \otimes L^2([0,T]; L^2(\mathcal{D})) .$$

4.3 Interpolation with kernels and error estimates

While we introduced some basic terminology for reproducing kernel Hilbert spaces, we did not yet come to the main point of interest, which is the interpolation by kernel functions or by the Lagrange basis. In the following, we will first investigate the idea of interpolation and then sum up interpolation error results concerning kernel functions which are of interest for this thesis.

4.3.1 Interpolation in native spaces

Definition 4.7 (Interpolation with strictly positive definite kernel functions [Wen04, p. 64,82]). *Let a strictly positive definite kernel function k with its associated native space $\mathcal{N}_k(\Gamma)$ be given. Furthermore, we have a function $f : \Gamma \to \mathbb{R}$, from that native space, that is evaluated at a finite set X of distinct collocation points such that*

$$X := \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\} \subset \Gamma , \qquad f_j := f(\boldsymbol{y}_j) \quad \forall j = 1, \dots, N_\Gamma .$$

Then, the interpolant $s_{f,X} \in \mathcal{N}_k(\Gamma)$ *is defined as*

$$s_{f,X}(\boldsymbol{y}) := \sum_{j=1}^{N} \alpha_j k(\boldsymbol{y}, \boldsymbol{y}_j) \qquad \forall \boldsymbol{y} \in \Gamma , \tag{4.3}$$

with the interpolation condition *to describe f exactly at the collocation points*

$$s_{f,X}(\boldsymbol{y}_j) = f_j , \qquad 1 \le j \le N . \tag{4.4}$$

Finding coefficients $\{\alpha_j\}_{j=1}^N$, $\alpha_j \in \mathbb{R}$ such that the interpolation condition (4.4) is fulfilled, is the interpolation problem.

From a computational point of view, we can rewrite the interpolation condition such that we have to solve a system of linear equations with

$$A_{k,X} \boldsymbol{\alpha} = \boldsymbol{f} , \tag{4.5}$$

$$\boldsymbol{\alpha} := (\alpha_1 \dots \alpha_N)^\top , \quad \boldsymbol{f} := (f_1 \dots f_N)^\top$$

and the Gram-type interpolation matrix

$$A_{k,X} := \begin{pmatrix} k(\boldsymbol{y}_1, \boldsymbol{y}_1) & \dots & k(\boldsymbol{y}_1, \boldsymbol{y}_N) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{y}_N, \boldsymbol{y}_1) & \dots & k(\boldsymbol{y}_N, \boldsymbol{y}_N) \end{pmatrix} . \tag{4.6}$$

Obviously, symmetric and radial kernel functions lead to symmetric matrices $A_{k,X}$ and positive definite kernels result in positive definite interpolation matrices.

Regularization

Depending on the applied kernels and the choice and quantity of collocation points, the interpolation matrix $A_{k,X}$ may become very ill-conditioned. This might introduce large errors in the interpolation even if direct linear solvers are used to solve the system. One approach to overcome this issue is the introduction of a regularization. A standard technique in support vector machine learning is the *Tikhonov regularization* [TA77], which is based on a minimization problem with regularization term and would require to solve a modified version of equation (4.5),

$$(A_{k,X}{}^\top A_{k,X} + \epsilon_{reg}\, I_N)\boldsymbol{\alpha} = A_{k,X}{}^\top \boldsymbol{f} ,$$

with I the identity matrix. However, since we here only discuss kernel functions with symmetric positive definite interpolation matrices, the simplified *Lavrentiev regularization* [Lav67] can be applied, which omits to square the interpolation matrix. It is enough to solve

$$(A_{k,X} + \epsilon_{reg}\; I_N)\boldsymbol{\alpha} = \boldsymbol{f} , \tag{4.7}$$

thus one adds a small constant value to the diagonal of the interpolation matrix. The resulting equation (4.7) is no longer an interpolation problem, but a regression problem. To shorten the discussion in [RZ09], this regularization reduces the condition number, but introduces a new error in the order of the regularization parameter ϵ_{reg}.

Evaluation

The evaluation of the interpolant $s_{f,X}$ with

$$s_{f,X} : \boldsymbol{y} \mapsto \sum_{i=1}^{N} \alpha_i k(\boldsymbol{y}, \boldsymbol{y}_i)$$

at a set of points $X' = \{\boldsymbol{y}'_1, \dots, \boldsymbol{y}'_{N'}\}$ can be translated to the linear algebra operation of applying the matrix vector product

$$\boldsymbol{s} = A_{k,X',X}\, \boldsymbol{\alpha}$$

with $\boldsymbol{s} = \left(s_{f,X}\left(\boldsymbol{y}'_1\right) \dots s_{f,X}\left(\boldsymbol{y}'_{N'}\right)\right)^\top$ and

$$A_{k,X',X} := \begin{pmatrix} k\left(\boldsymbol{y}'_1, \boldsymbol{y}_1\right) & \dots & k\left(\boldsymbol{y}'_1, \boldsymbol{y}_N\right) \\ \vdots & \ddots & \vdots \\ k\left(\boldsymbol{y}'_{N'}, \boldsymbol{y}_1\right) & \dots & k\left(\boldsymbol{y}'_{N'}, \boldsymbol{y}_N\right) \end{pmatrix} \in \mathbb{R}^{N' \times N} .$$

Lagrange basis

In Section 4.1 we discussed Lagrange basis functions

$$\{L_i\}_{i=1}^{N_\Gamma}, \quad L_i : \Gamma \to \mathbb{R}, \quad \text{with } L_i(\boldsymbol{y}_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{4.8}$$

and proposed to use radial kernels k to construct these functions. In fact, we want the Lagrange basis functions to belong to the native space of some kernel k, thus

$$L_i \in \mathcal{N}_k(\Gamma)\,.$$

Therefore, the Lagrange basis functions can be expressed by

$$L_i(\boldsymbol{y}) = \sum_{j=1}^{N_\Gamma} \alpha_j^i k(\boldsymbol{y}, \boldsymbol{y}_j) \qquad\qquad \forall i = 1 \dots N_\Gamma\,. \tag{4.9}$$

To compute these, or especially the coefficients α_j^i, we can proceed as before and solve an interpolation problem with the interpolation condition

$$L_i(\boldsymbol{y}_j) = \delta_{ij} \qquad\qquad \forall i, j = 1, \dots, N_\Gamma\,,$$

leading to the set of linear systems

$$A_{k,X_\Gamma} A_L = I_{N_\Gamma} \tag{4.10}$$

with A_L the matrix of coefficients $A_L := \left(\alpha_j^i\right)_{j,i=1}^{N_\Gamma}$ and $I_{N_\Gamma} \in \mathbb{R}^{N_\Gamma \times N_\Gamma}$ the identity matrix. From (4.10), we learn that the coefficients α_j^i can in fact be derived by inverting the interpolation matrix A_{k,X_Γ}.

4.3.2 Important kernel functions with error estimates

Next, we will discuss a series of kernel functions which will become important throughout this thesis. To understand their approximation properties, estimates for interpolation errors

$$e_{f,X} := \|f - s_{f,X}\| \tag{4.11}$$

in some appropriate norm are cited from the literature. But before we start, we have to introduce a bit more terminology. The fill distance $h_{X,\Gamma}$ generalizes the typical mesh width h from mesh based methods to general point sets. It is given in

Definition 4.8 (Fill distance [Wen04, p. 25])**.** *For a set of points $X = \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\}$ in a bounded domain $\Gamma \subseteq \mathbb{R}^d$ the fill distance is defined to be*

$$h_{X,\Gamma} = \sup_{\boldsymbol{y} \in \Gamma} \min_{1 \le j \le N} \|\boldsymbol{y} - \boldsymbol{y}_j\|_2\,.$$

Error estimates will usually be given for functions which are elements of the native space of the kernel. The function's native space norm is one ingredient for the upper bound of the estimates. Even though we implicitly introduced the norm of the native space already in Section 4.2 we will use the following definition to sum up the details:

Definition 4.9 (Norm of native space $\mathcal{N}_k(\Gamma)$)**.** *Let the native space $\mathcal{N}_k(\Gamma)$ with kernel function k be given. Suppose we have for a function $f \in \mathcal{N}_k(\Gamma)$ the representation $f(\boldsymbol{y}) = \sum_{j=1}^{N_f} \alpha_j k(\boldsymbol{y}, \boldsymbol{y}_j)$. The* native space norm *of that function is then defined as*

$$\|f\|_{\mathcal{N}_k(\Gamma)} := \sqrt{(f, f)_{\mathcal{N}_k(\Gamma)}} := \left(\sum_{j=1}^{N_f} \sum_{j'=1}^{N_f} \alpha_j \alpha_{j'} k(\boldsymbol{y}_j, \boldsymbol{y}_{j'}) \right)^{1/2} .$$

Gaussian kernel

Definition 4.10 (Gaussian kernel [Wen04, Theorem 11.22, Section 11.4])**.** *The* Gaussian kernel *$k_\epsilon : \Gamma \times \Gamma \to \mathbb{R}$ is given by*

$$k_\epsilon(\boldsymbol{y}, \boldsymbol{y}') := \varphi_\epsilon(\|\boldsymbol{y} - \boldsymbol{y}'\|) := e^{-\epsilon^2 \|\boldsymbol{y} - \boldsymbol{y}'\|^2}$$

with $\epsilon \in \mathbb{R}^+$ a scaling parameter.

By construction, this kernel has no compact support. Note that kernel-based interpolation with Gaussian kernels has a close relationship to *kriging* [Mat63] and Gaussian process regression [RW05]. Using [Wen04, Theorem 11.22, p. 190f] and the fact that Gaussian kernels are positive definite, we get

Theorem 4.4 (Error estimate for Gaussian kernels)**.** *Let Γ be a cube in $\mathbb{R}^d$. For the Gaussian kernel $k_\epsilon(\boldsymbol{y}, \boldsymbol{y}') := \varphi_\epsilon(\|\boldsymbol{y} - \boldsymbol{y}'\|)$ exists a constant $c > 0$ such that the error between a function $f \in \mathcal{N}_{k_\epsilon}(\Gamma)$ and its interpolant $s_{f,X}$ can be bounded by*

$$\|f - s_{f,X}\|_{L_\infty(\Gamma)} \leq e^{c \log(h_{X,\Gamma})/h_{X,\Gamma}} \|f\|_{\mathcal{N}_{k_\epsilon}(\Gamma)} ,$$

for $h_{X,\Gamma}$ sufficiently small.

In other words, Gaussian kernels achieve even a bit better than point-wise exponential convergence if the function f stems from the kernel's associated native space.

Wendland kernels

Wendland kernels are compactly supported functions with some minimality properties for the degrees of the polynomials involved in their construction. They have been introduced in [Wen95]. Also, [Wen04, Chapter 9] has an in-depth introduction. We stick here to a small selection of Wendland kernels given in

Definition 4.11 (Wendland kernels [Wen04, Theorem 9.13, Corollary 9.14])**.** Wendland kernels *are given by*

$$k_{d,k}(\boldsymbol{y}, \boldsymbol{y}') := \varphi_{d,k}(\|\boldsymbol{y} - \boldsymbol{y}'\|) ,$$

where $\boldsymbol{y}, \boldsymbol{y}' \in \Gamma \subseteq \mathbb{R}^d$. The $k_{d,k}$ are positive definite and it holds $\varphi_{d,k} \in C^{2k}(\mathbb{R})$. For $k = 0, 1, 2, 3$ we have the explicit formulas

$$\begin{aligned}
\varphi_{d,0}(r) &= (1-r)_+^{\lfloor d/2 \rfloor + 1}\,, \\
\varphi_{d,1}(r) &= (1-r)_+^{\ell+1}[(\ell+1)r + 1]\,, \\
\varphi_{d,2}(r) &= (1-r)_+^{\ell+2}[(\ell^2 + 4\ell + 3)r^2 + (3\ell + 6)r + 3]\,, \\
\varphi_{d,3}(r) &= (1-r)_+^{\ell+3}[(\ell^3 + 9\ell^2 + 23\ell + 15)r^3 + (6\ell^2 + 36\ell + 45)r^2 + (15\ell + 45)r + 15]\,,
\end{aligned}$$

with $\ell := \lfloor d/2 \rfloor + k + 1$ and the notation

$$(r)_+ = \begin{cases} r & \textit{if } r \geq 0\,, \\ 0 & \textit{if } r < 0\,. \end{cases}$$

Due to their compact support, their interpolation matrix becomes sparse. This has some advantages in terms of computational complexity. Again, we have an error estimate for interpolation of functions in the associated native space.

Theorem 4.5 (Error estimate for Wendland kernels [Wen04, Theorem 11.17])**.** *For $k_{d,k}(\boldsymbol{x}, \boldsymbol{y}) := \varphi_{d,k}(\|\boldsymbol{x} - \boldsymbol{y}\|_2)$ with $\Gamma \subseteq \mathbb{R}^d$ bounded and satisfying an interior cone condition and $f \in \mathcal{N}_{k_{d,k}}(\Gamma)$, there exist constants $C, h_0 > 0$ such that*

$$|D^{\boldsymbol{\alpha}} f(\boldsymbol{y}) - D^{\boldsymbol{\alpha}} s_{f,X}(\boldsymbol{y})| \leq C h_{X,\Gamma}^{k+1/2-|\boldsymbol{\alpha}|} \|f\|_{\mathcal{N}_{k_{d,k}}(\Gamma)}$$

for arbitrary $\boldsymbol{\alpha} \in \mathbb{N}_0^d$ with $|\boldsymbol{\alpha}| \leq k$ and all $\boldsymbol{y} \in \Gamma$, as long as $h_{X,\Gamma} \leq h_0$.

Since we did not yet introduce the concept of an interior cone condition, we catch up on this now by

Definition 4.12 (Interior cone condition [Wen04, Definition 3.6])**.** *A set $\Gamma \subseteq \mathbb{R}^d$ is said to satisfy an interior cone condition if there exists an angle $\theta \in (0, \pi/2)$ and a radius $r > 0$, such that for every $\boldsymbol{y} \in \Gamma$ a unit vector $\boldsymbol{\xi}(\boldsymbol{y})$ exists, such that the cone*

$$C(\boldsymbol{y}, \boldsymbol{\xi}(\boldsymbol{y}), \theta, r) := \left\{ \boldsymbol{y} + \lambda \boldsymbol{y}' : \boldsymbol{y}' \in \mathbb{R}^d,\, \|\boldsymbol{y}'\|_2 = 1,\, {\boldsymbol{y}'}^\top \boldsymbol{\xi}(\boldsymbol{y}) \geq \cos\theta,\, \lambda \in [0, r] \right\}$$

is contained in Γ.

In the case of Wendland kernels, we also have a clear idea of how their associated native spaces look like:

Theorem 4.6 ([Wen04, Theorem 10.35])**.** *Let the Wendland kernels be defined as above. In the case of $k = 0$, we assume to have $d \geq 3$, otherwise there is no restriction. Then, the corresponding native space is a Sobolev space. More precisely, it is*

$$\mathcal{N}_{k_{d,k}}(\mathbb{R}^d) = H^{d/2+k+1/2}(\mathbb{R}^d)\,.$$

Consequently, the error estimate for Wendland kernels is a statement for functions f in some Sobolev space.

Matérn kernels

Other kernels which will be considered in this thesis are the Matérn kernels. They can be given as follows:

Definition 4.13 (Matérn kernels [Fas07, Section 4.4])**.** Matérn kernels *are defined as*

$$k_\beta(\boldsymbol{y}, \boldsymbol{y}') := \frac{K_{\beta-\frac{d}{2}}(\|\boldsymbol{y}-\boldsymbol{y}'\|)\|\boldsymbol{y}-\boldsymbol{y}'\|^{\beta-\frac{d}{2}}}{2^{\beta-1}\Gamma(\beta)}, \quad \beta > \frac{d}{2},$$

where K_ν is the modified Bessel function of the second kind of order ν and Γ the gamma function.

These kernels are strictly positive definite as long as $d < 2\beta$. According to [Fas07, Section 4.4], we have for special choices of parameter β, simplified representations of the Matérn kernel function (up to a dimension-dependent scaling constant) as

$$\begin{aligned}
k_{\frac{d+1}{2}}(\boldsymbol{y}, \boldsymbol{y}') &:= e^{-\|\boldsymbol{y}-\boldsymbol{y}'\|}, \\
k_{\frac{d+3}{2}}(\boldsymbol{y}, \boldsymbol{y}') &:= (1 + \|\boldsymbol{y}-\boldsymbol{y}'\|)\, e^{-\|\boldsymbol{y}-\boldsymbol{y}'\|}, \\
k_{\frac{d+5}{2}}(\boldsymbol{y}, \boldsymbol{y}') &:= \left(3 + 3\|\boldsymbol{y}-\boldsymbol{y}'\| + \|\boldsymbol{y}-\boldsymbol{y}'\|^2\right) e^{-\|\boldsymbol{y}-\boldsymbol{y}'\|}.
\end{aligned}$$

For technical reasons, thus a missing optimized implementation of the modified Bessel function of second kind on GPUs, the latter Matérn kernel functions will be used throughout this thesis.

As for the other kernel functions, we can have an error estimate in terms of the fill distance and functions from native space. It reads as

Theorem 4.7 (Error estimate for Matérn kernels [Fas07, Theorem 14.5, Example 15.4])**.** *Let $\Gamma \subseteq \mathbb{R}^d$ be bounded satisfying an interior cone condition. There exist positive constants h_0 and C (independent of $\boldsymbol{y}$ and f) such that it holds for Matérn kernels k_β that*

$$|D^{\boldsymbol{\alpha}} f(\boldsymbol{y}) - D^{\boldsymbol{\alpha}} s_{f,X}(\boldsymbol{y})| \le C h_{X,\Gamma}^{\beta-d/2-|\boldsymbol{\alpha}|} \|f\|_{\mathcal{N}_{k_{d,k}}(\Gamma)}$$

as long as $\boldsymbol{\alpha} \in \mathbb{N}_0^d$, $|\boldsymbol{\alpha}| \le \beta - \lceil \frac{d+1}{2} \rceil$, $h_{X,\Gamma} \le h_0$ and $f \in \mathcal{N}_{k_\beta}(\Gamma)$.

Here, again we can get some intuition on the native space, if we have a look at

Theorem 4.8 ([Fas07, p. 109])**.** *Let be the Matérn kernels defined as above and $\beta > d/2$. With $\Gamma = \mathbb{R}^d$, the corresponding native space is a Sobolev space with*

$$\mathcal{N}_{k_\beta}(\mathbb{R}^d) = W^{\beta,2}(\mathbb{R}^d).$$

Note again that we always assume here that the interpolated function is in the native space of the applied kernel basis function. However, this does not necessarily have to be true for the response function of the model and application random PDE problems in this thesis. Moreover, in case of the strong solution for the two-phase Navier-Stokes equations, no regularity theory might exist at all. In some cases, it is possible to derive error estimates for functions which do not stem from the corresponding native space, cf. e.g. [Fas07, Section 15.3]. Nonetheless, in this thesis, errors in the solution fields are usually analyzed by empirical means. The error estimates given here, are used to get a rough idea of the potential approximation order of a given kernel function.

4.4 Estimation of stochastic moments

Let us remember from Definition 2.21 two important stochastic moments for the random-coefficient PDE problem, thus the expectation value

$$\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x},t) := \int_\Gamma \boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)\rho(\boldsymbol{y})d\boldsymbol{y}$$

and the covariance, which can be expressed as

$$\operatorname{Cov}\left[\boldsymbol{u}\right](\boldsymbol{x},\boldsymbol{x}',t) = \mathbb{E}\left[\boldsymbol{u}(\cdot,\boldsymbol{x},t)\boldsymbol{u}(\cdot,\boldsymbol{x}',t)\right] - \mathbb{E}\left[\boldsymbol{u}(\cdot,\boldsymbol{x},t)\right]\mathbb{E}\left[\boldsymbol{u}(\cdot,\boldsymbol{x}',t)\right] . \tag{4.12}$$

To estimate the first stochastic moment of the solution of a random PDE by stochastic collocation, we remember equations (2.10) – (2.12) and the definition of the approximate solution of the random PDE problem by the stochastic collocation method in equation (4.2). We get

$$\begin{aligned}\mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x},t) &= \int_\Gamma \boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t)\rho(\boldsymbol{y})d\boldsymbol{y}\\ &\approx \int_\Gamma (\mathcal{I}_{N_\Gamma}\boldsymbol{u})(\boldsymbol{y},\boldsymbol{x},t)\rho(\boldsymbol{y})d\boldsymbol{y}\\ &= \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)\int_\Gamma L_i(\boldsymbol{y})\rho(\boldsymbol{y})d\boldsymbol{y}\\ &= \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)\sum_{j=1}^{N_\Gamma}\alpha_j^i\int_\Gamma k(\boldsymbol{y},\boldsymbol{y}_j)\rho(\boldsymbol{y})d\boldsymbol{y}\\ &= \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t)\sum_{j=1}^{N_\Gamma}\alpha_j^i\mathbb{E}\left[k(\cdot,\boldsymbol{y}_j)\right](\boldsymbol{y}_j) .\end{aligned}$$

Here, we use linearity of the expectation value operator. The above formulation can be simplified, if we combine the second sum in the last equation into vector $\boldsymbol{g} \in \mathbb{R}^{N_\Gamma}$ with

$$\boldsymbol{g} = (g_1 \dots g_{N_\Gamma})^\top , \quad g_i := \sum_{j=1}^{N_\Gamma}\alpha_j^i\mathbb{E}\left[k(\cdot,\boldsymbol{y}_j)\right](\boldsymbol{y}_j) .$$

From (4.9), we know that the α_j^i's are the entries of matrix A_L with $A_{k,X_\Gamma}A_L = I_N$, thus we have

$$A_L = {A_{k,X_\Gamma}}^{-1} .$$

Algorithm 1 Estimation of first moment

```
Require: positive definite kernel $k$
1: function EXPECTATION($\boldsymbol{x}, t$)
2:     construct $X_\Gamma := \left\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_{N_\Gamma}\right\}$
3:     for $i = 1, 2, \dots, N_\Gamma$ do
4:         evaluate PDE for $\boldsymbol{y_i} \to \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)$
5:     estimate $\boldsymbol{e} = (e_j)_{j=1}^{N_\Gamma}$: $e_j = \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)\right](\boldsymbol{y}_j)$
6:     solve $A_{k,X_\Gamma}\boldsymbol{g} = \boldsymbol{e}$, $\boldsymbol{g} = (g_i)_{i=1}^{N_\Gamma}$
7:     return $\sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t) g_i$
```

By introducing the vector of expectation values of the basis functions $k(\cdot, \boldsymbol{y}_j)$

$$\boldsymbol{e} = (e_1 \dots e_{N_\Gamma})^\top \,, \quad e_j := \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)\right](\boldsymbol{y}_j)\,,$$

we get $\boldsymbol{g} = {A_{k,X_\Gamma}}^{-1}\boldsymbol{e}$ and can approximate $\boldsymbol{g}$ by numerically solving the linear system

$$A_{k,X_\Gamma}\boldsymbol{g} = \boldsymbol{e}\,.$$

Therefore, the first stochastic moment is thereafter given by

$$\mathbb{E}\left[u\right](\boldsymbol{x}, t) \approx \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t) g_i\,. \tag{4.13}$$

Algorithm 1 sums up the necessary steps to estimate the first moment of a random-coefficient PDE problem.

Analogously, we can construct the estimate for the covariance. Since we already know, how to evaluate the expectation value, we only need to approximate the correlation, which is $\mathbb{E}\left[\boldsymbol{u}(\cdot, \boldsymbol{x}, t)\boldsymbol{u}(\cdot, \boldsymbol{x}', t)\right](\boldsymbol{x}, \boldsymbol{x}', t)$. By linearity, we get

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{u}(\cdot, \boldsymbol{x}, t)\boldsymbol{u}(\cdot, \boldsymbol{x}', t)\right](\boldsymbol{x}, \boldsymbol{x}', t) &= \int_\Gamma \boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x}', t)\rho(\boldsymbol{y})d\boldsymbol{y} \\
&\approx \sum_{i=1}^{N_\Gamma}\sum_{i'=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}_{i'}, \boldsymbol{x}', t) \int_\Gamma L_i(\boldsymbol{y})L_{i'}(\boldsymbol{y})\rho(\boldsymbol{y})d\boldsymbol{y} \\
&\approx \sum_{i=1}^{N_\Gamma}\sum_{i'=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}_{i'}, \boldsymbol{x}', t) \sum_{j=1}^{N_\Gamma}\sum_{j'=1}^{N_\Gamma} \alpha_j^i \alpha_{j'}^{i'} \int_\Gamma k(\boldsymbol{y}, \boldsymbol{y}_j)k(\boldsymbol{y}, \boldsymbol{y}_{j'})\rho(\boldsymbol{y})d\boldsymbol{y} \\
&\approx \sum_{i=1}^{N_\Gamma}\sum_{i'=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}_{i'}, \boldsymbol{x}', t) \sum_{j=1}^{N_\Gamma}\sum_{j'=1}^{N_\Gamma} \alpha_j^i \alpha_{j'}^{i'} \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'})\,.
\end{aligned}$$

To write this result in terms of linear algebra operations, we introduce matrix $E \in \mathbb{R}^{N_\Gamma \times N_\Gamma}$ with

$$E = (e_{jj'})_{j,j'=1}^{N_\Gamma}\,, \quad e_{jj'} := \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'})\,.$$

Algorithm 2 Estimation of second moment

```
Require: positive definite kernel $k$
1: function COVARIANCE($\boldsymbol{x}, \boldsymbol{x}', t$)
2:     construct $X_\Gamma := \left\{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_\Gamma}\right\}$
3:     for $i = 1, 2, \ldots, N_\Gamma$ do
4:         evaluate PDE for $\boldsymbol{y_i} \to \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)$
5:     estimate $E = (e_{jj'})_{j,j'=1}^{N_\Gamma} : e_{jj'} := \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'})$
6:     compute $G = (A_{k,X_\Gamma})^{-\top} E\, (A_{k,X_\Gamma})^{-1}$ , $G = (g_{ii'})_{i,i'=1}^{N_\Gamma}$
7:     return $\left(\sum_{i=1}^{N_\Gamma}\sum_{i'=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}_{i'}, \boldsymbol{x}', t)\, g_{ii'}\right) - \left(\text{EXPECTATION}(\boldsymbol{x}, t) \cdot \text{EXPECTATION}(\boldsymbol{x}', t)\right)$
```

Matrix $G \in \mathbb{R}^{N_\Gamma \times N_\Gamma}$ encodes the last double sum

$$G = (g_{ii'})_{i,i'=1}^{N_\Gamma} , \quad g_{ii'} := \sum_{j=1}^{N_\Gamma}\sum_{j'=1}^{N_\Gamma} \alpha_j^i \alpha_{j'}^{i'} \mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'})$$

and can be obviously calculated by the matrix triple product

$$G = (A_{k,X_\Gamma})^{-\top} E\, (A_{k,X_\Gamma})^{-1} .$$

Therefore, we have a short-hand notation for the approximation of the correlation of the random PDE solution field, which is given as

$$\mathbb{E}\left[\boldsymbol{u}(\cdot, \boldsymbol{x}, t)\boldsymbol{u}(\cdot, \boldsymbol{x}', t)\right](\boldsymbol{x}, \boldsymbol{x}', t) \approx \sum_{i=1}^{N_\Gamma}\sum_{i'=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i, \boldsymbol{x}, t)\boldsymbol{u}(\boldsymbol{y}_{i'}, \boldsymbol{x}', t)\, g_{ii'} .$$

It gives rise to Algorithm 2 which outlines the important estimation steps for the full covariance function. Note, that the evaluation of the covariance function according to (4.12) might result in cancellation, in some cases. Other stochastic quantities like the second moment or the variance of a random PDE solution can be derived from the given approximations.

4.5 Auxiliary numerical methods

This section reviews the approximation of expectation values $\mathbb{E}[k(\cdot, \boldsymbol{y})]$ and $\mathbb{E}[k(\cdot, \boldsymbol{y})k(\cdot, \boldsymbol{y}')]$, sampling of the stochastic space and linear solvers for the interpolation problem.

4.5.1 Numerical quadrature

We start by investigating methods to approximate integrals of the form

$$\mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)\right](\boldsymbol{y}_j) = \int_\Gamma k(\boldsymbol{y}, \boldsymbol{y}_j)\rho(\boldsymbol{y})d\boldsymbol{y} ,$$

$$\mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'}) = \int_\Gamma k(\boldsymbol{y}, \boldsymbol{y}_j)k(\boldsymbol{y}, \boldsymbol{y}_{j'})\rho(\boldsymbol{y})d\boldsymbol{y} .$$

Since the discussion of random PDE problems is limited to cases with a finite noise assumption with uniformly distributed random variables, we assume to have

$$\Gamma = \Gamma_1 \times \ldots \times \Gamma_d \subseteq \mathbb{R}^d, \quad \Gamma_i := [a_i, b_i]\,.$$

Otherwise, quadrature on the (unbounded) full spaces could be necessary. In the following, we closely follow [GG98] to describe the general setting of numerical quadrature to approximate the above integrals. Let us have the general quadrature problem

$$I^d f := \int_\Gamma f(\boldsymbol{y}) d\boldsymbol{y}\,.$$

Furthermore the function f (restricted to Γ) shall be given in some function space

$$\mathscr{F} := \{f \mid f : \Gamma \to \mathbb{R}\}\,.$$

In numerical quadrature, we are looking for quadrature formulas

$$Q^{l,d} f := \sum_{i=1}^{N_Q^{l,d}} w_{li} f(\boldsymbol{q}_{li})\,,$$

with $w_{li} \in \mathbb{R}$ quadrature weights, $\boldsymbol{q}_{li} \in \Gamma$ *abscissas* or quadrature points, $l \in \mathbb{N}$ the quadrature (resolution) level and $N_Q^{l,d}$ the number of abscissas for a given quadrature level and dimension d. All quadrature points of a given level and dimensionality are collected in a set $X_Q^{l,d}$ with

$$X_Q^{l,d} := \left\{\boldsymbol{q}_{li} : 1 \le i \le N_Q^{l,d}\right\} \subset \Gamma\,.$$

We furthermore call abscissas on two subsequent levels *nested* if

$$X_Q^{l,d} \subset X_Q^{l+1,d}\,.$$

The quadrature error $e_Q^{l,d}$ is obviously

$$e_Q^{l,d} := |I^d f - Q^{l,d} f|\,.$$

(Quasi-)Monte Carlo quadrature

Monte Carlo (MC) and quasi-Monte Carlo (QMC) quadrature methods are a robust way to approximate (higher-dimensional) integrals. In the following [Caf98] and [DKS13] give rise to a short overview of this class of methods.

The basic methodology of (Q)MC quadrature is to use identical quadrature weights

$$w_{li} := \frac{\mathrm{Vol}(\Gamma)}{N_{Q_{MC}}^{l,d}} = \frac{\mathrm{Vol}(\Gamma)}{N_{Q_{QMC}}^{l,d}}\,,$$

where $\mathrm{Vol}(\Gamma)$ is the volume of the quadrature domain. In the setting described here, this is

$\mathrm{Vol}(\Gamma) := \prod_{i=1}(b_i - a_i)$. The number of abscissas is given by $N^l_{Q_{MC}}$ or $N^l_{Q_{QMC}}$, with

$$N^{l,d}_{Q_{MC}} = N^{l,d}_{Q_{QMC}} = 2^l \,.$$

Note that the number of quadrature points is not coupled to the dimensionality of the stochastic space Γ, here.

Monte Carlo In the pure Monte Carlo case, one uses $N^{l,d}_{Q_{MC}}$ abscissas, which are i.i.d. (independent and identically distributed) uniform random samples from Γ. By this construction, the corresponding quadrature rule $Q^{l,d}_{MC}$ is an empirical approximation to the expectation value $\mathbb{E}\left[f\right] := \int_\Gamma f(\boldsymbol{y})\rho(\boldsymbol{y})d\boldsymbol{y}$, where $\boldsymbol{y}$ is uniformly distributed over Γ. Due to the Strong Law of Large Numbers [Gri02, Section 2.13], we have a probabilistic convergence statement of the form

$$\lim_{N^{l,d}_{Q_{MC}} \to \infty} Q^{l,d}_{MC} f = I^d f, \quad \mathbb{E}\left[Q^{l,d}_{MC} f\right] = I^d f \,.$$

For $\Gamma = [0,1]^d$, we have [Caf98, Theorem 2.1], which states for large numbers of abscissas that

$$e^{l,d}_{Q_{MC}} \approx \sigma \left(N^{l,d}_{Q_{MC}}\right)^{-\frac{1}{2}} \nu = O\left((N^{l,d}_{Q_{MC}})^{-\frac{1}{2}}\right) = O\left(2^{-\frac{1}{2}l}\right) \,,$$

with $\sigma = (\mathrm{Var}\left[f\right])^{\frac{1}{2}}$ and $\nu \sim \mathcal{N}(0,1)$ thus a standard normal random variable. The big advantage of pure Monte Carlo quadrature is its independence of the dimensionality of Γ due to $e^{l,d}_{Q_{MC}} = O\left((N^{l,d}_{Q_{MC}})^{-\frac{1}{2}}\right)$. The downside of this method is its slow convergence rate of $\frac{1}{2}$ which requires to have huge numbers of abscissas to get a high probability of a low approximation error.

Quasi-Monte Carlo Many techniques have been proposed to overcome this difficulty, see e.g. [Caf98] for an overview. We here discuss the quasi-Monte Carlo approach because of its ease of use and its good convergence properties. To do this, we first have to introduce the concept of *discrepancy*. We give here the definition for $\Gamma = [0,1]^d$. Let for a set of points $\mathcal{Q} := \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_N\}$ be

$$R_N(J) = \frac{1}{N}\#\{\boldsymbol{q}_i \in J\} - \mathrm{Vol}(J)$$

the error in the Monte Carlo approximation to the volume of $J \subseteq [0,1]^d$ with $\mathcal{Q}$. Following [Caf98], we can now define *discrepancy* $D^*_N(\mathcal{Q})$ as

$$D^*_N(\mathcal{Q}) := \sup_{J \in E^*} \left|R_N(J)\right| \,, \tag{4.14}$$

where E^* is the set of all rectangular subsets in $[0,1]^d$ with one of the corners at the origin of the domain at $(0,\ldots,0)^\top$. Discrepancy gives a measure for the uniformness of a point set.

The Koksma-Hlawka theorem [Caf98, Theorem 5.1] states that

$$e^{l,d}_{Q_{MC}} \leq V[f] D^*_N(\mathcal{Q}) \,.$$

$V[f]$ is the variation in the Hardy-Krause sense, cf. [Caf98], which is a constant depending on f. The error $e^{l,d}_{Q_{MC}}$ is here understood to be the *Monte Carlo* quadrature error with respect to the point set $\mathcal{Q}$. The rather obvious consequence of this inequality is that we are looking for sequences of quadrature points with small discrepancy to minimize the error in Monte Carlo-like quadrature formulas.

Quasi-random sequences $X_{Q_{QMC}} := \{\boldsymbol{q}_i\}_{i=1}^{N}$ are designed to have a small discrepancy. They are constructed by deterministic algorithms. In [Caf98], these sequences have the requirement of

$$D_N(X_{Q_{QMC}}) \leq c(\log(N))^k N^{-1} \,,$$

with D_N as in equation (4.14) but with the suprenum over all rectangular subsets of $[0,1]^d$. The constants c and k do not depend on N but might depend on the dimension. Some existing quasi-random sequences have $k = d$. We furthermore call a sequence *uniformly distributed*, if $\lim_{N\to\infty} D_N(X_{Q_{QMC}}) = 0$. Eventually, quasi-Monte Carlo quadrature rules are Monte Carlo-type quadrature rules with quasi-random sequences as abscissas.

Let us close this paragraph by giving two examples of quasi-random sequences leading to corresponding quasi-Monte Carlo quadrature rules. The first one is the multi-dimensional *Halton* sequence. The ith element of a d-dimensional Halton sequence in $[0,1]^d$ is given as

$$\boldsymbol{q}_{li} = (\varphi_{R_1}(i), \varphi_{R_2}(i), \ldots, \varphi_{R_d}(i)),$$

$$\varphi_R(i) = \sum_{j=0}^{M} i_j R^{-j-1}, \quad \text{with} \quad i \equiv \sum_{j=0}^{M} i_j R^j, \quad M = [log_R i]$$

with the quadrature point set $X^{l,d}_{Q_{QMCH}} = \{\boldsymbol{q}_{li}\}_i$ for a Halton-based QMC quadrature rule. The idea is thus to represent the integer i dimension-wise with respect to different radices. Furthermore, the digits i_j are used as coefficient in a new basis representation. From [Caf98], we know the discrepancy of the Halton sequence to be

$$D_N(X_{Q_{QMCH}}) \leq c_d(\log N)^d N^{-1}$$

with c_d depending on d. By a modified version of the Koskma-Hlawka inequality, we have with $N := N^{l,d}_{Q_{QMCH}} = 2^l$,

$$e^{l,d}_{Q_{QMCH}} \leq c'_d V[f](\log N)^d \, (N)^{-1} = O\left(c'_d(\log N)^d \, (N)^{-1}\right) = O\left(c'_d(l)^d 2^{-l}\right) .$$

Consequently a quasi-Monte Carlo quadrature rule with the Halton series has almost first order convergence in the number of quadrature points.

The second quasi-random sequence that shall be discussed due to its popularity in quasi-Monte Carlo quadrature is the Sobol' sequence introduced in [Sob67]. According to [MC94], it is based on the idea to represent integers in a 2-adic expansion. Furthermore, following works of Niederreiter [Nie87, Nie92], it is also an example of so-called (s,t)-nets. Morokoff and Caflisch [MC94] state that the error behavior of the Sobol' sequence applied to quadrature is equivalent to the one of the Halton sequence with a much smaller constant. For further discussion on the Sobol' sequence, the interested reader should have a look at the original publication [Sob67].

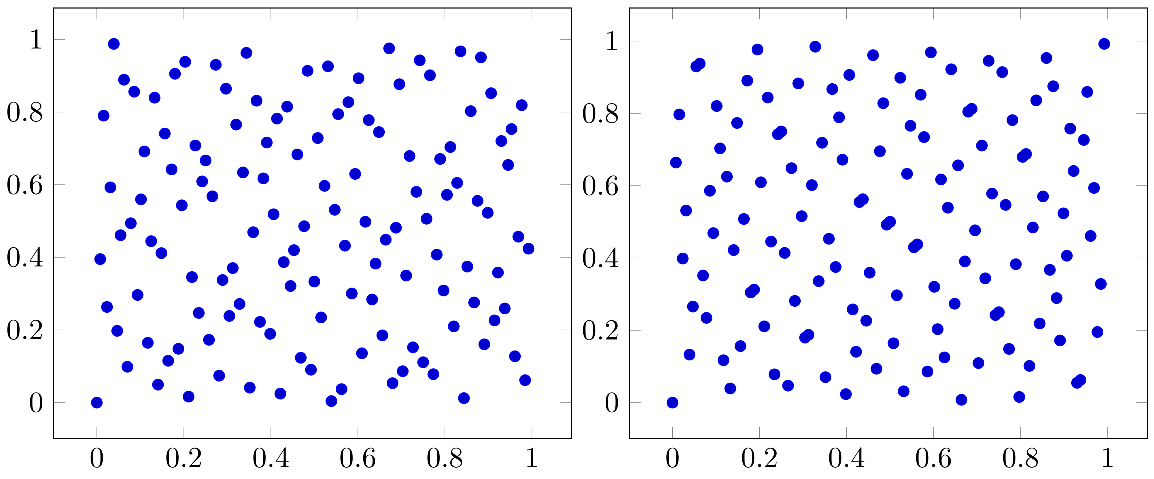


Figure 4.1: Sampling of 128 points in two dimensions by the Halton (left) and the Sobol' (right) sequence.

Figure 4.1 compares the point sampling in two dimensions for the Halton and the Sobol' sequence.

Univariate quadrature

As we could see, most (quasi-)Monte Carlo quadrature rules are straight-forward to apply to multivariate functions. Another classical approach for higher-dimensional problems is to use univariate quadrature rules as starting point to set up multi-dimensional cubature. Therefore, we now first have a look at some standard univariate quadrature methods which are then generalized to higher dimensions.

Newton-Cotes rules Newton-Cotes quadrature rules approximate integrals by polynomial interpolation on an equidistant set of points and analytic integration of the polynomials. We here follow [Sto04, Chapter 3] and [GG98] with an adapted notation. While the original Newton-Cotes formulas perform quadrature on as many abscissas as the underlying polynomial requires to be uniquely determined, it is usual to apply so-called *composite rules*. These are quadrature rules which use the Newton-Cotes formulas on subintervals of the full integration interval and combine these results to a single solution. Consequently, one ends up with composite rules with

$$N^{l,1}_{Q_{NC}} = 2^{l-1} + 1, \quad l \geq 2$$

quadratures points whose positions are given as

$$X^{l,d}_{Q_{NC}} := \{\boldsymbol{q}_{li}\}_{i=1}^{N^{l,1}_{Q_{NC}}}, \quad \text{with} \quad \boldsymbol{q}_{li} = a_1 + (i-1)\cdot h, \quad h = \frac{b_1 - a_1}{N^{l,1}_{Q_{NC}} - 1}.$$

Being based on linear interpolation, the composite trapezoidal rule Q_{NC_1} has coefficients

$$w_{li} := \begin{cases} \frac{h}{2} & i = 1, N^{l,1}_{Q_{NC}} \\ h & \text{otherwise} \end{cases}$$

and for $f \in C^2([a_1, b_1])$ there is a $\xi \in (a_1, b_1)$ such that

$$e^{l,1}_{Q_{NC_1}} = \frac{h^2(b_1 - a_1)}{12} f^{(2)}(\xi) = O(h^2) = O(2^{-2l}) \,.$$

Using polynomial interpolation of degree two, we e.g. have the composite Simpsons rule Q_{NC_2} with coefficients

$$w_{li} := \begin{cases} \frac{h}{3} & i = 1, N^{l,1}_{Q_{NC}} \\ \frac{2}{3}h & i \text{ odd} \\ \frac{4}{3}h & i \text{ even} \end{cases} ,$$

keeping in mind that index i starts from 1 instead of 0 in difference to standard literature. We can find for a given function $f \in C^4([a_1, b_1])$ a $\xi \in (a_1, b_1)$ with

$$e^{l,1}_{Q_{NC_2}} = \frac{(b_1 - a_1)}{180} h^4 f^{(2)}(\xi) = O(h^4) = O(2^{-4l}) \,.$$

Clenshaw-Curtis quadrature The Clenshaw-Curtis quadrature goes back to [CC60]. We here adapt the summary in [GG98]. Note that the usual formulation of this rule approximates integration on the interval $[-1, 1]$. Thus a suitable implementation first performs a substitution of the form

$$\int_{a_1}^{b_1} f\left(2\frac{t - a_1}{b_1 - a_1} - 1\right) \frac{2}{b_1 - a_1} dt = \int_{-1}^{1} f(y) dy$$

and then applies the usual Clenshaw-Curtis quadrature rule definitions for $[-1, 1]$ which are as follows: The number of abscissas for a given level is given as

$$N^{l,1}_{Q_{CC}} = 2^{l-1} + 1, \quad l \geq 2 \,,$$

with the actual abscissas $\boldsymbol{q}_{li}$ at non-equidistant points

$$X^{l,d}_{Q_{CC}} := \{\boldsymbol{q}_{li}\}_{i=1}^{N^{l,1}_{Q_{CC}}} \,, \quad \text{with} \quad \boldsymbol{q}_{li} = -\cos\frac{\pi(i-1)}{N^{l,1}_{Q_{CC}} - 1}$$

and weight coefficients w_{li}

$$w_{li} := \begin{cases} \frac{1}{N^{l,1}_{Q_{CC}}(N^{l,1}_{Q_{CC}} - 2)} & i = 1, N^{l,1}_{Q_{CC}} \\ \frac{2}{N^{l,1}_{Q_{CC}} - 1}\left(1 + {\sum'}_{j=1}^{(N^{l,1}_{Q_{CC}}-1)/2} \frac{1}{1-4j^2} \cdot \cos\frac{2\pi(i-1)j}{N^{l,1}_{Q_{CC}} - 1}\right) & \text{otherwise} \end{cases} ,$$

with ${\sum'}_{s=1}^{s_{max}} a_s := \frac{1}{2}a_{s_{max}} + \sum_{s=1}^{s_{max}-1} a_s$, as usual in the literature. For functions $f \in C^r([-1, 1])$ we have with $l \to \infty$ an error bound

$$e^{l,1}_{Q_{CC}} = O(2^{-lr}) \,.$$

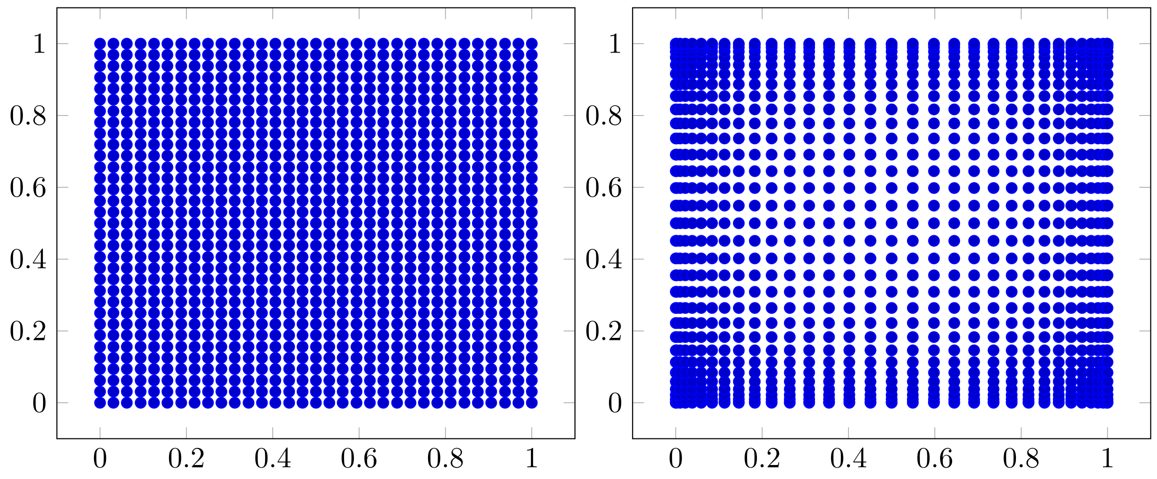


Figure 4.2: The abscissas of the full tensor product quadrature in two dimensions for the univariate composite Newton-Cotes formulas (left) and the Clenshaw-Curtis quadrature rule (right) on level 6.

Full tensor product quadrature

As long as we can describe the integration domain as product of univariate intervals, we can rather easily set up higher-dimensional quadrature rules by using tensor products of the already described univariate quadrature methods $Q^{l,1}$. Let us take here, again, the notation of [GG98].

With the definitions from above, we can introduce a d-dimensional full tensor product quadrature rule $Q_F^{l,d}$ as

$$Q_F^{l,d} f := \left(Q^{l,1} \otimes \cdots \otimes Q^{l,1}\right) f := \sum_{i_1=1}^{N_Q^{l,1}} \cdots \sum_{i_d=1}^{N_Q^{l,1}} w_{li_1} \cdots w_{li_d} f(q_{li_1}, \dots, q_{li_d}) \, .$$

Obviously, this quadrature rule needs to have

$$N_{Q_F}^{l,d} = \left(N_Q^{l,1}\right)^d = (2^{l-1}+1)^d = O(2^{ld})$$

abscissas and function evaluations, which becomes rather prohibitive in runtime for higher quadrature levels in high dimensions. The quadrature points are

$$X_{Q_F}^{l,d} := \{\boldsymbol{q}_{li}\}_{i=1}^{N_{Q_F}^{l,d}} = \prod_{j=1}^{d} X_Q^{l,1} \, .$$

Figure 4.2 shows abscissas of tensorized versions of the Newton-Cotes formulas and the Clenshaw-Curtis quadrature in two dimensions. Following [DKS13, Section 2.1], we have the error estimates for e.g. the tensorized Simpsons rule as

$$e_{Q_F}^{l,d} = O\left(2^{\frac{-4l}{d}}\right)$$

and for the Clenshaw-Curtis quadrature as

$$e_{Q_F}^{l,d} = O\left(2^{\frac{-lr}{d}}\right) ,$$

for $l \to \infty$. In both cases the smoothness requirements are equivalent to those of the applied univariate quadrature rules, thus $f \in C^4(\Gamma)$ and $f \in C^r(\Gamma)$.

Smolyak sparse grid quadrature

We had the requirement of $f \in C^p(\Gamma)$, for the full tensor product quadrature rule, with $p \in \{4, r\}$ depending on the specific method. This smoothness requirement has to be sharpened to introduce a multivariate quadrature rule by Smolyak [Smo63], which is also based on univariate quadrature rules. The advantage are quadrature methods with small errors at a lot less abscissas, even in high dimensions.

We again closely follow Griebel and Gerstner [GG98]. Note that the basic approach of Smolyak presented here, has been generalized and extended by Bungartz, Griebel and collaborators, cf. e.g. [GG98, BG04]. They use the notion of *sparse grids* to describe their constructions. Even though, it might not be correct in all cases, *Smolyak construction* and *sparse grids* are used here equivalently, to ease formulation.

Multi-indices $\boldsymbol{s} \in \mathbb{N}^d$ with $\boldsymbol{s} := (s_1, \dots, s_d)^\top$ generalize the concept of a vector to indices and help to shorten the notation in many tensor product constructions. By using these with the obvious definition of $|\boldsymbol{s}|_1 := \sum_{i=1}^d s_i$, we can introduce the function spaces involved in the Smolyak quadrature by

$$\mathcal{W}_d^p := \left\{ f : \Gamma \to \mathbb{R} \;\middle|\; \left\| \frac{\partial^{|\boldsymbol{s}|_1} f}{\partial y_1^{s_1} \dots \partial y_d^{s_d}} \right\|_\infty < \infty, s_i \leq p \right\} ,$$

thus we require f to have bounded mixed derivatives. Starting from a univariate quadrature rule, we can first introduce difference quadrature rules

$$\Delta^{l,1} f := \left(Q^{l,1} - Q^{l-1,1}\right) f := Q^{l,1} f - Q^{l-1,1} f ,$$

$$Q^{0,1} f := 0, \quad Q^{1,1} = (b-a) f\left(\frac{b-a}{2}\right) ,$$

which describe the difference in quadrature on two subsequent levels of points. In the case of nested quadrature rules $Q^{l,1}$, the difference quadrature formula $\Delta^{l,1} f$ is a weighted sum over function evaluations at abscissas $X_Q^{l,1}$, otherwise it uses the union of both subsequent quadrature point levels $X_Q^{l-1,d} \cup X_Q^{l,1}$. As one can see, the previously introduced full tensor product quadrature formula $Q_F^{l,d}$ would read in this notation as

$$Q_F^{l,d} f := \sum_{|\boldsymbol{k}|_\infty \leq l} \left(\Delta^{k_1,1} \otimes \dots \otimes \Delta^{k_d,1}\right) f ,$$

where summation over a multi-index $\boldsymbol{k} \in \mathbb{N}^d$ equals to d nested sums over the element-wise indices of the multi-index and obviously $|\boldsymbol{k}|_\infty := \max_i\{k_i\}$.

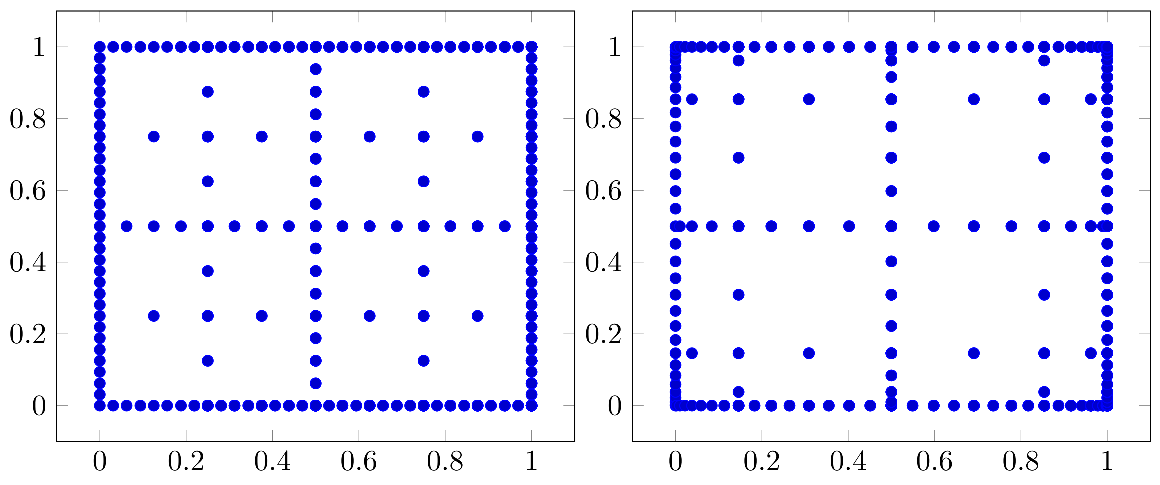


Figure 4.3: The abscissas of the Smolyak sparse grid quadrature in two dimensions for the univariate composite Newton-Cotes formulas (left) and the Clenshaw-Curtis quadrature rule (right) on level 6.

Smolyak introduced in [Smo63] the multi-variate quadrature rule

$$Q_S^{l,d} f := \sum_{|\boldsymbol{k}|_1 \le l+d-1} \left(\Delta^{k_1,1} \otimes \cdots \otimes \Delta^{k_d,1} \right) f \,,$$

for $f \in \mathcal{W}_d^p$, $k \in \mathbb{N}^d$ and the level of quadrature $l \in \mathbb{N}$. In Figure 4.3 the abscissas of the Smolyak sparse grid quadrature rule are displayed in two dimensions for the univariate composite Newton-Cotes and Clenshaw-Curtis quadrature rules. Furthermore, we have the quadrature rule $Q_C^{l,d}$ which only involves sums of full tensor product quadrature rules with different levels of quadrature in each dimension.

$$Q_C^{l,d} f := \sum_{l \le |\boldsymbol{k}|_1 \le l+d-1} (-1)^{l+d-|\boldsymbol{k}|_1-1} \binom{d-1}{|\boldsymbol{k}|_1 - l} \left(Q^{k_1,1} \otimes \cdots \otimes Q^{k_d,1} \right) f \,. \tag{4.15}$$

According to [GG98], this rule is equal to the Smolyak sparse grid quadrature. Since it combines standard tensor product full grid quadrature rules to a sparse grid rule, it is often also called sparse grid combination technique, cf. [GSZ92].

In the case of nested abscissas on subsequent quadrature levels and $N_Q^{l,1} = 2^{l-1} + 1$, we can take [BG04, Lemma 3.6] to see that the number of abscissas for the sparse grid quadrature is approximately

$$N_{Q_S}^{l,d} = O(2^l l^{d-1}) \,,$$

for $l \to \infty$, which is a lot less than the amount of quadrature points involved in the full tensor product quadrature. From [GG98], we also know that the asymptotic number of abscissas is even for the non-nested case $O(2^l l^{d-1})$ but it has a much higher constant. Based on the above construction, the set of abscissas is always

$$X_{Q_S}^{l,d} = \bigcup_{l \le |\boldsymbol{k}|_1 \le l+d-1} \left(\prod_{i=1}^{d} X_Q^{k_i,1} \right) .$$

Finally, [GG98] give an estimate for the Smolyak quadrature based on univariate Clenshaw-Curtis quadrature rules. With $N_{Q_{CC}}^{l,1}$ and $f \in \mathcal{W}_d^r$, we have for $l \to \infty$,

$$e_{Q_S}^{l,d} = O\left(2^{-lr} l^{(d-1)(r+1)}\right) .$$

Exact quadrature

Remember the objective to use quadrature methods to approximate integrals e.g. of the form $\int_\Gamma k(\boldsymbol{y}, \boldsymbol{y}_j)\rho(\boldsymbol{y})d\boldsymbol{y}$. We can sometimes evaluate these integrals analytically, at least for rather simple cases of kernels and densities. Let us exemplify this for the case of a Gaussian kernel

$$k_\epsilon(\boldsymbol{y}, \boldsymbol{y}') := e^{-\epsilon \|\boldsymbol{y}-\boldsymbol{y}'\|_2^2}$$

and the density ρ a product of uniform densities

$$\rho(\boldsymbol{y}) := \rho_1(y_1) \cdot \ldots \cdot \rho_d(y_d), \quad \rho_i(y_i) := \begin{cases} \frac{1}{b_i - a_i} & y_i \in [a_i, b_i] \\ 0 & \text{otherwise} \end{cases} .$$

Then, we can use the error function

$$\operatorname{erf}(y) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-t^2} dt$$

to express the analytic solutions of the targeted integrals with the notation $\boldsymbol{y}_j := \left(y_j^1 \ldots y_j^d\right)^\top$ as

$$\int_\Gamma k_\epsilon(\boldsymbol{y}, \boldsymbol{y}_j)\rho(\boldsymbol{y})d\boldsymbol{y} = \prod_{i=1}^{d} \frac{\pi^{1/2}(\operatorname{erf}(\epsilon^{1/2}(b_i - y_j^i)) - \operatorname{erf}(\epsilon^{1/2}(a_i - y_j^i)))}{2\epsilon^{1/2}(b_i - a_i)} ,$$

$$\int_\Gamma k(\boldsymbol{y}, \boldsymbol{y}_j)k(\boldsymbol{y}, \boldsymbol{y}_{j'})\rho(\boldsymbol{y})d\boldsymbol{y} =$$
$$\prod_{i=1}^{d} \frac{\pi^{\frac{1}{2}} e^{-\frac{1}{2}\epsilon\left(y_j^i - y_{j'}^i\right)^2}}{2\sqrt{2}\,\epsilon^{\frac{1}{2}}(b_i - a_i)} \left(\operatorname{erf}\left((\epsilon/2)^{\frac{1}{2}}\left(2b_1 - y_j^i - y_{j'}^i\right)\right) - \operatorname{erf}\left((\epsilon/2)^{\frac{1}{2}}\left(2a_1 - y_j^i - y_{j'}^i\right)\right)\right) .$$

Even though there is no closed solution of the error function, standard mathematical libraries usually implement it with almost full machine precision by some internal approximation. It is therefore rather easy to evaluate the above integrals with almost full machine precision.

The (almost) exact evaluation of the questioned integrals will sometimes allow to have faster and more accurate approximations to stochastic moments. However, the general idea is to stick to numerical quadrature in most of the cases, to be able to use arbitrary kernel functions, which might not have a solution as simple to evaluate as seen above.

sampling	common name	quad. pts.	collocation points	#points
$X^{l,1}_{\Gamma,U}$	uniform	$X^{l,1}_{Q_{NC}}$	$\boldsymbol{y}_{li} = a_1 + (i-1)\cdot\frac{b_1-a_1}{N^{l,1}_{\Gamma,U}-1}$	$2^{l-1}+1$
$X^{l,1}_{\Gamma,CC}$	Clenshaw-Curtis	$X^{l,1}_{Q_{CC}}$	$\boldsymbol{y}_{li} = -\cos\frac{\pi(i-1)}{N^{l,1}_{\Gamma,CC}-1}$	$2^{l-1}+1$
$X^{l,d}_{\Gamma,MC}$	Monte Carlo	$X^{l,d}_{Q_{MC}}$	$\boldsymbol{y}_{li} \in \Gamma$ i.i.d. unif. rand. samples	2^l
$X^{l,d}_{\Gamma,QMCH}$	Halton sequence	$X^{l,d}_{Q_{QMCH}}$	$\boldsymbol{y}_{li} \in X^{l,d}_{\Gamma,QMCH}$, see Section 4.5.1	2^l
$X^{l,d}_{\Gamma,QMCS}$	Sobol sequence	$X^{l,d}_{Q_{QMCS}}$	$\boldsymbol{y}_{li} \in X^{l,d}_{\Gamma,QMCS}$, see Section 4.5.1	2^l
$X^{l,d}_{\Gamma,F}$	full tensor prod.	$X^{l,d}_{Q_F}$	$\boldsymbol{y}_{li} \in \prod_{j=1}^d X^{l,1}_{\Gamma}$	$(2^{l-1}+1)^d$
$X^{l,d}_{\Gamma,S}$	sparse grids	$X^{l,d}_{Q_S}$	$\bigcup_{l\le\lvert\boldsymbol{k}\rvert_1\le l+d-1}\left(\prod_{i=1}^d X^{k_i,1}_{\Gamma}\right)$	$O(2^l l^{d-1})$

Table 4.1: Summary of the most important sampling methods with their equivalent abscissas from quadrature and the number of points necessary to sample a given level l.

4.5.2 Stochastic space sampling

Let us now comment on methods to sample the stochastic space Γ, thus to find a finite subset $X_\Gamma \subset \Gamma$ of collocation points with $X_\Gamma := \left\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_{N_\Gamma}\right\}$. All samplings of interest were already described as a byproduct of quadrature, sometimes with a slightly different terminology. Consequently, only a tabular overview is given. Note that using the construction from Chapter 4, the choice of collocation points is independent of the evaluation of integrals necessary for the stochastic moment estimates. Therefore, using e.g. a quasi-Monte Carlo approach to sample the stochastic space still allows to use sophisticated methods like sparse grid quadrature with a Clenshaw-Curtis rule for quadrature of the basis functions' moments. Table 4.1 summarizes possible choices of collocation point sampling. In practice, the Halton sequence will be used in all isotropic approximations.

4.5.3 Linear solvers

One important part of the algorithms to approximate the stochastic moments of a random-coefficient PDE problem, is the solution of the interpolation problem, given as linear system

$$A_{k,X_\Gamma}\boldsymbol{g} = \boldsymbol{e}$$

for the first moment and the evaluation of

$$G = \left(A_{k,X_\Gamma}\right)^{-\top} E \left(A_{k,X_\Gamma}\right)^{-1}$$

for the correlation, where $A_{k,X_\Gamma} \in \mathbb{R}^{N_\Gamma,N_\Gamma}$ is the kernel interpolation matrix. By construction, we know that compactly supported radial basis function kernels like the Wendland kernel lead to a sparse interpolation matrix while others like the Gaussian kernel or the Matérn kernel have a dense interpolation matrix A_{k,X_Γ}. The two obvious choices to solve linear systems (or equivalently to invert the matrix A_{k,X_Γ} by solving $A_{k,\Gamma}X = I$ with $X = (A_{k,X_\Gamma})^{-1}$) are direct linear solvers or iterative linear solvers. To avoid reproducing too much of the classic literature on linear solvers (e.g. [TB97, Saa03]), the idea in the following is to briefly mention the most important solver candidates with special focus on their advantages for the specific application.

In case of dense matrices, QR or LU decompositions are the usual choice, cf. [TB97]. The number of operations involved to solve a dense linear system by these methods is in the order of $O({N_\Gamma}^3)$. In that sense, these approaches are independent of the potentially bad conditioning of interpolation matrices. However, bad conditioning leads to big errors in the results, cf. [TB97]. Nevertheless, direct linear solvers are intensively applied to solve kernel interpolation problems for dense matrices.

On the other hand, iterative methods are most of the time used to solve interpolation problems with matrices stemming back from compactly supported kernels. These matrices tend to have $O(N_\Gamma)$ non-zero entries and allow for a $O(N_\Gamma)$-operation matrix-vector product. In case of strictly positive definite kernel functions, the sparse interpolation matrices A_{k,X_Γ} are also symmetric and positive definite. Thus, standard Krylov methods such as *conjugate gradient* (CG) or *GMRES* [Saa03] are typically used. Moreover, iterative solvers tend to be a good choice for dense interpolation matrices, too. Noting that e.g. an unpreconditioned CG solver would still require $O(N_\Gamma)$ iterations of complexity $O({N_\Gamma}^2)$ in the dense matrix case, preconditioning, cf. Chapter 7 or [BCM99], might lead to a drastic reduction in the number of iterations in an iterative Krylov method. Furthermore, fast multipole methods [BN92] or other fast matrix-vector products might even allow to reduce the necessary dense matrix-vector product from $O({N_\Gamma}^2)$ operations to $O(N_\Gamma \log N_\Gamma)$ numerical instructions.

Overall, in this thesis, the objective is to use both approaches. As standard solution technique, a dense LU or QR decomposition is primarily used. This allows to have a universal solution method, which is necessary in those cases, when a number of different reproducing kernels (both compactly supported and global) are tested for their numerical convergence properties. In preparation for e.g. Chapter 7, thus for the solution of kernel interpolation problems with hundreds of thousands or millions of unknowns, also a (parallel) preconditioned conjugate gradient method [Saa03] shall be used. Being an iterative method for symmetric positive definite matrices, it also allows to use preconditioners and custom built matrix-vector product applications, as it will be necessary later. Another advantage is the ease of parallelizing Krylov subspace methods even on multiple GPUs.

4.6 Approximation errors

In the last sections, the different numerical methods to evaluate stochastic moments by radial basis function kernel-based stochastic collocation have been discussed. This is now followed by a short review of the overall approximation error. Here, we start with a summary of all involved approximations. Afterwards the error splitting and some remarks on standard estimates are given.

4.6.1 Summary of approximations

Let us remember that in RBF kernel-based stochastic collocation, one is looking for solutions $\boldsymbol{u} \in L^2(\Omega; [0,T]; L^2(\mathcal{D}))$ for a general random PDE problem, which reads as: For a given stochastic space $(\Omega, \mathcal{F}, P)$ find a solution $\boldsymbol{u} : \Omega \times \bar{\mathcal{D}} \times [0,T] \to \mathbb{R}^r$ such that it holds almost surely

$$\mathcal{L}(\boldsymbol{a})\boldsymbol{u} = \boldsymbol{f}[\boldsymbol{b}] \qquad \text{in } \Omega \times \bar{\mathcal{D}} \times [0,T]\,,$$

cf. Definition 2.22. Moreover, it is usually of high interest to evaluate stochastic moments of these solutions, thus e.g. the first stochastic moment

$$\mathbb{E}\left[\boldsymbol{u}\right] = \int_{\Omega} \boldsymbol{u}\, dP(\omega)\,.$$

To approximate $\mathbb{E}\left[\boldsymbol{u}\right]$, a finite noise assumption was introduced, leading to some finite-dimensional version of the original random PDE problem with solutions

$$\boldsymbol{u}_{KL} \in L^2(\Gamma; [0,T]; L^2(\mathcal{D}))\,.$$

Here, it is assumed that the finite noise assumption is fulfilled by replacing the random input parameter fields by a truncated Karhunen-Loève series expansion. In the next step, the stochastic collocation method introduces an approximation in stochastic space by a finite number of solution evaluations. The approximate solution by stochastic collocation is $(\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{KL}) \in \mathcal{P}(\Gamma) \otimes L^2([0,T]; L^2(\mathcal{D}))$, or more specifically for the RBF kernel-based stochastic collocation

$$(\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{KL}) \in \mathcal{N}_k(\Gamma) \otimes L^2([0,T]; L^2(\mathcal{D}))\,.$$

Note that we expect here the Lavrentiev regularization, cf. Section 4.3.1 to be part of the stochastic collocation approximation. In most cases, evaluating the solution $\boldsymbol{u}_{KL}(\boldsymbol{y}_i, \boldsymbol{x}, t)$ analytically is not possible. Therefore, some finite difference or finite element approximation is necessary, before stochastic collocation can be applied. In this thesis, all approximate PDE solutions will be derived by finite differences in space and time, cf. e.g. Section 5.1. Solutions to finite difference approximations will be assumed to be given in some approximation space $V_{h_\mathcal{D},\delta t}([0,T] \times \mathcal{D}) \subset L^2([0,T]; L^2(\mathcal{D}))$. We denote with $D_{h_\mathcal{D},\delta t}$ the projection of a function $\boldsymbol{u}_{KL} \in L^2(\Gamma; [0,T]; L^2\mathcal{D})$ to the approximation space $L^2(\Omega) \otimes V_{h_\mathcal{D},\delta t}([0,T] \times \mathcal{D})$, thus we have

$$D_{h_\mathcal{D},\delta t}\boldsymbol{u}_{KL} \in L^2(\Gamma) \otimes V_{h_\mathcal{D},\delta t}([0,T] \times \mathcal{D})\,.$$

Combining the stochastic collocation approximation with the approximate PDE solution by finite differences leads to a discrete solution

$$(\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D},\delta t}\boldsymbol{u}_{KL}) \in \mathcal{N}_k(\Gamma) \otimes V_{h_\mathcal{D},\delta t}([0,T] \times \mathcal{D})\,.$$

Finally, stochastic moments are approximated by quadrature, giving e.g. for the first stochastic moment the approximation

$$\mathbb{E}\left[\boldsymbol{u}\right] \approx Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D},\delta t}\boldsymbol{u}_{KL}\,.$$

4.6.2 Error splitting

We would now like to analyze and identify the different error components involved in this construction. Note that we restrict ourselves to the time-stationary setting and expect $\boldsymbol{u}$ to be $\mathbb{R}$-valued, thus $\boldsymbol{u} : \Omega \times \mathcal{D} \to \mathbb{R}$. Thus we have to approximate stochastic collocation solutions

$$(\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL}) \in \mathcal{N}_k(\Gamma) \otimes V_{h_\mathcal{D}}(\mathcal{D}) \,,$$

with $D_{h_\mathcal{D}}$ the projection operator on the finite difference approximation space $V_{h_\mathcal{D}}$. In case of the first stochastic moment, we have the overall error $\varepsilon_\mathbb{E}$, which is usually given by

$$\varepsilon_\mathbb{E} := \left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} .$$

The first step will be to give an upper bound of this error by the sum over the error in the Karhunen-Loève approximation ε_{KL}, the stochastic collocation and discretization error and quadrature error ε_Q. We thus have

$$\left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \tag{4.16}$$

$$\leq \left\| \mathbb{E}\left[\boldsymbol{u}\right] - \mathbb{E}\left[\boldsymbol{u}_{KL}\right] \right\|_{L^2(\mathcal{D})} + \left\| \mathbb{E}\left[\boldsymbol{u}_{KL}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \tag{4.17}$$

$$\leq \left\| \mathbb{E}\left[\boldsymbol{u} - \boldsymbol{u}_{KL}\right] \right\|_{L^2(\mathcal{D})} + \left\| \mathbb{E}\left[\boldsymbol{u}_{KL}\right] - \mathbb{E}\left[\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL}\right] \right\|_{L^2(\mathcal{D})} + \underbrace{\left\| \mathbb{E}\left[\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})}}_{=:\varepsilon_Q} \tag{4.18}$$

$$= \left\| \mathbb{E}\left[\boldsymbol{u} - \boldsymbol{u}_{KL}\right] \right\|_{L^2(\mathcal{D})} + \left\| \mathbb{E}\left[\boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL}\right] \right\|_{L^2(\mathcal{D})} + \varepsilon_Q \tag{4.19}$$

$$\leq \underbrace{\left\| \boldsymbol{u} - \boldsymbol{u}_{KL} \right\|_{L^2(\Omega;L^2(\mathcal{D}))}}_{=:\varepsilon_{KL}} + \left\| \boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\Omega;L^2(\mathcal{D}))} + \varepsilon_Q \tag{4.20}$$

$$= \varepsilon_{KL} + \left\| \boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\Gamma;L^2(\mathcal{D}))} + \varepsilon_Q \,. \tag{4.21}$$

In (4.20), a trivial conclusion of the Hölder inequality is used for the first term. Furthermore, in the last equation, it was possible to apply by measure change

$$\left\| \boldsymbol{v} \right\|_{L^2(\Omega;L^2(\mathcal{D}))} = \left\| \hat{\boldsymbol{v}} \right\|_{L^2(\Gamma;L^2(\mathcal{D}))} ,$$

where $\boldsymbol{v}$ is a random field for the stochastic space $(\Omega, \mathcal{F}, P)$ and $\hat{\boldsymbol{v}}$ is the same random field with respect to the stochastic space $(\Gamma, \mathcal{B}^{N_{KL}}, \rho d\boldsymbol{y})$. To be concise, $\boldsymbol{u}_{KL}$ is used instead of $\hat{\boldsymbol{u}}_{KL}$ in (4.20).

A second step involves an estimate for the error in stochastic collocation and discretization, which is most often discussed in the literature. Here, we can derive

$$\left\| \boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\Gamma;L^2(\mathcal{D}))} \leq \underbrace{\left\| \boldsymbol{u}_{KL} - D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\Gamma;L^2(\mathcal{D}))}}_{=:\varepsilon_{h_\mathcal{D}}} + \underbrace{\left\| D_{h_\mathcal{D}} \boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\Gamma;L^2(\mathcal{D}))}}_{=:\varepsilon_{N_\Gamma}} ,$$

and get an upper bound by the sum over the finite difference discretization error $\varepsilon_{h_\mathcal{D}}$ and the stochastic collocation error ε_{N_Γ}.

Overall, we thus end up having the following estimate for the approximation error of the first stochastic moment:

$$\left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \leq \varepsilon_{KL} + \varepsilon_{h_\mathcal{D}} + \varepsilon_{N_\Gamma} + \varepsilon_Q \tag{4.22}$$

$$\begin{aligned} &= \|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))} + \|\boldsymbol{u}_{KL} - D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \\ &\quad + \|D_{h_\mathcal{D}}\boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \\ &\quad + \left\| \mathbb{E}\left[\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} . \end{aligned} \tag{4.23}$$

The first error term ε_{KL} strongly depends on the chosen random PDE problem and the covariance structure of the input random field $\boldsymbol{a}(\omega, \boldsymbol{x})$. For some random PDE problems, it has been shown that the eigenmode decay of the input covariance leads to a similar decay in the covariance spectrum of the solution field $\boldsymbol{u}(\omega, \boldsymbol{x})$. In case of an input covariance with exponential eigenvalue decay, this gives an exponential convergence in the solution $\boldsymbol{u}_{KL}$ after Karhunen-Loève expansion. Thus, in some cases, ε_{KL} can be expected to be exponentially convergent for a growing number of collocation points N_Γ. At the same time the PDE solution error $\varepsilon_{h_\mathcal{D}}$ is often assumed to have an algebraic convergence in the mesh width $h_\mathcal{D}$. The stochastic collocation error ε_{N_Γ} depends on the chosen kernel function and the smoothness of the solution $(D_{h_\mathcal{D}}\boldsymbol{u})(\boldsymbol{y}, \boldsymbol{x})$ with respect to its finite-dimensional stochastic dependence, cf. Section 4.3.2. Exponential convergence is possible if the function $(D_{h_\mathcal{D}}\boldsymbol{u})(\boldsymbol{y}, \cdot)$ is element of the native space $\mathcal{N}_{k_\epsilon}(\Gamma)$ of the Gaussian kernel. Finally, depending on the problem, the quadrature error might decay exponentially by using e.g. tensorized or sparse grid versions of the Clenshaw-Curtis quadrature rule.

Some of the above assumptions require a decent amount of analysis of the involved random PDE problems and discretizations. In some cases, as the two-phase incompressible Navier-Stokes equations (in strong formulation), this might even require to solve rather important open problems, especially if the stochastic influence is no longer just in a coefficient but in initial or boundary conditions or the computational domain. This is why the above error components will be analyzed by *empirical* means in Section 6.3.

4.7 Computational complexities

Next, the computational cost of the approximation of first stochastic moments of a random PDE problem by the RBF stochastic collocation method shall be discussed. Remember that this approximation consists of the solution of deterministic PDE problems by finite differences, numerical integration to derive the first moment of the RBF kernel basis functions and solution of a kernel interpolation problem. Other parts of the mean approximation, such as the sampling in stochastic space, are neglected due to minor complexity, cf. also Section 4.8.5.

Solution of deterministic PDEs Let us start with the computational costs of the finite difference approximation D_h of elliptic PDE problems. For a uniform two-dimensional discretization grid with $N_{\mathcal{D}}$ grid points, we have the relationship

$$N_{\mathcal{D}} \sim \left(\frac{1}{h_{\mathcal{D}}}\right)^2$$

for the usual mesh width $h_{\mathcal{D}}$. The standard five-point finite difference discretization results in a sparse linear system with $O(N_{\mathcal{D}})$ non-zero matrix entries. Solving this problem with an iterative Krylov solver requires at most $O(N_{\mathcal{D}})$ iterations, with the dominating operation the matrix-vector product. Preconditioning might improve on this, cf. Chapter 8. Nevertheless, overall, the upper bound to the computational costs to solve the elliptic PDE, as discussed before, is

$$C_D^{elliptic}(N_{\mathcal{D}}) = O({N_{\mathcal{D}}}^2) \quad \text{or} \quad C_D^{elliptic}(h_{\mathcal{D}}) = O\left(\left(\frac{1}{h_{\mathcal{D}}}\right)^4\right) .$$

An analogous discussion is possible for the discretization of the three-dimensional two-phase Navier-Stokes equations. Their numerical treatment will be outlined in detail in Chapter 5. However, short statement on the computational complexity can be done here, already. We will see that a finite difference - based space discretization of the two-phase Navier-Stokes equations requires

$$N_{\mathcal{D}} \sim \left(\frac{1}{h_{\mathcal{D}}}\right)^3$$

Furthermore, a discretization in time becomes necessary. For a simulated time T and a time step size of δt, the number of time steps is

$$N_{\mathcal{T}} \sim \frac{T}{\delta t} .$$

Noting that the solution of a three-dimensional elliptic problem per time-step, cf. Section 5.1.2, is the dominant part, the number of operations to solve the deterministic two-phase Navier-Stokes equations is

$$C_D^{Navier}(N_{\mathcal{D}}, N_{\mathcal{T}}) = O({N_{\mathcal{D}}}^2 N_{\mathcal{T}}) \quad \text{or} \quad C_D^{Navier}(h_{\mathcal{D}}, \delta t) = O\left(\left(\frac{1}{h_{\mathcal{D}}}\right)^6 \frac{1}{\delta t}\right) .$$

Quadrature The involved quadrature problem is often approximated by a tensorized Clenshaw-Curtis rule. In the univariate case, there is an efficient method ([Gen72]), to compute Clenshaw-Curtis rules by $O(N_Q^{l,1} \log N_Q^{l,1})$ operations, with $N_Q^{l,1}$ the number of univariate abscissas, cf. Section 4.5.1. For a stochastic space with N_{FN} or N_{KL} dimensions, sticking to full tensor-product rules based on univariate Clenshaw-Curtis rules gives a runtime complexity of

$$C_Q\left(N_Q^{l,1}, N_{KL}\right) = O\left(\left(N_Q^{l,1} \log N_Q^{l,1}\right)^{N_{KL}}\right) \quad \text{or}$$

$$C_Q\left(l_Q, N_{KL}\right) = O\left(\left(l_Q\, 2^{l_Q}\right)^{N_{KL}}\right) .$$

RBF-based kernel interpolation To solve the RBF kernel stochastic collocation problem, a kernel interpolation with a dense or sparse linear system has to be solved, cf. Section 4.5.3.

Direct or unpreconditioned iterative solvers for a dense linear system require a cubic number operations in the number of unknowns. The number of unknowns is equal to the number of collocation points N_Γ. As e.g. motivated in Section 4.3.2, we might also be interested to express the number of collocation points in terms of the fill distance $h_{X_\Gamma,\Gamma}$. Here, the relationship is very roughly

$$N_\Gamma \sim \left(\frac{1}{h_{X_\Gamma,\Gamma}}\right)^{N_{KL}} .$$

Overall, we have the computational costs for the kernel interpolation given as

$$C_{\mathcal{S}}(N_\Gamma) = O\left({N_\Gamma}^3\right) \quad \text{or} \quad C_{\mathcal{S}}(h_{X_\Gamma,\Gamma}) = O\left(\left(\frac{1}{h_{X_\Gamma,\Gamma}}\right)^{3\,N_{KL}}\right) .$$

Approximation of the first stochastic moment The full approximation of a first stochastic moment of a random PDE problem requires to solve – per collocation point – a quadrature problem at cost $C_{\mathcal{D}}$ and a deterministic PDE problem at complexity C_Q. Moreover, the collocation problem or kernel interpolation problem with cost $C_{\mathcal{S}}$ has to be solved. Therefore, the overall computational complexity for this method is in the elliptic random PDE problem case

$$\begin{aligned} C^{elliptic}(N_\Gamma, N_{\mathcal{D}}, N_Q^{l,1}, N_{KL}) &= N_\Gamma\, C_D^{elliptic}(N_{\mathcal{D}}) + N_\Gamma\, C_Q\left(N_Q^{l,1}, N_{KL}\right) + C_{\mathcal{S}}(N_\Gamma) \\ &= O\left(N_\Gamma {N_{\mathcal{D}}}^2 + N_\Gamma\left(N_Q^{l,1}\log N_Q^{l,1}\right)^{N_{KL}} + {N_\Gamma}^3\right) \end{aligned} \tag{4.24}$$

or

$$\begin{aligned} &C^{elliptic}(h_{X_\Gamma,\Gamma}, h_{\mathcal{D}}, l_Q, N_{KL}) \\ &\quad= (h_{X_\Gamma,\Gamma})^{-N_{KL}}\, C_D^{elliptic}(h_{\mathcal{D}}) + (h_{X_\Gamma,\Gamma})^{-N_{KL}}\, C_Q(l_Q, N_{KL}) + C_{\mathcal{S}}(h_{X_\Gamma,\Gamma}) \\ &\quad= O\left((h_{X_\Gamma,\Gamma})^{-N_{KL}}\,(h_{\mathcal{D}})^{-4} + (h_{X_\Gamma,\Gamma})^{-N_{KL}}\left(l_Q\, 2^{l_Q}\right)^{N_{KL}} + (h_{X_\Gamma,\Gamma})^{-3\,N_{KL}}\right) . \end{aligned}$$

Approximation of the first stochastic moment of the random two-phase Navier-Stokes equations has a computational cost of

$$\begin{aligned} &C^{Navier}(N_\Gamma, N_{\mathcal{D}}, N_{\mathcal{T}}, N_Q^{l,1}, N_{KL}) \\ &\qquad= N_\Gamma\, C_D^{Navier}(N_{\mathcal{D}}, N_{\mathcal{T}}) + N_\Gamma\, C_Q\left(N_Q^{l,1}, N_{KL}\right) + C_{\mathcal{S}}(N_\Gamma) \\ &\qquad= O\left(N_\Gamma {N_{\mathcal{D}}}^2 N_{\mathcal{T}} + N_\Gamma\left(N_Q^{l,1}\log N_Q^{l,1}\right)^{N_{KL}} + {N_\Gamma}^3\right) \end{aligned} \tag{4.25}$$

or

$$\begin{aligned} &C^{Navier}(h_{X_\Gamma,\Gamma}, h_{\mathcal{D}}, \delta t, l_Q, N_{KL}) \\ &\quad= (h_{X_\Gamma,\Gamma})^{-N_{KL}}\, C_D^{Navier}(h_{\mathcal{D}}, \delta t) + (h_{X_\Gamma,\Gamma})^{-N_{KL}}\, C_Q(l_Q, N_{KL}) + C_{\mathcal{S}}(h_{X_\Gamma,\Gamma}) \\ &\quad= O\left((h_{X_\Gamma,\Gamma})^{-N_{KL}}\,(h_{\mathcal{D}})^{-6}(\delta t)^{-1} + (h_{X_\Gamma,\Gamma})^{-N_{KL}}\left(l_Q\, 2^{l_Q}\right)^{N_{KL}} + (h_{X_\Gamma,\Gamma})^{-3\,N_{KL}}\right) . \end{aligned}$$

Note that the above formulation of complexity estimates requires knowledge on the specific choice of all parameters $N_\Gamma, N_\mathcal{D}, N_\mathcal{T}, N_Q^{l,1}, N_{KL}$. In practice, we would rather want to express complexities in terms of a given upper error bound to the approximation in the first stochastic moment. This requires a careful error (coupling) analysis. We will postpone this analysis to Section 6.3, where it will be given by empirical means.

Equations (4.24) and (4.25) clarify the influence of the three dominant computation parts. Quadrature can usually be expected to have a minor influence on the overall method, due to small constants, potential pre-computation of the integrals or even analytic formulas to evaluate the integrals. However, quadratic and cubic complexities in the number of space discretization points and collocation points have a huge impact on performance. These performance limitations will be solved in Part II of this thesis.

4.8 Multi-GPU parallel implementation

Kernel-based moment estimation including stochastic collocation is implemented by several numerical libraries. These are a set of templated C++ libraries for GPU parallel point sampling, quadrature and (multi-)GPU parallel kernel-based interpolation and the C-based multi-GPU parallel library *parla* for dense linear algebra routines. To avoid a strenuous review of the libraries' structure, the intention is to place a focus on important algorithmic and technical details which allow for optimal performance on GPUs and best possible scalability in the multi-GPU case. The discussion is structured by types of numerical methods, ignoring the potential different ordering of code parts in the actually implemented codes. Before we come to the implementation details, a short paragraph is dedicated to general remarks on the the way, the code is implemented for GPUs. Two other paragraphs will cover data structures on single GPUs and in the multi-GPU case.

4.8.1 General remarks

The implemented codes use CUDA [NVI14b] as C/C++ programming extension for (Nvidia) GPUs. In the following, the reader is expected to know the basics of memory hierarchies and the parallel programming model used in CUDA and on GPUs in general. An in-depth introduction to this topic can be e.g. found in [SK10] with the official manuals in [NVI14b]. If not otherwise stated, CUDA version 5.0 is used. Another important library which is applied without further discussion is *Thrust* 1.5.3 [HB10], a C++ template library which mimics the vector class of the C++ Standard Template Library together with a large set of algorithms which are implemented for GPUs. This software is part of CUDA Software Development Kit [NVI10]. It is expected that this library delivers good performance for the provided algorithms, thus there is no special need to optimize these code parts. If possible, existing GPU libraries are used for more complex operations. These are mentioned as necessary. GPU kernels are only hand-implemented, if there is no library or it is possible to achieve higher performance by self-implemented codes. Even though the implemented software can often handle single and double precision operations and data structures, all numerical methods are carried out in double precision.

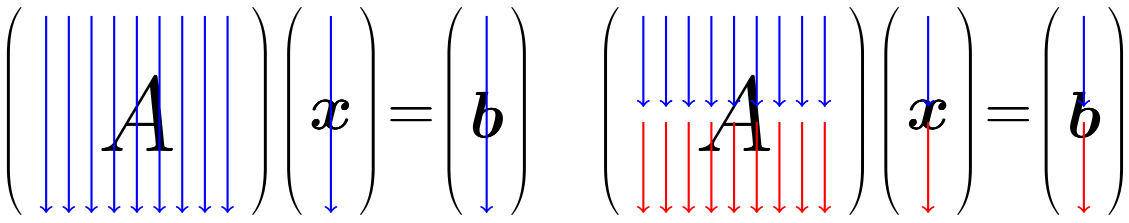


Figure 4.4: Column-major ordering of matrices and vectors in the single-GPU case (left) and the domain decomposition case (right) for two GPUs. Vector $\boldsymbol{x}$ requires a global communication before each multi-GPU parallel matrix-vector product.

4.8.2 Data structures on a single GPU

Matrices

Matrices are stored in column-major ordering in GPU and CPU memory with linearized index to be compatible with the applied dense linear algebra software libraries *cuBLAS*, which is the GPU BLAS library delivered with the CUDA Toolkit [NVI10], and *CULA* [EM 13], a commercial single-GPU *LAPACK* [ABB+99] implementation. Therefore, we have for an entry a_{ij} of the matrix $A \in \mathbb{R}^{m\times n}$ the mapping

$$a_{ij} \quad \longrightarrow \quad \texttt{a[i+j*m]}\,,$$

cf. Figure 4.4. CUDA kernels that have to access all elements of a matrix, map one entry to one thread and 512 threads are usually put together into a thread block. With GPUs of compute capability less than 3.0 (cf. [NVI14b, Appendix G]), the indexing of thread blocks has also to be split up in two index dimensions, since the first thread block index is limited to up to 65535 blocks, which does not allow to address the full memory by threads. Consequently, we have for the evaluation of an element a_{ij} the thread index mapping

$$bs_1 = 512, \quad bs_2 = bs_3 = 1\,,$$

$$gs_1 = 65535, \quad gs_2 = (((m\cdot n + bs_1 - 1)/bs_1) + 65534)/65535, \quad gs_3 = 1$$
$$idx = b_2 \cdot 65535 \cdot bs_1 + b_1 \cdot bs_1 + t_1\,,$$
$$a_{ij} \to \texttt{a[idx]}\,,$$

where gs is the grid size / number of blocks, b is the block index, bs is the block size and t is the thread index, with all these quantities being defined in three dimensions. A closer look at this indexing scheme shows that it results in a rather wasteful use of threads, which however does not necessarily limit performance. Starting with Kepler GPUs, this limitation is removed, leading to an indexing scheme of

$$bs_1 = 512, \quad bs_2 = bs_3 = 1\,,$$
$$gs_1 = (m\cdot n + bs_1 - 1)/bs_1, \quad gs_2 = gs_3 = 1$$
$$idx = b_1 \cdot bs_1 + t_1\,.$$

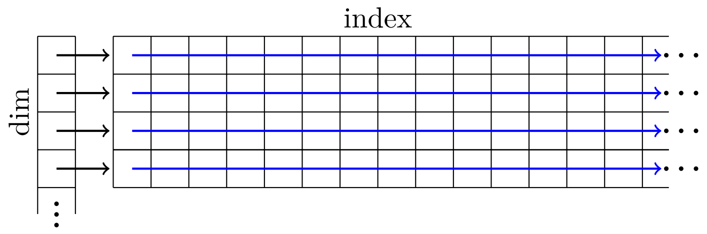


Figure 4.5: Multi-dimensional data is stored according to the *structs of arrays* principle with a pointer array on CPU/GPU adressing contiguous GPU arrays per dimension.

Multi-dimensional data on GPUs

Collocation and quadrature points or more specifically their coordinates are stored in what is usually called *structs of arrays*, cf. the corresponding literature, e.g. [Far11, Chapter 6]. Consequently, the coordinate entries are placed dimension-wise consecutively in memory. This is the preferred way to store multi-dimensional data to be efficiently evaluated on GPUs. Since the implemented code shall cover arbitrary finite dimensions in the stochastic space, a fixed struct cannot be used to group together the elements. Instead a dynamically allocated array of pointers is used. While it is obvious to store the arrays containing the tuple entries in GPU memory, the array of pointers to these arrays might be placed in CPU or GPU memory. In fact, both approaches are sometimes necessary depending on whether the list of points are accessed from GPU kernels or by CPU-based commands like the memory copy operation of CUDA `cudaMemcpy`.

Handling arbitrary-dimensional data causes some problems in the implementation of GPU kernels which access all dimensions of an entry at once and have to create temporary memory with that dimensionality. On CPUs, one would easily create dynamically allocated temporary memory. However, even though this language feature has been introduced with CUDA 3.2 within kernel calls, it is not supported for GPUs with compute capability 1.3, cf. [SK10]. Furthermore, even with the latest CUDA version and hardware, numerical tests with dynamic memory allocations per GPU thread have shown to deliver very bad performance on the newer 2.0 compute capability GPUs and non-optimal performance even on the latest compute capability 3.5 (Kepler) GPUs. Therefore, it is prohibitive to use this feature in this case.

Pre-allocating temporary memory for each thread requires to have $O(d \cdot t)$ storage, where d is the dimensionality and t is the number of threads on the GPU (which is usually as high as the number of entries of the multi-dimensional array). In many applications, it is impossible to allocate that additional amount of memory, especially when pushing GPUs to their limits. A rather straight-forward idea would be to allocate $O(d \cdot p)$ storage, and couple the memory to each multiprocessor. Here, p is the actual number of streaming processors or effectively executed parallel threads on a GPU. Then, each processor would take that part of the storage statically over time. However, this approach is not supported by the parallel programming model exposed by CUDA.

The only option is thus to use a pre-defined fixed-size (constant memory or register-based) array. Here, one can use the template C++ approach to introduce the dimension as template parameter. This does not work for pure C codes and might also heavily impact the time to compile. A direct translation to C is to use a preprocessor directive to define a dimension. Both approaches do not work, if the dimensionality is discovered at run-time, which shall be done here. Therefore, in all implementations, there is a fixed upper limit of dimensions, which is set, for now, to 20. Temporary memory with that fixed size is allocated and only the first d entries are used. Even though this procedure wastes a bit of memory, it comes rather close to an optimal $O(d \cdot p)$ storage approach.

4.8.3 Multi-GPU data structures

Matrices

Whenever dense matrices shall be distributed over several GPUs, the idea is to apply a decomposition of the full matrix into blocks of rows as indicated in Figure 4.4, which is a classical approach. Moreover, this technique allows to have a very simple per-GPU matrix-vector multiplication. Note that e.g. [Sør12] proposes to additionally decompose the matrix in blocks of columns. This will be implemented as soon, as it turns out that the pure row decomposition is a bottleneck for further scaling.

Vectors

Vectors are treated equivalently to matrices, in the multi-GPU case. Therefore, row vectors are not decomposed, while column vectors are distributed across the GPUs. This requires a global communication before each matrix-vector product, cf. Figure 4.4.

4.8.4 Kernel matrix setup

The setup of interpolation matrices $A_{k,X}$ defined by (4.6) is done in a CUDA kernel with the collocation points as input. Accessing the matrix entries is done as described in Section 4.8.2. The generalized norm evaluation uses d-dimensional temporary memory which is implemented as also discussed in Section 4.8.2. For small dimension numbers, the necessary reduction operation over all dimensions (to compute the norm) is carried out in a sequential way, per thread. This might become less performant in larger dimensions where a more complex shared memory parallel reduction per matrix entry would be of interest. Classes for kernel functions, including Gaussian, Wendland and Matérn kernels, are available and can be easily extended. All arithmetic operations and special function evaluations are carried out by the *CUDA Math API*.

4.8.5 Stochastic space sampling

Monte Carlo collocation points are created using the GPU random number generator library *cuRAND* of the CUDA Programming Toolkit. Here, the default pseudo-random number generator XORWOW [Mar03] is used to generate uniformly distributed values within the interval $[0, 1]$. By an appropriate transformation, they are mapped to the necessary stochastic domain Γ. There are no published estimates on the theoretical parallel computational complexity of cuRAND, at least to the authors knowledge.

Multi-dimensional Sobol' quasi-Monte Carlo sequences are also generated by the cuRAND library in a similar way. The Halton sequence is not available in cuRAND 5.0 and also not in version 5.5. Therefore, it is generated using the GNU Scientific Library (*GSL*) [Gou09] and copied to GPU.

Both univariate sequences, thus the uniform sequence and the Clenshaw-Curtis quadrature point sequence, are easily created by their definitions from Section 4.5.2 and appropriate operations from Thrust. The multivariate full tensor product of these points is generated by a hand-written kernel which uses the univariate sequences as input.

Sparse grid points are for now constructed according to the formula in Table 4.1. First, the multi-indices of the tensor product samplings are generated on GPU per subspace. Finally, all points are unified by sorting and compacting the sequences. This is for sure not the fastest possible approach, however it easily fits into the implemented modularized libraries. For fast methods, one could e.g. have a look at [MWB+11], however sampling is usually no performance bottleneck, anyway.

4.8.6 Quadrature methods

If not otherwise stated, the abscissas are generated as already described. (Quasi-)Monte Carlo methods require a simple reduction operation with constant weights. For composite Newton-Cotes formulas, custom weights have to be generated. In the Clenshaw-Curtis case this is done with a GPU-based $O(N_{Q_{CC}}^{l,1} \log(N_{Q_{CC}}^{l,1}))$ implementation using the Fast Fourier Transform (FFT) from the *cuFFT* library of the CUDA Toolkit as e.g. proposed in [Gen72]. The current implementation is based on [von05]. For the full tensor product quadrature, the GPU-computed univariate quadrature weights and points are taken and tensorized in appropriate kernels.

Smolyak sparse grid quadrature is implemented with the combination technique [GSZ92, GG98], thus by (4.15). Like in the sparse grid point sampling case, first the multi-indices of the subspaces are computed sequentially. These are used to set up and solve the anisotropic full tensor product quadrature problems on the GPU. All results are combined as in (4.15). However, his approach does not turn out to deliver optimal performance. This is because each of the sequentially started small quadrature problems uses only a fraction of the GPU peak performance. An optimal implementation would schedule in parallel as many quadrature problems as possible. This is future work.

4.8.7 Linear solvers

As motivated in Section 4.5.3, direct solvers are used in the single-GPU case and iterative solvers in the multi-GPU case. The direct solver is the LU decomposition with partial pivoting and row interchanges from the CULA library [EM 13].

In the parallel case, the intention is to have a library with dense iterative solvers, as pointed out in Section 4.5.3. It is *distributed* memory multi-GPU parallel such that it can scale on a cluster of GPUs connected by some fast network. To the authors knowledge, there are almost no software libraries available, which support this. While *PETSc* [BAB+13] is one of the most prominent software frameworks in this context, its only support for GPUs is based on an interface to the sparse linear algebra library *CUSP* [BG12], to the authors knowledge. Thus, there is no dense linear algebra for GPUs available in PETSc. Another prominent multi-GPU linear algebra library is *MAGMA* [TDB10] which has hybrid multi-GPU support for some of its dense solvers. However, it is unclear whether it has an easily accessible multi-GPU parallel dense matrix-vector product for an iterative solver and it has also the issue of being only sparsely documented, at least for the multi-GPU part. The library *LAMA* [KFBS13] might be the only software tool, which delivers the required BLAS dense linear algebra standard routines for distributed memory multi-GPU clusters. Though, since some of the numerical methods developed for this thesis require a high amount of problem-specific modifications, the high-level structure of this library makes it a non-optimal candidate, here.

Therefore, the new, lightweight, distributed memory multi-GPU parallel library *parla* with parallel standard BLAS linear algebra routines (like matrix-vector product, scalar product, vector addition, ...) and a BLAS-like interface is developed and extended by the iterative Krylov subspace methods of interest. The major building blocks of parla are cuBLAS and the Message Passing Interface (MPI) [Mes94] for parallel communication between the GPUs. Because of the described decomposition of matrices, cf. Section 4.8.3, a parallel matrix-vector product requires only to communicate the vector, on which to apply the matrix. Product evaluation and result stay local to the GPU. Furthermore, the *overlapping communication and computation* technique [Mic09] can be used to start with computation on the part of the vector, which is local to the GPU, while the other vector parts are exchanged over the network in background. In the implementation, the so-called *CUDA-aware OpenMPI* 1.7.3 library for parallel communication is used. It allows to pass GPU memory pointers to MPI commands. The MPI implementation then takes care of using the fastest possible strategy to transfer data from GPU memory over the network to a remote GPU. Taking overlapping and CUDA-awareness together will allow to have perfect scale-up and good speed-up in all dense iterative solvers. For more details on the underlying techniques, see also Section 5.3.2.

Besides of the matrix-vector product, the other routines are implemented rather straightforward by locally using cuBLAS methods and, if necessary, communicating the results by MPI. The same holds for the implementation of the conjugate gradient method, which is done according to standard textbooks [Saa03].

4.8.8 Stochastic collocation framework and coupling to flow solver

The implemented libraries for sampling, quadrature and kernel-based interpolation can be easily combined to solve RBF kernel based stochastic collocation problems, delivering a general framework for these kinds of problems. In the case of the model problems from Sections 3.1 and 3.2, convergence benchmark codes are implemented that contain both, the solvers for the underlying PDE problems (by the sparse linear algebra framework CUSP) and the numerical methods for stochastic collocation.

In case of (two-phase) flow simulations, cf. Chapter 5, stochastic collocation is built in a separate software tool. Figure 4.6 illustrates this. Appropriate flow solver parameter files are created by some *Python* code. For each collocation point, one flow simulation is executed, which is a trivially parallel operation. Their results are written into VTK [SAH00] files. Afterwards, the separate stochastic collocation software reads flow fields from the VTK files. Custom methods for reading and writing files of this format to and from GPU memory have been implemented. Finally the stochastic collocation problem and moment estimate is performed on GPU as for the model problems. Moments of flow fields are written back to VTK files

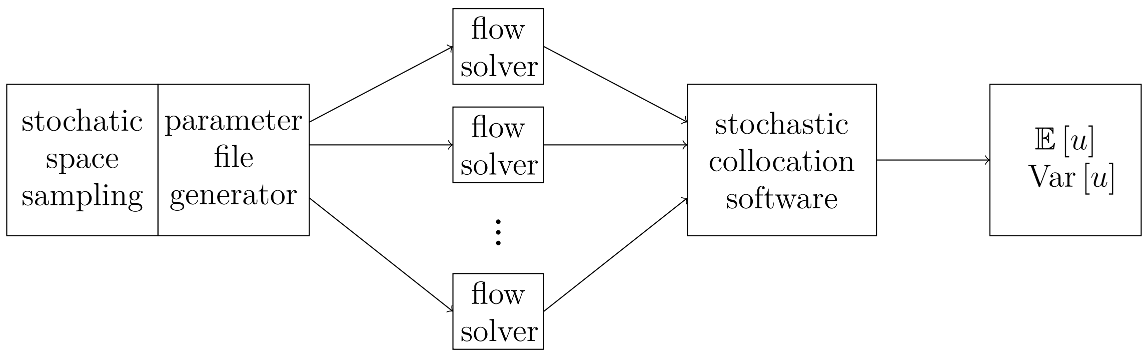


Figure 4.6: Stochastic collocation and moment approximations for two-phase flow problems are computed with a loose coupling by VTK files between the flow solver and the stochastic collocation software.

and visualized by *Paraview* [Hen07], if necessary. Note that it is also possible to extract rather complicated quantities of interest of the generated flow fields in Paraview and estimate stochastic moments of these quantities by the GPU-based stochastic tool.

All details about solving the (deterministic) two-phase incompressible Navier-Stokes equations including implementation and performance results on multi-GPU clusters, will be discussed in the next chapter.

5 Multi-GPU parallel solver for the two-phase Navier-Stokes equations

While the previous chapters highlighted all information on a kernel-based stochastic collocation method with high convergence order and low number of collocation points, this chapter is involved with the efficient treatment of the deterministic PDE problem that has to be solved within each stochastic collocation point of the random two-phase Navier-Stokes application problem. Thus we seek for a fast and efficient solver for the equation system (3.3)–(3.9) or (3.17)–(3.23) introduced in Section 3.3. The objective is to have a full multi-GPU parallel solver for this problem, in order to have the best possible time-to-solution using the latest available massively parallel compute hardware.

Base for all multi-GPU developments is the MPI-parallel two-phase fluid solver NaSt3DGPF [STCE06, CGS09], which was shortly introduced in Chapter 1. This chapter thus primarily addresses the multi-GPU parallelization of an existing research code up to a point in which all main components are fully available on GPUs. It will be shown that it is possible to achieve speedups on GPUs in contrast to CPUs which are in the range of a factor of tree, when comparing equally priced hardware. Furthermore, it is possible to measure clear differences in power consumption between the different kinds of processor architecture, leading to an additional power consumption reduction of a factor of two by comparing the power consumed by a single-GPU system with the power consumed by a multi-core CPU system. The additional MPI parallelization of the overall flow solver allows to further scale on a cluster of GPUs achieving a weak scaling efficiency larger than 90 percent. Overall, these achievements will outline the clear necessity of optimally parallelized and hardware optimized PDE solvers in frameworks of large scale uncertainty quantification analysis.

This chapter starts by giving a brief overview of the steps to remodel the two-phase Navier-Stokes equations (3.3)–(3.9) to a jump-discontinuity free formulation and to discretize this derived equation system. Thereafter, techniques to perform an efficient single-GPU parallelization of a finite difference flow solver are outlined. These are followed by a discussion of scalable multi-GPU parallelizations based on domain decomposition and MPI. Finally, an in-depth performance analysis including single-GPU results, multi-GPU scalability and energy consumption is presented.

The results outlined in this chapter have been previously published by the author in two journal articles [GZ10, ZG13]. They will be summarized and put into a more complete background within this thesis. Also, a few updates on recent technology advancements are given. Note that not all details of [GZ10, ZG13] are covered here and performance results mainly base on [ZG13].

5.1 Numerical treatment of the incompressible two-phase Navier-Stokes equations

Let us start by remembering the two-phase incompressible Navier-Stokes equations as presented in Section 3.3. To be concise, we discuss here the original deterministic formulation from equations (3.3)–(3.9) instead of the random version after finite noise assumption in equations (3.17)–(3.23), which has no influence on the applied numerical techniques or the implementation. The two-phase Navier-Stokes equations are defined on a connected domain $\mathcal{D} \subset \mathbb{R}^3$ with boundary $\mathbb{\Gamma}$ over time $t \in [0, T]$. To describe two interacting fluid phases, $\mathcal{D}$ is covered by the non-overlapping, time-dependent sub-domains $\mathcal{D}_1(t)$ and $\mathcal{D}_2(t)$ with separation interface $\mathbb{\Gamma}_f(t)$ (thus $\mathcal{D} = \mathcal{D}_1(t) \cup \mathcal{D}_2(t) \cup \mathbb{\Gamma}_f(t)$). Both domains $\mathcal{D}_1$ and $\mathcal{D}_2$ have their own material parameters for density $\rho_1, \rho_2 \in \mathbb{R}$ and viscosity $\mu_1, \mu_2 \in \mathbb{R}$. For $i = 1, 2$, the equation system is given by

$$\begin{aligned}
\rho_i \partial_t \boldsymbol{u}_i + \rho_i (\boldsymbol{u}_i \cdot \nabla) \boldsymbol{u}_i &= \nabla \cdot \mu_i (\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T) - \nabla p_i + \rho_i \boldsymbol{g} && \text{in } \mathcal{D}_i \times [0, T]\,, && (5.1)\\
\nabla \cdot \boldsymbol{u}_i &= 0 && \text{in } \mathcal{D}_i \times [0, T]\,, && (5.2)\\
\boldsymbol{u}_i &= \boldsymbol{u}_{0_i} && \text{in } \mathcal{D}_i \times \{0\}\,, && (5.3)\\
\mathcal{B}[\boldsymbol{a}_{\mathbb{\Gamma}}] \boldsymbol{u}_i &= \boldsymbol{b}_{\mathbb{\Gamma}} && \text{on } \mathbb{\Gamma} \times [0, T]\,, && (5.4)\\
\frac{\partial p_i}{\partial \boldsymbol{n}_{\mathbb{\Gamma}}} &= 0 && \text{on } \mathbb{\Gamma} \times [0, T]\,, && (5.5)\\
\boldsymbol{u}_1 &= \boldsymbol{u}_2 && \text{on } \mathbb{\Gamma}_f \times [0, T]\,, && (5.6)\\
[\mathbf{T}] \cdot \boldsymbol{n}_{\mathbb{\Gamma}_f} &= \sigma \kappa \boldsymbol{n}_{\mathbb{\Gamma}_f} && \text{on } \mathbb{\Gamma}_f \times [0, T]\,. && (5.7)
\end{aligned}$$

and is solved for the velocities $\boldsymbol{u}_i : \mathcal{D}_i \times [0, T] \to \mathbb{R}^3$ and pressures $p_i : \mathcal{D}_i \times [0, T] \to \mathbb{R}$ in the respective subdomains with their appropriate couplings given in equations (5.6) and (5.7). As already outlined in Section 3.3, we furthermore have the volume force $\boldsymbol{g} \in \mathbb{R}^3$, the surface tension coefficient $\sigma \in \mathbb{R}$, the interface curvature $\kappa \in \mathbb{R}$, the interface's surface normal $\boldsymbol{n}_{\mathbb{\Gamma}_f} \in \mathbb{R}^3$ and the stress tensor $\boldsymbol{T}_i := -p_i \boldsymbol{I} + (\nabla \boldsymbol{u_i} + \{\nabla \boldsymbol{u_i}\}^T) \in \mathbb{R}^3 \times \mathbb{R}^3$. The jump $(\boldsymbol{T}_1 - \boldsymbol{T}_2)$ along the interface $\mathbb{\Gamma}_f$ is described by $[\boldsymbol{T}]$. In addition, the generalized boundary conditions in equation (5.4) can be replaced by a more complicated set of domain-dependent boundary conditions, cf. Section 3.3.2. For further details and related models, see for example [SSO94, TSLV11, GR11] and [CGS09].

In the following, the above equation system is first modified in a way such that the sharp interface jump condition in (5.7) is replaced by a volume force and densities and viscosities become smooth parameters. Thereafter, the resulting PDE system needs a coupling between the momentum equation and the continuity equation which is done by a classical pressure correction method. Finally, discretization, the iterative solver for the (to be derived) pressure Poisson problem and complex geometries are briefly discussed.

5.1.1 Continuous formulation using level-sets

We now follow the common approach to introduce a level-set function $\phi : \mathcal{D} \times [0, T] \to \mathbb{R}$, which allows to distinguish the two fluid phases, cf. [CGS09]. Level-set techniques have been introduced in [OS88] and early works in the context of two-phase flows are e.g. [SSO94, SF99].

The level-set function ϕ is a signed distance function defining the two domains $\mathcal{D}_1$, $\mathcal{D}_2$ as

$$\phi(\boldsymbol{x}, t) \begin{cases} < 0 & \text{if } \boldsymbol{x} \in \mathcal{D}_1 \\ = 0 & \text{if } \boldsymbol{x} \in \mathbb{F}_f \\ > 0 & \text{if } \boldsymbol{x} \in \mathcal{D}_2 \end{cases} .$$

Also, it obays the Eikonal equation

$$|\nabla \phi| = 1 \,,$$

which makes it a distance function. The free surface $\mathbb{F}_f$, thus the interface between both fluids is given by

$$\mathbb{F}_f(t) = \{\boldsymbol{x} : \phi(\boldsymbol{x}, t) = 0\} \,.$$

Based on the Continuum Surface Force scheme [BKZ92] it is possible to reformulate the discontinuous equation system (5.1)-(5.7) into a continuous representation. This has been proposed by [SSO94] for two-dimensional incompressible two-phase flows and extended in [CGS09] to three dimensions. We follow here the latter work and introduce, by slightly abusing notation, domain-dependent densities and viscosities

$$\rho(\phi) := \rho_1 + (\rho_2 - \rho_1)\, H(\phi), \quad \mu(\phi) := \mu_1 + (\mu_2 - \mu_1)\, H(\phi)$$

with the Heaviside step function

$$H(\phi) := \begin{cases} 0 & \text{if } \phi < 0 \\ \frac{1}{2} & \text{if } \phi = 0 \\ 1 & \text{if } \phi > 0 \end{cases} .$$

This function is smoothed out in ϵ-environment of the free surface leading to jump-free functions $H^\epsilon(\phi)$, $\rho^\epsilon(\phi)$ and $\mu^\epsilon(\phi)$, cf. [CGS09] for more details. It is then possible do derive the initial-boundary value problem

$$\begin{aligned}
\rho^\epsilon(\phi)\frac{D\boldsymbol{u}}{Dt} &= \nabla \cdot (\mu^\epsilon(\phi)\boldsymbol{S}) - \nabla p - \sigma\kappa(\phi)\delta^\epsilon(\phi)\nabla\phi + \rho^\epsilon(\phi)\boldsymbol{g} && \text{in } \mathcal{D} \times [0,T]\,, && (5.8)\\
\nabla \cdot \boldsymbol{u} &= 0 && \text{in } \mathcal{D} \times [0,T] && (5.9)\\
\partial_t \phi + \boldsymbol{u} \cdot \nabla \phi &= 0 && \text{in } \mathcal{D} \times [0,T]\,, && (5.10)\\
|\nabla \phi| &= 1 && \text{on } \mathbb{F} \times [0,T]\,, && (5.11)\\
\boldsymbol{u} &= \boldsymbol{u}_0 && \text{in } \mathcal{D} \times \{0\}\,, && (5.12)\\
\mathcal{B}[\boldsymbol{a}_\mathbb{F}]\boldsymbol{u} &= \boldsymbol{b}_\mathbb{F} && \text{on } \mathbb{F} \times [0,T]\,, && (5.13)\\
\frac{\partial p}{\partial \boldsymbol{n}_\mathbb{F}} &= 0 && \text{on } \mathbb{F} \times [0,T]\,, && (5.14)\\
\phi &= \phi_0 && \text{in } \mathcal{D} \times \{0\}\,, && (5.15)\\
\frac{\partial \phi}{\partial \boldsymbol{n}_\mathbb{F}} &= 0 && \text{on } \mathbb{F} \times [0,T]\,, && (5.16)
\end{aligned}$$

with $\boldsymbol{u} : \mathcal{D} \times [0,T] \to \mathbb{R}^3$ and $p : \mathcal{D} \times [0,T] \to \mathbb{R}$ the velocity and pressure fields defined on the full domain $\mathcal{D}$. Above, the short-hand notations $\frac{D\boldsymbol{u}}{Dt} := \partial_t\boldsymbol{u} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u}$ and $\boldsymbol{S} := (\nabla\boldsymbol{u} + \{\nabla\boldsymbol{u}\}^T)$

were used. It turns out that the jump condition for the stress tensor translates to the volume force $-\sigma\kappa^\epsilon(\phi)\delta^\epsilon(\phi)\nabla\phi$. In this case δ^ϵ denotes a smoothed out Dirac functional and κ is the curvature of the free surface which is given in the level-set case (cf. [OS88]) as

$$\kappa(\phi) = \nabla \cdot \frac{\nabla\phi}{\|\nabla\phi\|} \qquad \text{on } \mathbb{\Gamma}_f \, .$$

Equation (5.10) couples the transport of the level-set function to the velocity field and thus allows to describe the evolution of the phase interface over time. The last five equations are boundary and initial conditions for the velocity, pressure and level-set fields.

5.1.2 Pressure projection approach for two-phase flows

The solution method for the two-phase incompressible Navier-Stokes equations used here is based on a pressure correction approach going back to the work [Cho68]. Following the lines of [CGS09] it is assumed here to use a first-order time discretization with time step size δt. Then the solution method starts by approximating an intermediate velocity field $\boldsymbol{u}^\star$ by the momentum equation (5.8) without the pressure gradient relative to some time step t_n, thus

$$\frac{\boldsymbol{u}^\star - \boldsymbol{u}^n}{\delta t} = -(\boldsymbol{u}^n \cdot \nabla)\boldsymbol{u}^n + \frac{1}{\rho^\epsilon(\phi^n)}\left(\nabla \cdot (\mu^\epsilon(\phi^n)\boldsymbol{S}^n) - \sigma\kappa(\phi^n)\delta^\epsilon(\phi^n)\nabla\phi^n\right) + \boldsymbol{g} \, ,$$

with $\boldsymbol{u}^n(\boldsymbol{x}) := \boldsymbol{u}(\boldsymbol{x}, t^n)$, $\phi^n(\boldsymbol{x}) := \phi(\boldsymbol{x}, t^n)$ and the obvious definition for $\boldsymbol{S}^n$. In the classical sense of a multi-step time integration method, it is then necessary to solve

$$\frac{\boldsymbol{u}^{n+1} - \boldsymbol{u}^\star}{\delta t} + \frac{\nabla p^{n+1}}{\rho^\epsilon(\phi^{n+1})} = 0 \, , \tag{5.17}$$

$$\nabla \cdot \boldsymbol{u}^{n+1} = 0 \tag{5.18}$$

to approximate the Navier-Stokes equations (5.8)-(5.16) for a new time step $t^{n+1} := t^n + \delta t$ ignoring for now the coupling and approximation of the level-set function. Superscripts $n+1$ indicate that the respective fields are given at the new time step. Taking the negative divergence of (5.17) under assumptions of (5.18) leads to a Poisson equation

$$-\nabla \cdot \left(\frac{\delta t}{\rho^\epsilon(\phi^{n+1})}\nabla p^{n+1}\right) = -\nabla \cdot \boldsymbol{u}^\star \tag{5.19}$$

for the pressure at the new time step with the previously imposed homogeneous Neumann boundary condition

$$\left.\frac{\partial p^{n+1}}{\partial \boldsymbol{n}_{\mathbb{\Gamma}}}\right|_{\mathbb{\Gamma}} = 0$$

and some compatibility condition for existence and some closure conditions for uniqueness of solutions for (5.19), cf. [CGS09]. The pressure, which could also be seen as a Lagrange multiplier to couple the momentum and continuity equation, is finally used as correction term

Algorithm 3 Two-phase flow solver algorithm using pressure projection

```
1: for $n = 1, 2, \ldots$ do
2:     set boundary conditions for $\boldsymbol{u}^n$
3:     compute intermediate velocity field $\boldsymbol{u}^*$:
       $$\begin{aligned}\frac{\boldsymbol{u}^* - \boldsymbol{u}^n}{\delta t} &= -(\boldsymbol{u}^n \cdot \nabla)\boldsymbol{u}^n + \frac{1}{\rho^\epsilon(\phi^n)}\nabla \cdot (\mu^\epsilon(\phi^n)\boldsymbol{S}^n) \\ &\quad -\frac{1}{\rho(\phi^n)}\,\sigma\kappa(\phi^n)\delta^\epsilon(\phi^n)\nabla\phi^n + \boldsymbol{g}\end{aligned}$$
4:     apply boundary conditions and transport level-set function:
       $$\phi^* = \phi^n + \delta t\,(\boldsymbol{u}^n \cdot \nabla\phi^n)$$
5:     reinitialize level-set function by solving
       $$\partial_\tau d + \widetilde{\mathrm{sign}}(\phi^*)(|\nabla d| - 1) = 0, \quad d^0 = \phi^*$$
6:     solve the pressure Poisson equation with $\phi^{n+1} = d$:
       $$\begin{aligned}-\nabla \cdot \left(\frac{\delta t}{\rho^\epsilon(\phi^{n+1})}\nabla p^{n+1}\right) &= -\nabla \cdot \boldsymbol{u}^* \\ \left.\frac{\partial p^{n+1}}{\partial \boldsymbol{n}_\Gamma}\right|_\Gamma &= 0\end{aligned}$$
7:     apply velocity correction:
       $$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \frac{\delta t}{\rho(\phi^{n+1})}\,\nabla p^{n+1}$$
```

to derive a divergence-free velocity field $\boldsymbol{u}^{n+1}$ as

$$\boldsymbol{u}^{n+1} = \boldsymbol{u}^* - \frac{\delta t}{\rho^\epsilon(\phi^{n+1})}\,\nabla p^{n+1}\,.$$

Applying higher-order time discretizations instead of first-order explicit Euler is also possible, cf. [CGS09].

Until now, we did ignore the coupling between the momentum / continuity equation and the level-set transport equation assuming to have a level-set field ϕ^{n+1} at the new time step. The solution Algorithm 3 for the two-phase Navier-Stokes equations now outlines this in more detail. Herein, step four and five compute ϕ^{n+1}. The level-set transport in the fourth step uses the old velocity field. Furthermore, the reinitialization process in the fifth step, which is an initial value problem, makes sure that the level-set function approximately obeys the Eikonal equation (5.11). Here, $\widetilde{\mathrm{sign}}$ is a regularized sign function. For more details on the level-set reinitialization process see e.g. [SSO94, CGS09].

5.1.3 Discretization, iterative solver and complex geometries

The flow solver NaSt3DGPF has a large variety of discretization schemes in space and time as well as a series of solvers for the Poisson problem. However, the full multi-GPU implementation discussed here is fixed to a single set of schemes and a single solver, which tend to be the classical choices in standard applications. Therefore, we limit ourselves here to discuss this standard set. More details can be found in [STCE06, CGS09].

Finite differences / volumes are used as discretization method in space with a classical marker-and-cell (MAC) staggered uniform grid, cf. [HW65]. The velocity components are therefore discretized on the centers of the cell faces whereas pressure and level-set function are discretized on the centers of the cells themselves. Wherever it is necessary to evaluate quantities e.g. from cell centers on the cell faces, higher-oder interpolation is used. Most space derivative terms of the momentum equations are discretized by the fifth-order weighted essentially non-oscillatory (WENO) scheme [LOC94]. Here, the diffusion term is computed by second-order central differences. WENO is also applied to the gradient evaluation in the reinitialization equation and to the transport term in the level-set advection. The pressure Poisson equation is discretized with a standard seven-point second order stencil and solved with a Jacobi-preconditioned conjugate gradient (CG) method. In the GPU flow solver, the first-order explicit Euler scheme in Algorithm 3 is replaced by a second-order Adams-Bashforth scheme. This is further optimized by an adaptive time step selection mechanism, cf. [DGN98], which uses the Courant-Friedrichs-Lewy (CFL) condition [CFL67] to optimize for stable calculations. Time discretization for the artificial time τ in the reinitialization is done using the third-order Runge-Kutta scheme.

To include complex geometries in the flow solver, a flagging technique is used which goes back to early work in CFD [HW65]. Cells representing solid obstacles are flagged appropriately and are not considered in the PDE solution process. Additionally, boundary conditions have to be imposed on solid obstacle cells interfacing standard fluid cells. Possible choices of boundary conditions are described in Section 3.3.2.

5.2 Efficient GPU implementation

After having described the numerical model, the discretization and the solution method for the two-phase Navier-Stokes equations, we will now put the focus on the efficient (single-) GPU parallel implementation of the flow solver. Note again, that this is published work by the author [GZ10, ZG13]. To be concise, the reader is expected to know the general technical details of GPUs including especially thread management and memory hierarchies. Otherwise, [SK10, KH10] are good information resources. In this section, the implementation details for the code running on a single GPU are reported, whereas the next section will give details on the implementation for a distributed-memory multi-GPU cluster.

5.2.1 Code design principles

The GPU-based flow solver is implemented with the C/C++ programming extension CUDA [SK10, NVI14b], thus runs mainly on hardware designed by Nvidia Corporation. It is embedded in the original CPU code, however the main loop is fully GPU based. This also includes all data fields, which are stored in GPU memory. The only points, where the CPU and its memory is actively involved are the simulation setup, file / visualization input and output as well as parallel communication.

There is an ongoing discussion about the ease and time of implementing numerical codes compared to the performance that can be achieved. The presented work tries to balance this. While the original CPU code is a classically grown research code and has been developed since more than a decade, the presented GPU implementation reflects several man-months of work with the intention to keep the code itself readable and simple. This effectively means that most operations in the GPU code are done on *global memory* expecting to have more and more effective automatic caching strategies on GPUs available. Only very few, very memory-performance critical operations such as parallel reductions are performed in shared memory for optimal performance. This might lead to an overall performance which does not hit peak performance for a given GPU. However it turns out, that this programming approach leads to performance scalability over subsequent GPU generations, which is a big advantage.

Another important design principle is to always stick to double precision. While single-precision calculations would result in bad numerical approximations especially when going to small scales in the flow solver, mixed-precision techniques could speed up some code parts. However, again, this would lead to a more complicated code which would be highly tuned to some specific single-vs.-double precision performance ratio. As long as performance scalability in double precision is available with new hardware generations, it is therefore planned to stick to this precision.

5.2.2 CPU-GPU porting process

The presented GPU-based two-phase flow solver has been built up by a continuous porting process from the original code. Effectively, the code has been ported to CUDA separately for each C/C++ function with data being copied back and forth between CPU and GPU before and after each method. One usually starts porting the most time consuming routines and ends up unifying all data on GPU, thus removing all data copy operations.

This contrasts other approaches, in which the full code is built up completely from scratch ignoring existing implementations. The advantage of the continuous porting process lies in the ease of validating newly ported parts. Possible errors can be checked method by method. However a *clean sheet* implementation sometimes might allow for better final performance. Due to usually limited time in numerics research to be spend on implementations as-well-as the steadily increasing code performance results in a continuous porting process, this approach has been used to construct the given GPU code.

5.2.3 Thread-data-work mapping

Regular grid based GPU codes with heavy use of stencil operations, like the presented one, usually have a rather simple data layout and mapping between parallel threads, application data and the actual compute work. The given code has a one-to-one mapping of parallel threads to computational cells. Furthermore its original three-dimensional C++ grid data structure is replaced by a linearly allocated and addressed array. In terms of the actual compute work, the classical stencil operation loops are mapped to SIMT (single instruction multiple treads)

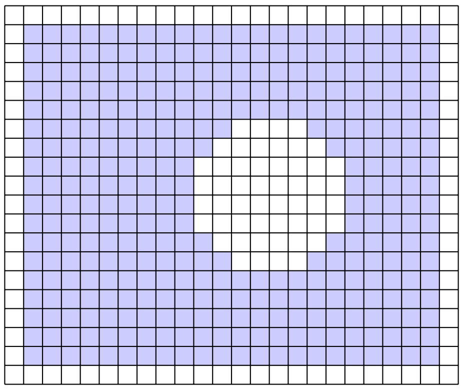


Figure 5.1: Complex geometries and boundary cells in the finite difference scheme lead to irregular data access patterns for data fields. In this 2D example, which includes a discretized circular solid geometry, computations have only to be done for the colored cells.

[NVI14b, Section 4.1] parallel *kernel* calls. Therein thread collaboration in terms of shared memory use is largely avoided as noted above. This strategy also simplifies the choice of the kernel execution configuration, thus the mapping of threads to *symmetric multiprocessors*. Taking a high number of threads per multiprocessor often maximizes occupancy and also gives very good throughput performance. For GT200 / GF100 GPUs usual numbers of threads per symmetric multiprocessor / thread block are thus 256 or 512.

The stencil `for` loops in the sequential C/C++ flow code have a large number of conditionals distinguishing domain boundaries, complex geometries and other special handlings, cf. Figure 5.1. However, it is well-known for GPUs that conditionals might lead to so-called *branch divergence* meaning that the GPU heavily sequentializes the different code execution paths, resulting in bad performance, cf. [NVI14b, Section 12.1] for more details. To reduce the impact of this technological limitation, access pattern data fields are precomputed for all important stencil operations reducing the number of conditionals per GPU thread to exactly one and adding one global memory read. This of course also decreases the available memory. However, a gain in performance of 25% is seen in the modified compute kernel runs resulting in a clear performance gain for the full application.

5.2.4 Large compute kernel handling

Some stencil operations require a rather high amount of compute instructions per computational cell leading to relatively large and complex GPU compute kernels. Early GPU hardware generations and CUDA compilers had the strong issue of using excessive amounts of registers per thread in cases of large compute kernels. Since these did no longer fit into hardware registers, the so-called *register spilling* moved many local variables from hardware registers to special parts of the slow global memory, resulting in bad performance. Even though newer hardware and compiler generations are able to better circumvent this issue, some performance impact remains.

A classical example for this kind of problem is the WENO stencil of fifth order which is heavily used in the discussed flow solver. It computes a series of first-order forward differences and combines these based on some smoothness indicator, cf. [LOC94]. Putting this in one kernel results in very bad performance. A solution is to split up the GPU kernel in a precomputing part, which evaluates the forward differences, and a post-processing part, which computes the smoothness indicators and combines the forward differences. Measurements show, that this approach triples performance for WENO evaluations, which is also partially due to reuse of compute results.

5.2.5 Sparse matrix-vector product and parallel reduction

The preconditioned conjugate gradient solver requires to apply sparse matrix-vector products and parallel reductions with the latter ones being also used in other parts of the code. Both components are performance critical and thus have to be discussed here.

Two-phase incompressible flows have a continuously changing Poisson problem system matrix due to the evolving free surface. Therefore, the matrix has to be constantly updated in each time step. In the given implementation, this is done in a preprocessing step once before the iterative solution process starts. Stencil coefficients are stored cell-wise which is almost equivalent to some diagonal sparse matrix format. The matrix-vector product itself is realized in a hand-coded, optimized GPU compute kernel, which includes the Neumann boundary conditions.

As previously discussed, shared memory accesses are largely avoided to reduce code complexity. The important exception is the applied parallel reduction method. Parallel reductions on GPUs following e.g. [Har07] perform sequential reductions in shared memory on each multiprocessor and combine the results in a tree-like structure. Using global memory here is no option. The specific implementation used in the GPU flow solver stems from [NVI10].

5.3 Multi-GPU parallelization

To scale the flow solver to large distributed-memory compute clusters equipped with GPUs, the MPI-based parallelization structure of the original CPU solver is adapted in a way that one CPU process is mapped to one GPU. The CPU parallelization itself uses the so-called *domain decomposition* technique which is also the base for the multi-GPU parallelization.

5.3.1 Domain decomposition on GPUs

Domain decomposition parallelization goes back to work of Schwarz [Sch90]. The idea is to divide the original computational grid into optimally equally-sized subdomains, which are distributed to different parallel processes, cf. Figure 5.2. Each parallel process then computes locally on its subset. Most PDE discretization schemes require stencil-like evaluations which access information only from very close neighboring cells. Therefore, few computational cells near to the inner subdomain boundaries lack information from their respective parallel neighbor subdomain. To get around this, one or several layers of computational cells, also called *ghost* or *halo* cells, are added at the inner boundaries. These contain copies of the respective parallel neighbor cells which are appropriately updated. Each update requires a parallel data exchange, as shown in Figure 5.2. NaSt3DGPF uses MPI for this parallel communication. The GPU-based implementation handles each subdomain by one GPU instead of one CPU process.

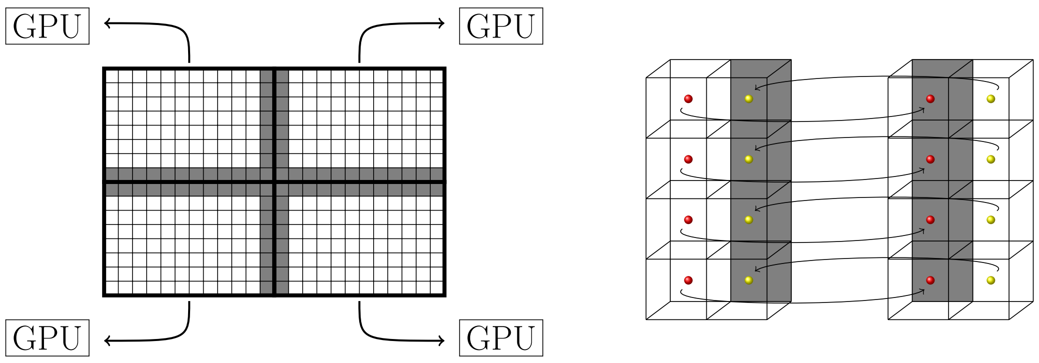


Figure 5.2: Domain decomposition is applied in the multi-GPU parallel code (left) which requires data exchanges between ghost cells of neighboring processes (right).

In classical MPI parallel CPU codes, parallel communication of data fields involves moving data from CPU memory to the network interface over the network and again from the receiving network interface to the memory of the receiving CPU. This transfer pipeline gets more complicated in the multi-GPU data transfer case because data is located in GPU instead of CPU memory. It is therefore necessary to further copy data between GPU and CPU and vice versa on the receiving side, requiring some additional time. Moreover, the higher memory and floating point peak throughput performance on GPUs in contrast to the controlling CPU thread results in a high imbalance of data processing capabilities. Thus, all interactions with the host (i.e. CPU) side are a performance bottleneck in general. Consequently, it is well-known that multi-GPU scalability is sometimes hard to achieve.

To get around this, a few technological and algorithmic optimizations for multi-GPU data transfers have been introduced over the last years.

5.3.2 Optimization strategies

Overlapping computation and communication

One hardware- and software-based optimization is *overlapping computation and communication* [Mic09]. Starting with *Tesla* generation GPUs from Nvidia (e.g. Nvidia Tesla C1070), it is possible to asynchronously launch GPU kernels while data transfers between CPU and GPU (and vice versa) and CPU code execution are performed. Depending on the kind of computational problem, it is thus possible to hide major parts of the necessary parallel communication time behind some long GPU kernel execution.

In the multi-GPU parallel version of the two-phase flow solver, this functionality has a major impact on the parallel scalability of the Poisson solver. As proposed e.g. in [Mic09], the full parallel data transfer can be hidden behind the application of the Poisson matrix stencil to the inner cells of each parallel subdomain, see Figure 5.3. The only remaining operation that has to be done after that concurrent work is the application of the Poisson matrix to the outer cells that depend on the communicated data. This however, is only a small fraction of the original matrix-vector product time.

Furthermore, data transfers between GPU and CPU are done from *pinned* CPU memory,

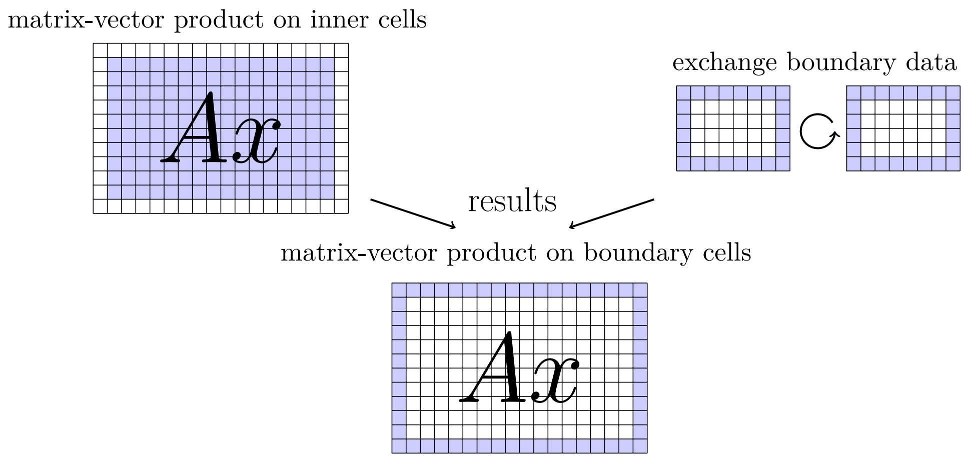

matrix-vector product on inner cells
Ax
exchange boundary data
results
matrix-vector product on boundary cells
Ax

Figure 5.3: It is possible to overlap large parts of the matrix-vector product with the boundary data exchange in the PCG solver. A small matrix subset is applied afterwards.

achieving better performance than classically allocated CPU memory, cf. [NVI14a]. Also, optimized InfiniBand libraries and hardware supporting the technique sometimes referred as *GPUDirect 1.0* are used, removing one additional data copy on the CPU side, cf. [Tec10].

Data prepacking

The performance of data transfers between GPU and CPU memory becomes very bad for many small subsequently transfered messages. This is a big issue when communicating slices from the three-dimensional grid data, as necessary in the ghost cell exchange. Depending on the orientation of the slice, data is highly scattered over memory. Having one transfer operation for each individual data piece would therefore result in prohibitive performance.

To overcome this, all boundary data is packed together in one contiguous data block on GPU before being transfered to CPU. Then, data is unpacked again and transfered by the classical CPU MPI parallelization routines. The same behavior is mirrored on the receiver side, cf. Figure 5.4. Note that unpacking and packing on CPU is also overlapped with GPU computation as described before.

Recent developments

More recently, some effort has been spend on achieving more seamless integration of GPU calculations into standard MPI parallel codes. There are now CUDA-aware MPI implementations [PWB$^+$12, BBR13] available, which allow to use a subset of the full MPI operation set directly on GPU memory, taking care for transfers to / from CPU memory by themselves. The advantage is better code readability and out-of-the-box optimizations inside the MPI library, such as interleaved message transfers for smaller latency of big messages. Since it is not expected to see significant speed-ups based on CUDA-aware MPI within the parallel communication of this specific multi-GPU parallel flow solver, this new technology was not added after finalizing the multi-GPU porting process.

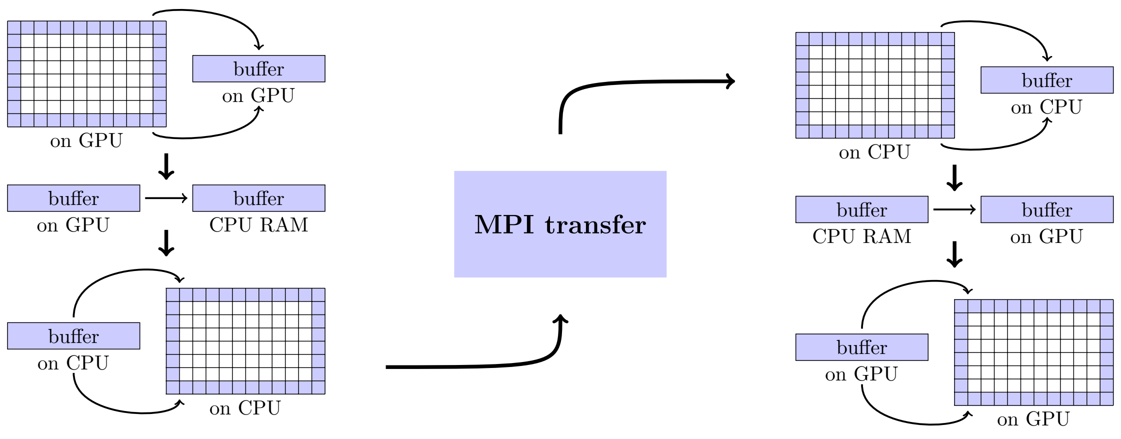


Figure 5.4: To overcome limitations of CPU-GPU data exchanges with respect to small messages, all boundary data is packed into a contiguous memory buffer as part of the multi-GPU data exchange.

Another very recent development is the use of Remote Direct Memory Access (RDMA) capabilities of the PCI Express bus. By combining new *Kepler*-generation GPUs and new InfiniBand hardware, it is possible to use what is called *GPUDirect RDMA* [BBR13]. This technology largely avoids data transfers through CPU memory by directly exchanging data between GPUs in one system or between a GPU and an InfiniBand network controller over the PCI Express bus, leading to a much better latency and sometimes to a better bandwidth. Due to the lack of appropriate hardware, this approach is not further investigated.

5.4 Performance results

There is a two-fold motivation to use GPUs to solve the two-phase incompressible Navier-Stokes equations as black-box problem inside a stochastic collocation method. On the one side, it is necessary to evaluate new massive-parallel technologies to be prepared for future large-scale compute clusters. On the other side, a clear performance gain compared to CPUs is expected. In the following, an in-depth performance analysis will show some significant speed-ups while maintaining good multi-GPU scalability.

5.4.1 Validation and best practice in GPU-CPU comparisons

Before we can discuss performance, we need to be sure to have a correct code. Validation has been made throughout the full porting process of the GPU flow solver between GPU and CPU implementation. Note that the CPU implementation has been already validated against experimental results, cf. [CGS04]. It was possible to have a method-by-method comparison of all simulation data fields between GPU and CPU. Since identical numerical techniques are applied, the ported code has to create matching simulation results up to machine accuracy with the only exception of parallel reductions in e.g. scalar products which have different summation orders on GPU and CPU.

In fact, equal results up to machine accuracy are observed for all routines not performing a parallel reduction. Overall, GPU simulation results are equal to CPU results up to discretization error and convergence order also including parallel reductions.

There is an ongoing discussion on best practice in GPU vs. CPU benchmarking. Comparing one full GPU to a single CPU core results in large speed-up numbers but it is of course unbalanced. Comparing a full CPU on a socket to one GPU is much better, but often does not reflect price differences for both hardware. Therefore, a comparison of equally priced hardware is preferred in the following. Noting that in a GPU-centric compute node design, the CPU is only necessary as a controller with weak performance and neglectable price, price comparison will be based on the GPU (without CPU) and multi-core/multi-socket CPUs only. An additional GPU to CPU comparison will be highlighting power consumption of the full GPU/CPU systems.

5.4.2 Benchmark setup

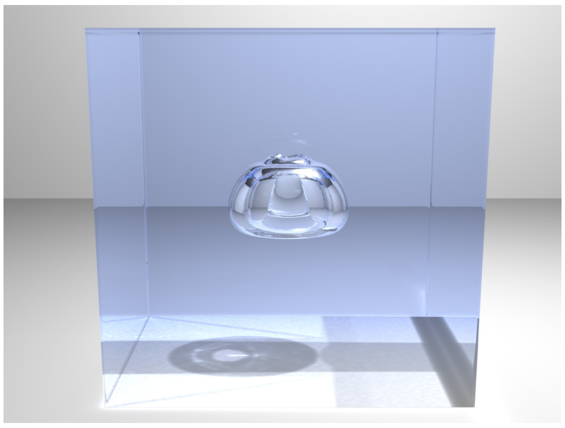
Overall three GPU configurations are used to perform the necessary benchmarks. The first one is an eight-GPU cluster based on two Nvidia Tesla S1070 GPU systems with four GT200 Tesla GPUs which are equivalent to M1060 GPUs. Each S1070 is connected to a host system with Intel Core i7-920 CPU at 2.66 GHz. The host systems are connected with a Mellanox ConnectX QDR 40G InfiniBand interconnect. At the time of performing these benchmarks, each M1060 GPU had an almost equal price to an Intel Xeon X5650 six-core CPU at 2.66 GHz. Such a system is taken for the price-based performance comparison.

The second GPU configuration is a 1U compute node with a four-core Intel Xeon E5620 CPU at 2.40 GHz and an installed Nvidia Tesla C2050 GPU, thus a GF100 Fermi generation processor. This GPU had more or less twice the price of one M1060 GPU when the benchmark was performed. Therefore, it will be compared to a dual 6-core Intel Xeon X5650 CPU system at 2.66 GHz each, thus to twelve CPU cores.

As common software framework for both GPU configurations Ubuntu Linux 10.04, OpenMPI, GCC 4.4 and CUDA 3.2 are used. CPU code is compiled with optimization flags `-O3 -march=native` and GPU code is compiled with `-O3` in general.

To test scalability over a larger number of GPUs, the GPU cluster of the Center for Computing and Communication of the RWTH Aachen University, Germany was used over the *Ressourcenverbund – Nordrhein-Westfalen* (RV-NRW) access scheme. This is the third GPU configuration. Access was given to 48 Fermi-type GPUs, thus 48 Nvidia Quadro 6000 GPUs installed in 24 dual Intel Xeon X5650 CPU systems with QDR InfiniBand interconnect. The available software framework is composed of Scientific Linux 6.1, GCC 4.4.5, OpenMPI 1.5.3 and CUDA 4.0.17.

Table 5.1 highlights the benchmark flow problem applied for the simulation tests. It is a rising bubble in water which reshapes over time, thus a real-world example representative for our typical applications. All measured timings reflect CPU wall-clock times including all data transfers between GPU and CPU (and vice versa). The run-times of a fixed number of twenty simulation time steps is computed at the given resolutions to compare between GPU and CPU. A resolution of $256 \times 256 \times 128$ e.g. leads to a compute time of about 51 minutes on one six-core Xeon X5650 processor using all cores.



domain size:	$20\,cm \times 20\,cm \times 20\,cm$
liquid phase:	water at 20^oC
gas phase:	air at 20^oC
volume forces:	standard gravity
init. bubble radius:	$3\,cm$
init. bubble center:	$(10\,cm, 6\,cm, 10\,cm)$

Table 5.1: The performance benchmark example is an air bubble rising in water, visualised on the left according to [ZG11] with the picture taken from [ZG13]. Its paramters are given on the right-hand side.

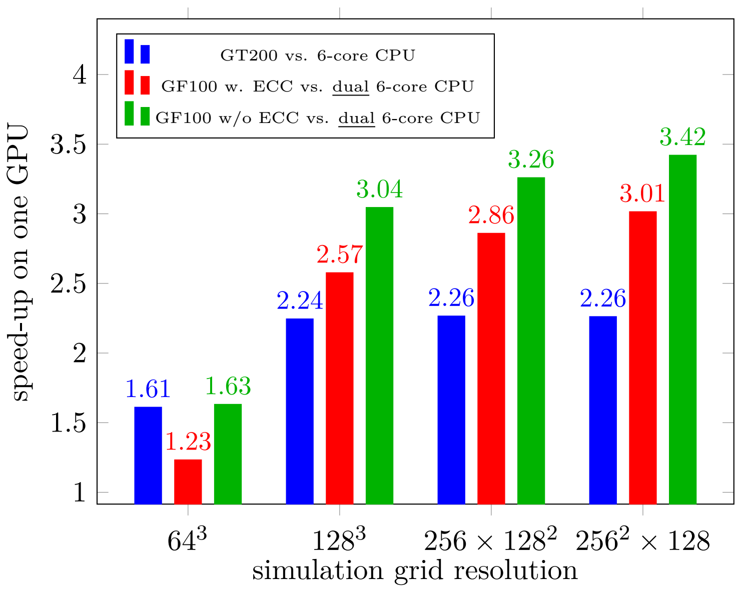
CPU/GPU type	double-precision floating-point peak performance	ratio	peak memory bandwidth	ratio
Intel Xeon X5650	∼64 Gflops		32 GB/s	
Nvidia M1060 (no ECC)	78 Gflops	1.2x	102 GB/s	3.2x
two Intel Xeon X5650s	∼128 Gflops		64 GB/s	
Nvidia C2050 (no ECC)	515 Gflops	4x	148 GB/s	2.3x

Table 5.2: CPUs and GPUs have different performance characteristics. The ratios reflect the speed-ups between the grouped CPUs and GPUs with respect to double-precision floating-point performance and memory bandwidth. Gflops numbers for the CPUs are accumulated values for the 6 / 12 physical cores and are extracted from the Top500 list as of June 2011 [Top11].

5.4.3 Performance expectations

As outlined in the code design principles, cf. Section 5.2.1, we compare a grown, active research code on the CPU side with a recently ported GPU version, which was not subject to excessive micro-benchmarking. This should be rather fair in terms of achievable performance and similar level of optimization.

Before we have a look at the actual performance measurements, we need to clarify what we can expect from the given hardware. To get a first impression, one can compare peak performance ratios for floating-point operations and memory bandwidth on the GPUs and CPUs under discussion. Table 5.2 summarizes the results. Note that non-ECC protected performance numbers are given for the GPUs with an expected further loss in performance of 8% when going to ECC corrected results. Comparing the GT200 GPU with the equivalently priced CPU hardware gives only a speedup of 1.2x in terms of floating point performance but more than a factor of three in memory bandwidth. Since the time-dominating part of the



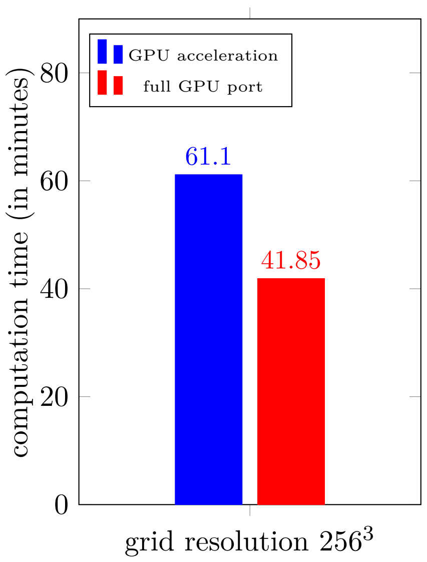


Figure 5.5: *Left:* Speed-ups of single GPUs are compared to *similar-priced* multi-core CPUs for different resolutions and different GPU generations. *Right:* By fully porting the fluid solver to the GPU we get more than a 30 percent improvement in runtime.

flow solver is the Poisson solver, which is largely memory bandwidth bound, the good memory bandwidth ratio is expected to dominate here. In the case of the GF100 GPU, the floating point performance ratio is a factor of four and the memory bandwidth ratio is a factor of 2.3. Therefore, it is expected to see speedups in the range of 2x to 3x for the price-based performance comparison.

5.4.4 Single-GPU performance analysis

All results of the price based single-GPU performance comparison are summarized in the left part of Figure 5.5. Both GPU generations, Tesla and Fermi, show the performance characteristics of growing speed-up for growing problem size until the implementation- and hardware-dependent peak performance is hit. Larger problem sizes lead to higher speed-ups e.g. due to better latency hiding for memory fetches.

There is a large variety of execution configuration choices with respect to maximum register count per kernel on GT200 GPUs and furthermore concerning compute architecture and cache configuration on GF100 GPUs. For more details on their impact on the presented GPU code, the interested reader is asked to refer to [ZG13, Section 4.4]. Here, results are only discussed for the best possible per GPU kernel configuration parameter choices.

Comparing the older GT200 GPU to a 6-core CPU at an equal price point, a speed-up of more than a factor of two is achieved. This is what could be expected. Moving over to the newer hardware, i.e. GF100 GPUs, results in speed-ups of a factor of three, comparing that single GPU to a parallel run on a 12-core CPU system. Without memory error checking and correction this speed-up is even higher. These impressive performance results are at the upper

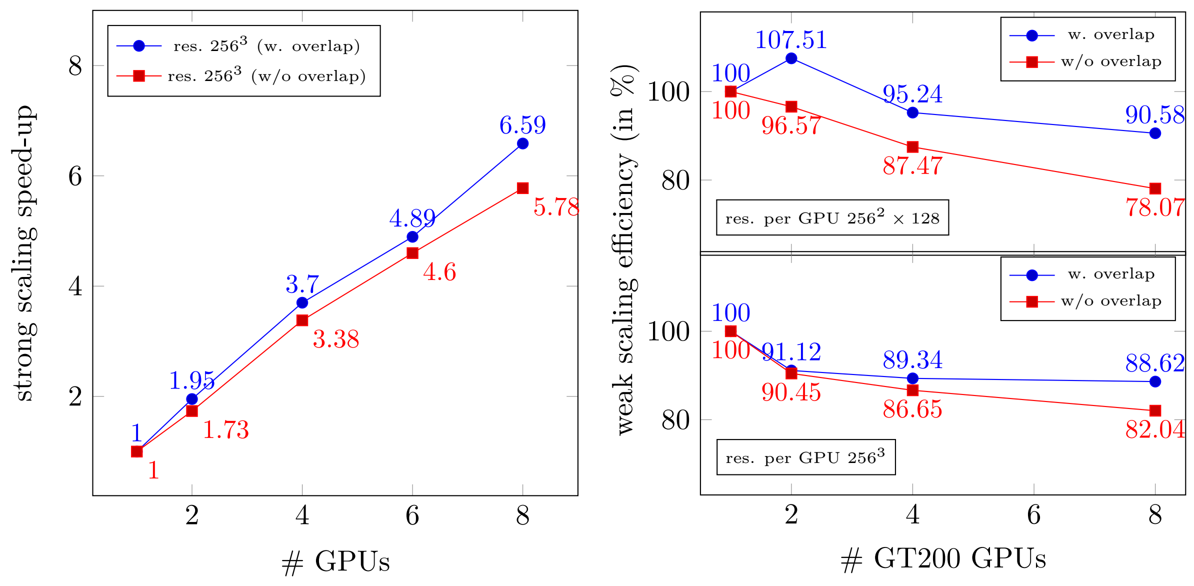


Figure 5.6: Strong scaling (i.e. speed-up) results (left) and weak scaling (i.e. scale-up) efficiency results (right) relative to one GPU on the *GT200-based* GPU cluster with and without overlapping of computation and communication.

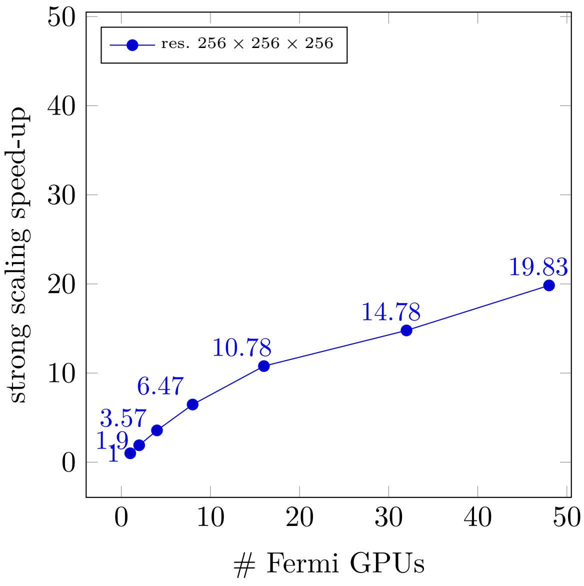
limit of what could be expected. Here, clearly, the automatic caching of Fermi GPUs is a big advantage. Finally, we can also state that the Fermi GPU is almost a factor of three faster than the Tesla GPU because of higher floating-point and memory throughput performance and caches.

It is also important to discuss the improvement of fully porting a code to GPU instead of acceleration on GPU, cf. the right-hand side part of Figure 5.5. In [GZ10], an acceleration of NaSt3DGPF, by porting a large part of the Poisson solver and the level-set reinitialization to GPUs, is outlined. Comparing execution times of this code with the fully ported GPU code as documented in [ZG13], one can see more than a 30% percent performance gain. A reason for this is that the accelerated Poisson solver already had a rather dominant performance gain. Other, more performance-wise balanced codes, might see a higher impact from full GPU ports.

5.4.5 Multi-GPU parallel scalability

After having discussed single-GPU speed-ups, now multi-GPU parallelization benchmark studies are outlined. Let us start with the Tesla-generation cluster with eight GPUs. The motherboards of the two host CPU systems have up to 48 PCI Express 2.0 lanes connected to one chipset. Obviously, running several GPUs connected to one CPU chipset might harm performance. To keep this performance limitation balanced, an equal number of GPUs is used on each machine, if possible. Therefore, in the two GPU case, each host system controls one GPU and in the four GPU case, each host system controls two GPUs.

Figure 5.6, on the left-hand side, outlines strong scaling or speed-up results comparing the runtime on several GPUs to the respective time on one GPU. The grid size of the benchmark flow simulation problem is 256^3. Results with and without the overlapping computation and



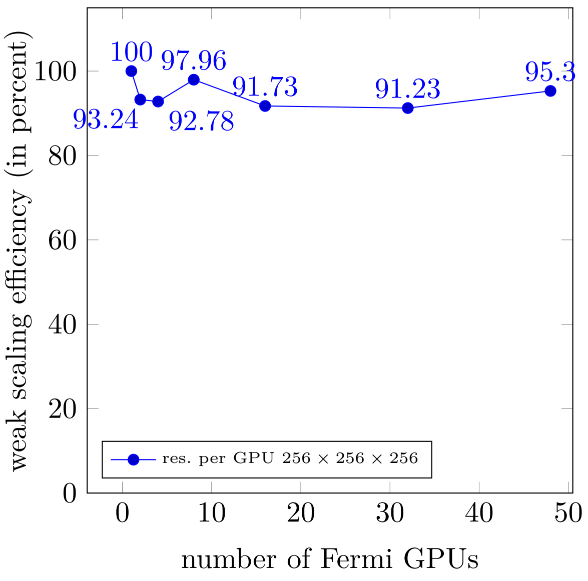


Figure 5.7: Strong scaling, i.e. speed-up, results (left) and weak scaling, i.e. scale-up results (right) relative to one GPU on the *Fermi-based* GPU cluster with overlapping of computation and communication.

communication technique, cf. Section 5.3.2, are shown, knowing that it can only be applied effectively for the Poisson solver. Due to the overlapping technique, it is possible to achieve a rather linear scaling up to eight GPUs with a maximum speed-up of 6.59. This is equivalent to a parallel strong scaling efficiency of above 82%, which is a decent result. Without overlapping, the speed-up on eight GPUs drops to 5.78 equivalent to only 72% parallel strong scaling efficiency.

Next, the same hardware is considered in a weak scaling / scale-up analysis, again for the full flow solver. In this setting, the number of grid points is fixed per GPU and the overall problem size is increased with growing GPU number. Figure 5.6, on the right-hand side, gives results for the per-GPU resolutions of $256^2 \times 128$ and 256^3 each time with and without overlapping in the Poisson solver. Both studies show an almost excellent weak scaling behavior on up to eight GPUs with roughly 90% parallel weak scaling efficiency when using the overlapping strategy. The drop in performance by not overlapping gets more pronounced for larger GPU counts. In the $256^2 \times 128$ case, one can observe an unexpected performance increase up to 107.51% parallel efficiency. This behavior is often due to array alignment effects.

The second multi-GPU test system features newer hardware and larger GPU counts. In the strong scaling analysis, cf. Figure 5.7 on the left-hand side, almost identical speed-ups as on the smaller GPU cluster can be observed for up to eight GPUs, noting that overlapping of computation and communication is applied in this test case. For higher GPU counts, there is a more pronounced drop in parallel strong scaling scalability. This is a well-known behavior that can be easily understood by reconsidering the left-hand side of Figure 5.5. Larger GPU counts lead to smaller per-GPU problem sizes with much worse performance. Nevertheless, it is still possible to achieve more than 40% strong scaling parallel efficiency on 48 GPUs.

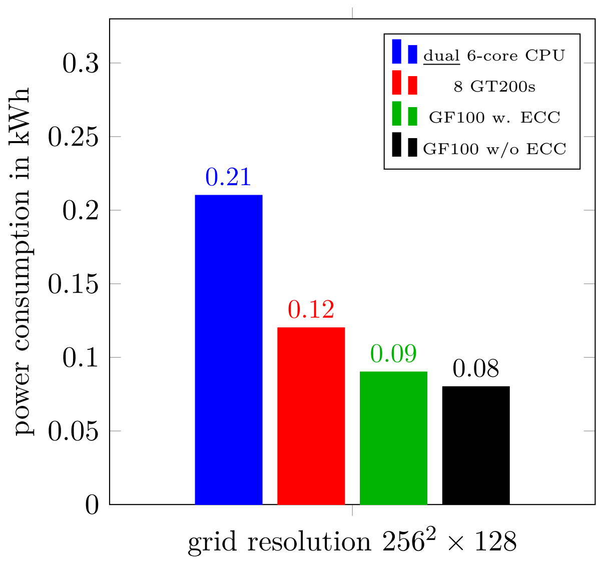


Figure 5.8: The measured power consumption figures show that the Fermi GPU with ECC is about a factor of 2.3 more power-efficient compared to standard CPUs.

In the final multi-GPU result, weak scaling efficiency with overlapping of computation is presented for a per-GPU problem size of 256^3 grid cells, cf. the right-hand side of Figure 5.7. It is rather impressive to observe an always above 91% parallel weak scaling efficiency for up to 48 GPUs. Clearly, a higher per-GPU PCI Express lane count and InfiniBand bandwidth lead to this kind of result compared to the eight GPU cluster. Fluctuation in the results is due to different cluster loads and array alignments. Note again, that this almost perfect scalability results is achieved on the newer GPU hardware generation and on more GPUs.

5.4.6 Power consumption

As last benchmark, the necessary energy consumption for a simulation problem with fixed size, thus again the first 20 time steps of the $256 \times 256 \times 128$ resolution benchmark simulation, is measured. Since energy monitoring of a single hardware component is rather involving, power consumption is measured from the full test systems. Figure 5.8 gives results of this study. The 12-core CPU system has the largest power consumption even when using all twelve cores. Surprisingly, the full eight GPU cluster already has a much better power consumption for a fixed problem size. Best results are achieved on Fermi generation GPUs. Here ECC-protected calculations are about a factor of 2.3 more power-efficient than CPU calculations, which is a quite impressive result. Without ECC protection, these results are even a bit better.

5.4.7 Performance summary and discussion

The performance analysis given before clearly outlines that GPUs in that given configuration outperform CPUs even in an equally priced hardware comparison. Here, for Fermi-generation GPUs, a speed-up of a factor of three is seen. At the same time, it is possible to achieve almost optimal parallel scale-up results even on a larger number of GPUs. Together with the more than halved energy consumption, theses numbers give a clear hint that GPUs and GPU-like hardware architectures form a good base for future hardware generations. Programmability is also acceptable if code optimization and performance expectations are balanced appropriately. Overall, the best possible performance base is given for the stochastic moment approximation in two-phase flows.

6 Numerical studies and empirical error coupling

This chapter concludes the first part of this thesis by giving numerical results for the proposed new method. We start by addressing numerical convergence studies and performance results for model and large-scale application problems. This is followed by a section on the convergence comparison to classical sparse and full tensor product methods in the field of uncertainty quantification. Finally, the topic of empirical error coupling is addressed.

The first section follows the model and application problems as introduced in Chapter 3. Error behavior of the stochastic moment approximation by the RBF kernel-based stochastic collocation method with respect to a growing number of collocation points will be studied. These results will allow to identify the strength of the proposed method. Note that there will be a series of real-world applications with respect to the random two-phase incompressible Navier-Stokes equations. Since most of the real-world problems are heavily limited by computational time, efficient implementations are of high interest. Therefore, all large-scale problem convergence results will be accompanied with performance results of the implemented numerical algorithms.

Afterwards, comparisons with standard numerical techniques in the field will be given. These will be the (generalized) Polynomial Chaos Expansion (PCE) [GS91] and stochastic collocation [NTW08b, BNT10] both by full and sparse spectral tensor product constructions. The state-of-the art uncertainty quantification framework *Dakota* [ABB$^+$09a] will be applied as comparison base for some of the model and application problems. It will turn out that kernel-based stochastic collocation is able to outperform full tensor product and even sparse grid-based methods in the preasymptotic regime in many cases. Furthermore, the proposed method achieves better convergence rates for problems, which have limited smoothness in the stochastic parameters.

The final part of this chapter highlights an empirical error coupling analysis of the given numerical method for a random elliptic and a random two-phase Navier-Stokes problem. This analysis will outline the dependence of the numerical error on the different approximations. Based on this, all involved errors will be balanced with respect to a fixed overall error tolerance. By that way, the complexity analysis of Section 4.7 can be further refined such that empirical complexity results with respect to a given error tolerance can be given. These will again motivate the work on preconditioning techniques in the second part of this thesis.

6.1 Convergence and performance studies

This section follows the model and application problems outlined in Chapter 3 to show numerical convergence results. Before we discuss the results, it is necessary to define the framework for the numerical tests.

6.1.1 Benchmark setup

All numerical studies for the RBF kernel stochastic collocation methods are carried out with the implemented GPU-based numerical software as described in Section 4.8.

Space discretization The two-dimensional random-coefficient elliptic problems, cf. Section 3.2, are discretized on a uniform grid by a standard five-point finite difference stencil with domain- and stochastics-dependent coefficients. To solve the resulting linear system, a GPU-based Jacobi-preconditioned conjugate gradient solver is used from the linear solvers framework CUSP, cf. Section 4.8. The iterative solver stops if the absolute residual $\boldsymbol{r}$ norm drops below $\|\boldsymbol{r}\|_2 = 10^{-15}$. If not stated otherwise, the number of degrees of freedom per coordinate direction is 512, thus we have to solve for $N_{\mathcal{D}} = 512^2$ unknowns.

Details on discretization in the multi-GPU based incompressible two-phase Navier-Stokes solver are already discussed in Chapter 5. Problem-specific information, such as space grid resolution, is given together with the corresponding numerical results.

Stochastic approximation Approximation in stochastic space is done by the RBF kernel-based stochastic collocation method. If not stated otherwise, the applied kernel functions are the Gaussian kernel k_ϵ with scaling parameter $\epsilon = 1.0$, compactly supported Wendland kernels $k_{N_{FN},k}$ with smoothness parameters $k = 0, 1, 2, 3$ and appropriate dimensionality N_{FN} and the Matérn kernel k_β with parameter $\beta = \frac{N_{NF}+3}{2}$. Remember that we call the dimension of the stochastic space N_{FN}, thus $\Gamma \subset \mathbb{R}^{N_{FN}}$. However, if the random input is modeled by a Karhunen-Loève expansion, we use the parameter N_{KL}, instead, leading to $\Gamma \subset \mathbb{R}^{N_{KL}}$. Regularization parameter ϵ_{reg}, cf. Section 4.3.1, is set to

$$\epsilon_{reg} = 10^{-12} .$$

The kernel interpolation problem is solved by direct LU factorization, cf. Section 4.5.3. Radial basis functions are isotropic, by standard, thus the norm involved in their construction is the (scaled) Euclidean distance $\|\cdot\| := \sigma\|\cdot\|_2$, with a default of $\sigma = 1.0$. The N_Γ collocation points are generated from a Halton sequence, cf. Section 4.5.2, of appropriate dimension. Furthermore, quadrature is carried out by a full tensor product rule $Q_F^{l_q,N_{FN}}$ constructed by univariate Clenshaw-Curtis quadrature rules $Q_{CC}^{l_q,1}$, cf. Section 4.5.1, if not indicated differently. The quadrature-level per dimension is usually $l_q = 7$. Sparse grid quadrature rules $Q_S^{l_q,N_{FN}}$ are used whenever the dimensionality of the stochastic space would lead to prohibitive computational run-times and memory requirements.

Error measurements The reference solution will be defined separately for each model and application problem. Analytic solutions or so-called *overkill* solutions, thus approximate solutions with a reasonable level of convergence, will be taken. In cases, in which the quantity of interest is a single number, relative errors, thus

$$\varepsilon_{rel} := \frac{|v_{sol} - v_{ref}|}{|v_{ref}|} ,$$

for v_{sol} the approximate solution and v_{ref} the reference solution, are computed. If stochastic moments of space-dependent, discretized random fields are evaluated, the error norm is the discrete analogon of the L_2 norm, which we call (slightly abusing the usual terminology) l_2 norm and define it by

$$\|\boldsymbol{v}\|_{l_2} := \left(\frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} {v_i}^2 \right)^{\frac{1}{2}} , \qquad \forall \boldsymbol{v} = (v_1, \ldots, v_{N_{\mathcal{D}}})^\top \in \mathbb{R}^{N_{\mathcal{D}}} .$$

We then define the (absolute) error by

$$\varepsilon_{abs} := \left\|\boldsymbol{v}_{sol} - \boldsymbol{v}_{ref}\right\|_{l_2} .$$

Convergence analysis structure The convergence studies per model and application problem have a similar structure. They start by a convergence comparison of the kernel-based method for different kernels. All convergence plots are given with respect to the number of collocation points, since this is the only important quantity in most real-world applications in which each solution of the underlying deterministic PDE is in the range of hours of compute time. In those cases, where the dimension of the stochastic space Γ is not fixed by construction, very often, a further convergence study is made. It shows the dependence of the kernel-based method on the dimension of the underlying problem. These studies are often followed up by a second moment convergence analysis, thus fields $\mathbb{E}\left[\boldsymbol{u}^2\right]$ are approximated.

Performance measurements All flow problems are additionally analyzed with respect to their performance on GPU hardware. The reference hardware is a GPU cluster consisting of up to 36 nodes each equipped with a four-core Intel Xeon E5620 CPU at 2.4 GHz, 12 GB DDR3 RAM, an Nvidia Tesla M2090 card with 6 GB DDR5 RAM and Mellanox ConnectX-2 QDR InfiniBand 40Gbps interconnect. The operating system is Ubuntu 12.04. Furthermore, CUDA 5.0, CULA R16a and OpenMPI 1.7.5 are in use. The stochastic collocation GPU code is compiled with optimization parameters `-O3 -arch sm_20` and runs here on a single GPU. Compilation optimization of the flow solver is discussed in Section 5.4.2.

Runtime measurements of the stochastic collocation code includes the full code runtime (with file I/O) measured by the command `time`. An average out of three code runs is taken. Timings of the flow solver are measured by the application itself with appropriate MPI calls. To account for the large measurement variations, due to phases of high GPU cluster load, flow solver runtimes are averaged throughout all realizations leading to an idealized fixed runtime per solution realization.

6.1.2 Problems with analytic solution

One-dimensional random-coefficient elliptic problem

This simple model problem is defined in Section 3.1.1. To avoid the influence of any approximative PDE solution method, the exactly known solution is taken as response surface function, thus no PDE solver is involved at all. The discretization grid is a 512×512 uniform grid.

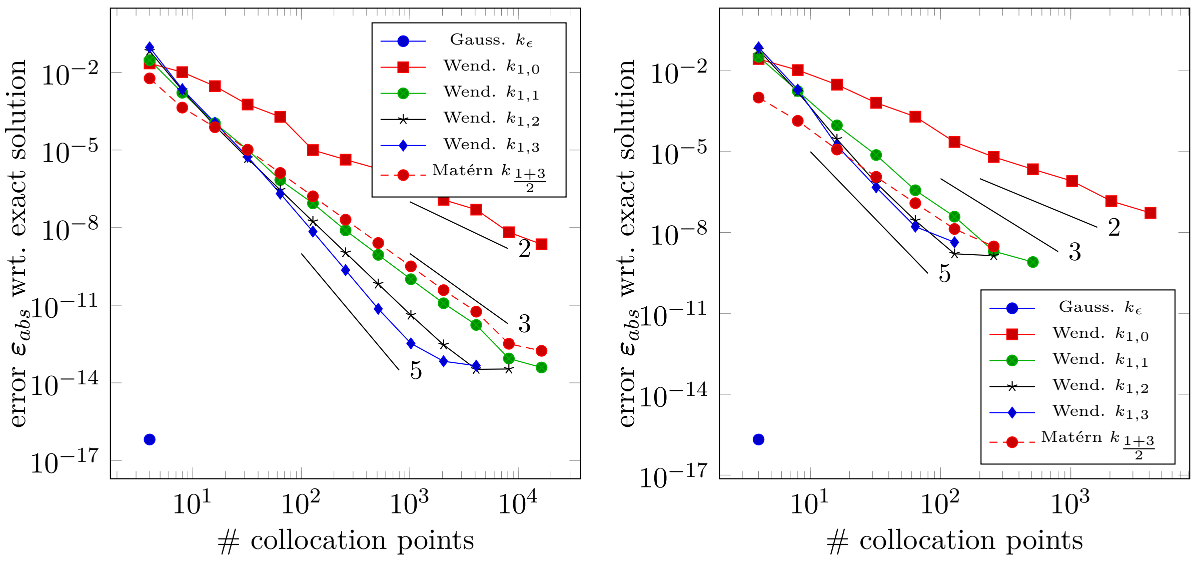


Figure 6.1: Error behavior of the expected solution field (left) and the second moment of the solution field (right) in case of the random-coefficient Poisson problem with analytic solution.

The convergence study for the mean solution field is presented in Figure 6.1 on the left-hand side. It shows the l_2 error with respect to the exactly known solution. The regularization parameter ϵ_{reg} is set to 10^{-15} for the Gaussian kernel. All other parameters remain at the defined standard. Approximation with the Gaussian kernel leads to a solution at machine precision with only four collocation points. This highlights the exponential convergence by using Gaussian kernels, if the response function is smooth enough. All other kernels have algebraic convergence rates with measured approximate orders 2, 3, 4 and 5 for Wendland kernels with $k = 0, 1, 2, 3$ and third-order convergence for that specific choice of a Matérn kernel. Convergence saturates between machine accuracy and the regularization, remembering that the regularization is set to $\epsilon_{reg} = 10^{-12}$ for these kernels, in contrast to the Gaussian. This is textbook-convergence.

The second moment of the one-dimensional random Poisson problem can be computed exactly from the known analytic solution. Figure 6.1 gives on the right-hand side convergence results with respect to the second moment. The error convergence rates are almost the same as in the study for the mean. However the error decay stagnates earlier at about 10^{-9} due to the fixed quadrature level.

g function

This second study with analytic solution is described further in Section 3.1.2. Its advantage is the ease of setting up problems with higher stochastic dimension. Note that the problem has limited smoothness in the stochastic parameter due to the absolute value operator. Relative (scalar) error is used in all numerical tests. The scaling of the Gaussian kernel is set to $\epsilon = 2.0$ in contrast to the defined standard. Furthermore the approximation by the Gaussian kernel is regularized with a regularization parameter $\epsilon_{reg} = 10^{-8}$.

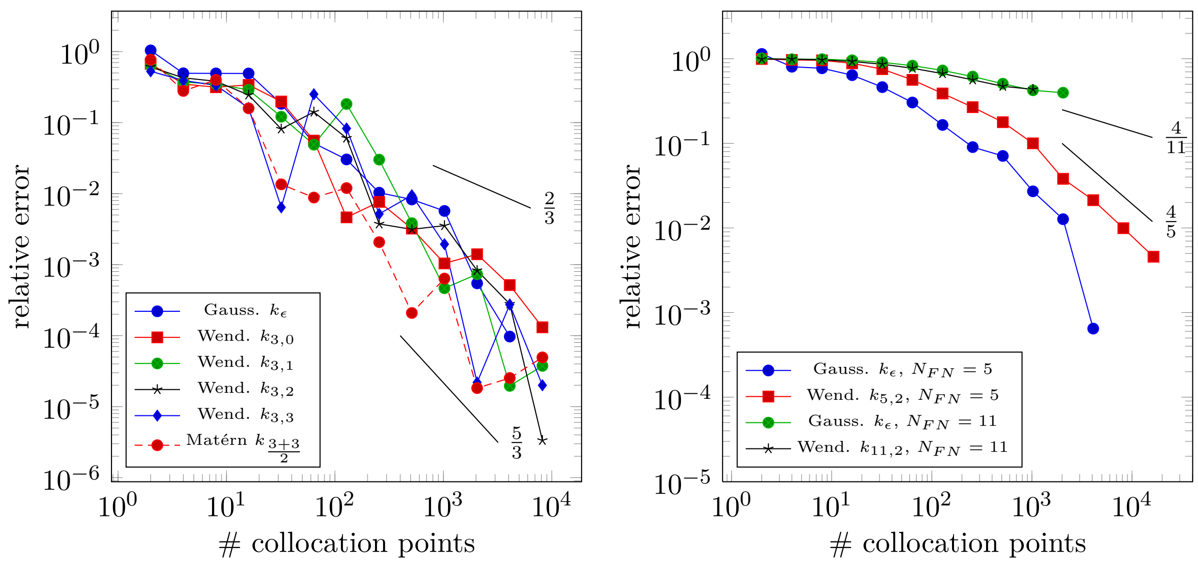


Figure 6.2: Error behavior of the mean of the g function using different kernels and stochastic dimension $N_{FN} = 3$ (left) and error behavior for changing dimension (right).

For the first kernel comparison study, the stochastic dimension is set to $N_{FN} = 3$. Figure 6.2 shows on the left-hand side the error behavior of the mean of this model problem for different kernels. The results do not show clear convergence rates, probably due to missing regularity of the g function. Nevertheless, we would like to understand, whether the measured results comply with knowledge on dimension-dependence from theory. When it comes to Wendland kernels, Theorem 4.5 in Section 4.3.2 suggests dimension-independent convergence rates with respect to the fill distance $h_{X,\Omega}$, if the function that shall be interpolated is in the respective native kernel space. We have to keep in mind that, due to the quasi-uniform distribution of Halton points, the very rough guess of $h_{X,\Omega} \sim {N_\Gamma}^{-1/N_{FN}}$ is possible, cf. [Wen04, Section 14.1]. Therefore, we should expect to see convergence rates, which are multiples of the inverse stochastic dimension. Combining this knowledge with the results from the first convergence studies, Wendland kernels are expected to show convergence rates (in the number of collocation points) ranging from $\frac{2}{3}$ to $\frac{5}{3}$. This is compatible with the presented results.

In addition to the general kernel comparison for stochastic dimension $N_{FN} = 3$, Figure 6.2 outlines on the right-hand side further convergence results for dimensions $N_{FN} = 5$ and $N_{FN} = 11$. Note that this study is computationally challenging, since the tensor product quadrature shows an exponential increase in the number of quadrature points with respect to the dimension. Due to computational runtime restrictions, the quadrature level is reduced to $l_q = 4$ in the five dimensional case and the eleven-dimensional case is computed by the sparse grid combination technique quadrature $Q_C^{l_q,N_{FN}}$ constructed by univariate Clenshaw-Curtis quadrature rules $Q_{CC}^{l_q,1}$ with quadrature level $l_q = 4$, cf. Section 4.5.1. To be concise, Matérn kernels are not considered.

For $N_{FN} = 5$, the Wendland kernel shows a convergence rate of roughly $\frac{4}{5}$, which fits into the dimension-dependence, and first indicators of higher-order to exponential convergence for the Gaussian kernel. The problem with eleven dimensions is more involved. Here, the Gaussian and Wendland kernel are still in the preasymptotic regime.

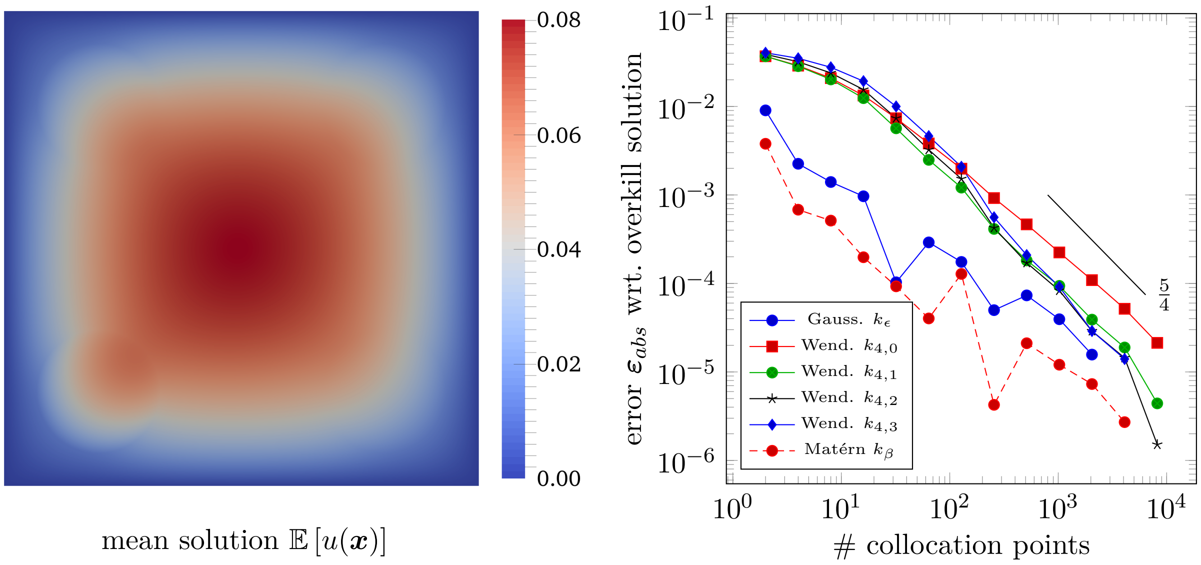


Figure 6.3: Mean solution field (left) and error behavior of the expected solution field of the two-dimensional random elliptic problem with piecewise constant random diffusion field for fixed stochastic dimension $N_{FN} = 4$ and growing number of collocation points based on a Halton sequence (right).

6.1.3 Two-dimensional random-coefficient elliptic problems

The next set of model problems is often investigated in papers on uncertainty quantification for elliptic PDE problems with random coefficients, cf. [NTW08b, NTW08a, BNT10, BNTT11]. They are more computationally challenging than the previous problems since a PDE has to be solved for each collocation point. Also, in general, there is no analytically known solution, thus overkill solutions have to be used throughout these studies.

Piecewise constant random diffusion field

Remembering the definition of this model problem in Section 3.2.1, it is generated by a finite noise assumption without Karhunen-Loève expansion. Here, the diffusion coefficient field is piecewise constant, with four regions of stochastically disturbed coefficients leading to a constant stochastic dimension of $N_{FN} = 4$. Quadrature to compute the coefficients is done by full tensor-product quadrature on level $l_q = 5$.

The comparison of mean solution field convergence for different kernel functions with the RBF kernel stochastic collocation framework is outlined in Figure 6.3 on the right-hand side. As reference solution, the solution field for a Wendland kernel $k_{4,3}$ at $N_\Gamma = 2^{14}$ collocation points is used and visualized on the left-hand side of Figure 6.3. Error is computed as the l_2 residual between the reference solution mean field and the solution mean field under investigation. The results show a convergence rate of around $\frac{5}{4}$ for the highest-order Wendland kernels. In this case, the Gaussian kernel seems to be still in the preasymptotic regime. The Matérn kernel has a better constant than the Wendland kernels.

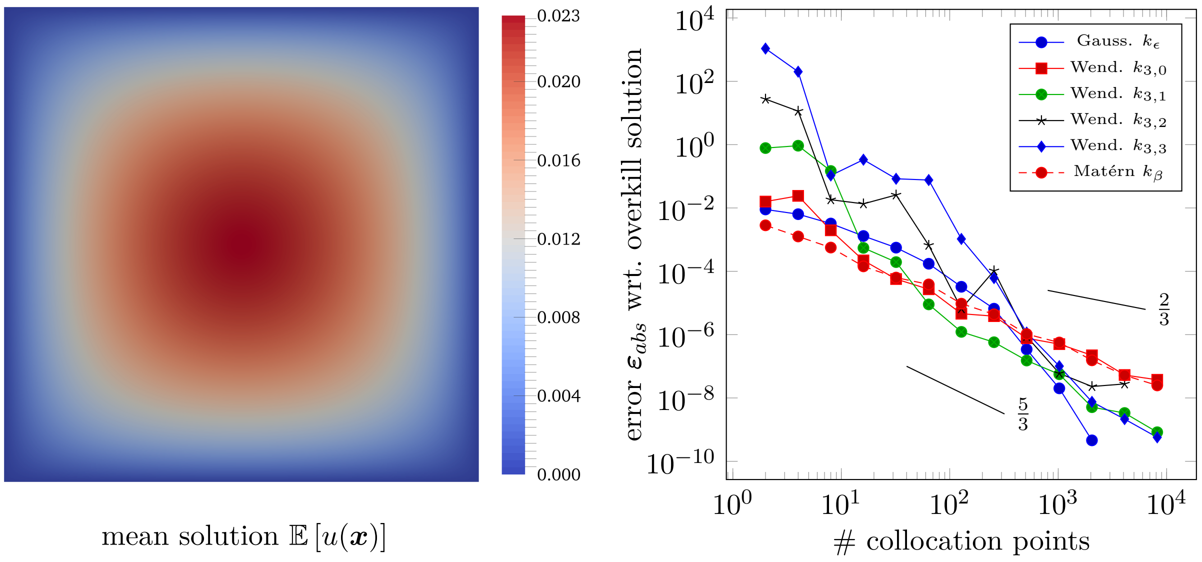
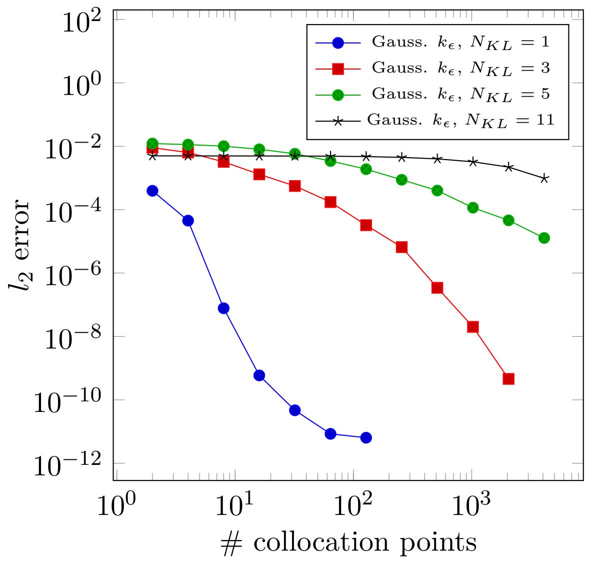


Figure 6.4: Results of the mean approximation of the Karhunen-Loève expansion based two-dimensional random elliptic problem at fixed stochastic dimension $N_{KL} = 3$ with a visualization of the mean solution field (left) and error convergence results for different kernels (right).

Karhunen-Loève expansion-based random diffusion field

The second random-coefficient elliptic problem has a random diffusion field given by a Karhunen-Loève expansion of the initially given infinite-dimensional random field. This problem is introduced in Section 3.2.2. A correlation length of $L_c = \frac{1}{16}$ is used in the following empirical analysis. On the right-hand side of Figure 6.4, results of the convergence analysis for different kernel functions with stochastic dimension $N_{KL} = 3$ are given. The reference solution is the mean solution field computed with the Wendland kernel $k_{3,3}$ for $N_\Gamma = 2^{14}$ collocation points. This solution is visualized on the left-hand side of Figure 6.4. Kernels with algebraic convergence show convergence rates of $\frac{2}{3}$ to $\frac{5}{3}$ after some preasymptotic behavior. The Gaussian kernel starts to show exponential convergence.

Two further convergence studies shall highlight the influence of the dimensionality of the problem (thus the number of Karhunen-Loève terms) on the convergence rates with respect to the mean of the solution field. For stochastic dimensions one and three, the quadrature level is still $l_q = 7$. Dimension five, thus $N_{KL} = 5$ is approximated with $l_q = 5$ and the same problem with $N_{KL} = 11$ is computed with $l_q = 2$, due to runtime and memory restrictions. Reference solutions are the solution for the Wendland kernel $k_{1,2}$ with $N_\Gamma = 2^{14}$ collocation points in dimension $N_{KL} = 1$, the solution for the Gaussian kernel with $N_\Gamma = 2^{14}$ collocation points in dimension $N_{KL} = 5$ and the solution for the Wendland kernel $k_{11,2}$ with $N_\Gamma = 2^{14}$ collocation points in dimension $N_{KL} = 11$. Indeed, different kernels are used to construct a reference solution in different stochastic dimensions. This is due to the fact that those solutions are chosen as reference solution which showed smallest errors on level 14 comparing subsequent approximation levels. In that sense, these overkill solutions converged to a fix-point. However, this criterion to choose a reference solution is not necessarily uniform for growing dimensions.



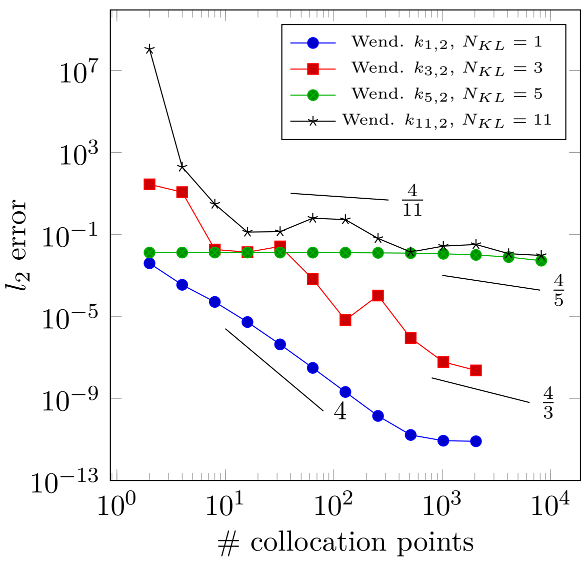


Figure 6.5: Error behavior of the mean of the solution field of the Karhunen-Loève expansion based two-dimensional random elliptic problem for a growing number of Karhunen-Loève terms with kernel-based stochastic collocation using Gaussian kernels (left) and Wendland kernels (right).

On the left-hand side of Figure 6.5, the convergence behavior of the Gaussian kernel for increasing stochastic dimension is shown. Exponential convergence is always achieved, with a longer preasymptotic regime in higher dimensions. This is a well-known behavior. Furthermore, we observe the necessary convergence stagnation due to regularization. The right-hand side plot of Figure 6.5 outlines the same study for the Wendland kernel $k_{N_{KL},2}$. All convergence graphs seem to follow asymptotically rates of roughly $\frac{4}{N_{KL}}$ with potentially preasymptotic results for $N_{KL} = 5, 11$. In the one-dimensional case, regularization again leads to convergence stagnation.

Figure 6.6 gives results for the approximation of the second moment of the random-coefficient elliptic problem with $N_{KL} = 3$. Tensor-product quadrature of level $l_q = 7$ is used in all cases. The reference solution is based on an approximation by Wendland kernels $k_{3,0}$ and $N_\Gamma = 2^8$ collocation points, cf. Figure 6.6 on the left-hand side. Similar convergence rates as in the mean approximation in three stochastic dimensions are achieved.

6.1.4 Random two-phase incompressible Navier-Stokes equations

The following application problems require the solution of the (two-phase) incompressible Navier-Stokes equations, which is a computational intensive task. Furthermore, due to missing solution theory for the strong formulation of these equations in three dimensions, theoretical results on the smoothness in stochastic space are not available.

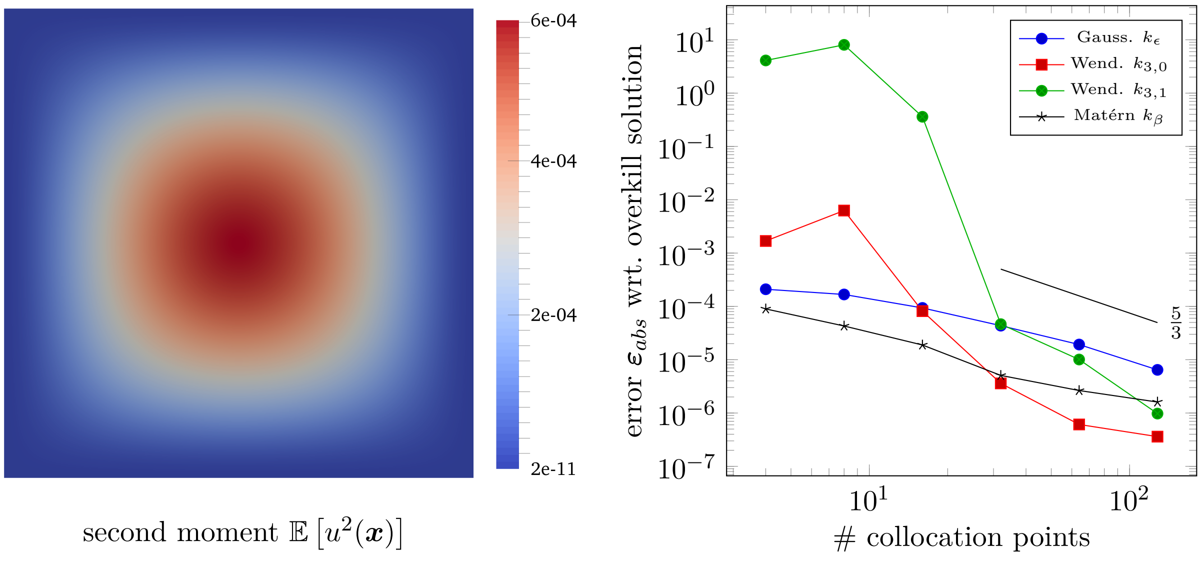


Figure 6.6: Second moment analysis results for the Karhunen-Loève based random elliptic problem at $N_{KL} = 3$ with a visualization of the second moment of the solution field (left) and convergence results for different kernel functions (right).

Flow over a backward-facing step

In the first flow problem, a classical backward-facing step simulation is considered, cf. Section 3.3.3. It is well-known that a vortex forms behind such as step. The approximation of the mean velocity field and the second moment of the velocity field at time $t = 10.0$ seconds is considered. Note that the mean of the so-called reattachment length, cf. Section 3.3.3, will be approximated in Section 6.2.6. All simulation results discussed here, have a space discretization of $\mathbb{N}_{\mathcal{D}} = 50 \times 40 \times 3$ grid points. Discretization in time is done with the second-order Adams-Bashforth method and adaptive time stepping, cf. Section 5.1.3.

Remember that we defined random input fields $u_{\Gamma_{in}}(\boldsymbol{y})$, $\mu_1(\boldsymbol{y})$ and $\rho_1(\boldsymbol{y})$ in Section 3.3.3, with $\boldsymbol{y} \in \Gamma = [-\sqrt{3}, \sqrt{3}]^3$. Stochastic approximation is now done with respect to the image of these input fields, thus the stochastic variable $\boldsymbol{y}' \in \Gamma' = [0.1, 1] \times [0.001, 1] \times [500, 1000]$ with

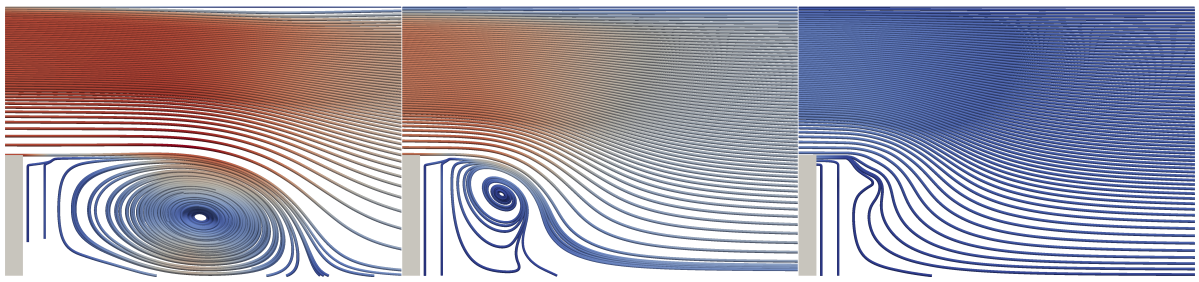


Figure 6.7: Streamline visualization of solution realizations of the backward facing step problem with stochastic parameters $\boldsymbol{y}'_{15} \approx (0.94, 0.26, 560)^\top$, $\boldsymbol{y}'_{20} \approx (0.24, 0.74, 580)^\top$ and $\boldsymbol{y}'_{32} \approx (0.11, 0.79, 724)^\top$ (from left to right).

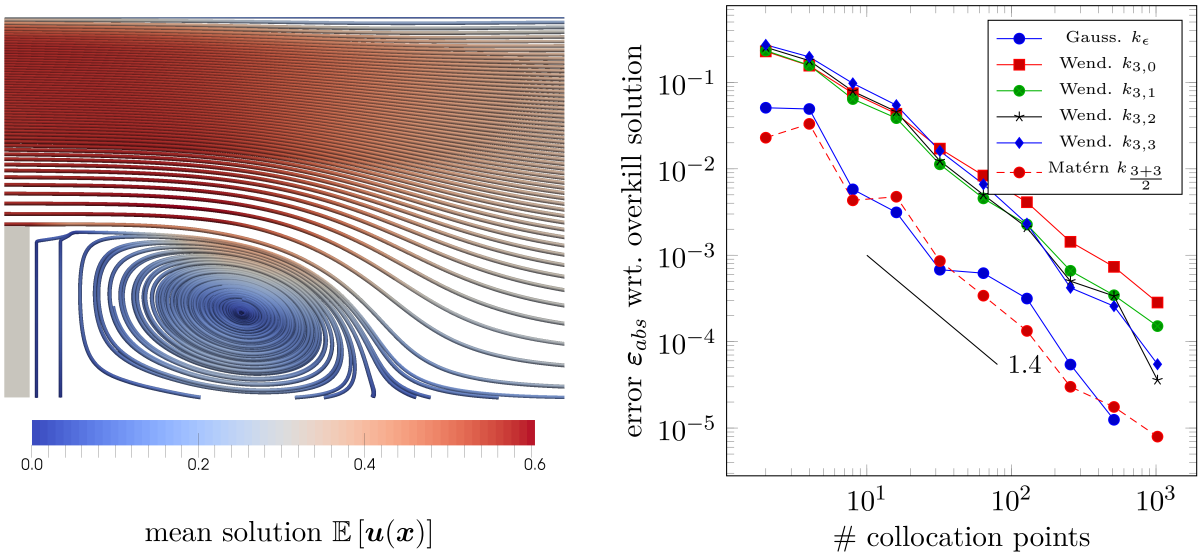


Figure 6.8: Results of the backward facing step model problem with a streamline visualization of a subset of the mean velocity field colored by the velocity magnitude (left) and error convergence results in the first component of the mean velocity field (right).

$\boldsymbol{y}' = (u_{\Gamma_{in}}, \mu_1, \rho_1)^\top$. To get a uniform approximation, we employ

$$\|\boldsymbol{y}'\| := \left\| \left(u_{\Gamma_{in}}, \mu_1, 10^{-3}\rho_1\right)^\top \right\|_2$$

in the radial basis function construction. It maps the stochastic variable to the unit cube. All other parameters in stochastic collocation remain as introduced in Section 6.1.1. Examples of three flow field realizations at $t = 10.0$ seconds are given in Figure 6.7.

Figure 6.8 shows results of the mean approximation of the velocity field. The reference solution is computed by a Gaussian kernel and $N_{\Gamma'} = 2^{10} = 1024$ Halton collocation points. A streamline visualization of the resulting mean solution field is given on the left-hand side of Figure 6.8. The right-hand side shows error convergence results in the first component of the velocity field for different kernel functions and a growing number of collocation points. While Wendland kernels of order zero and one achieve smaller convergence rates, all other kernels give rise to an almost identical (measured) convergence rate of 1.4. This fixed maximum convergence rate suggests a finite smoothness in stochastic space. Overall, Matérn and Gaussian kernels have a smaller preasymptotic error, such that an approximation error of less than $\boldsymbol{\varepsilon}_{abs} = 10^{-3}$ is possible by solving 32 flow problems.

Results for the second moment analysis of the velocity field are given in Figure 6.9. The reference solution is computed with the Matérn kernel $k_{\frac{3+3}{2}}$ and $N_{\Gamma'} = 1024$ collocation points. This solution is shown on the left-hand side of Figure 6.9. The right-hand side of the same figure gives error convergence results. Measured error convergence rates are in the range of 1.5 for the Matérn kernel and higher-order Wendland kernels. The Gaussian kernel runs into conditioning issues beyond 32 collocation points.

Summarizing the results of this study, we get a convergence rate between 1.4 and 1.5 to

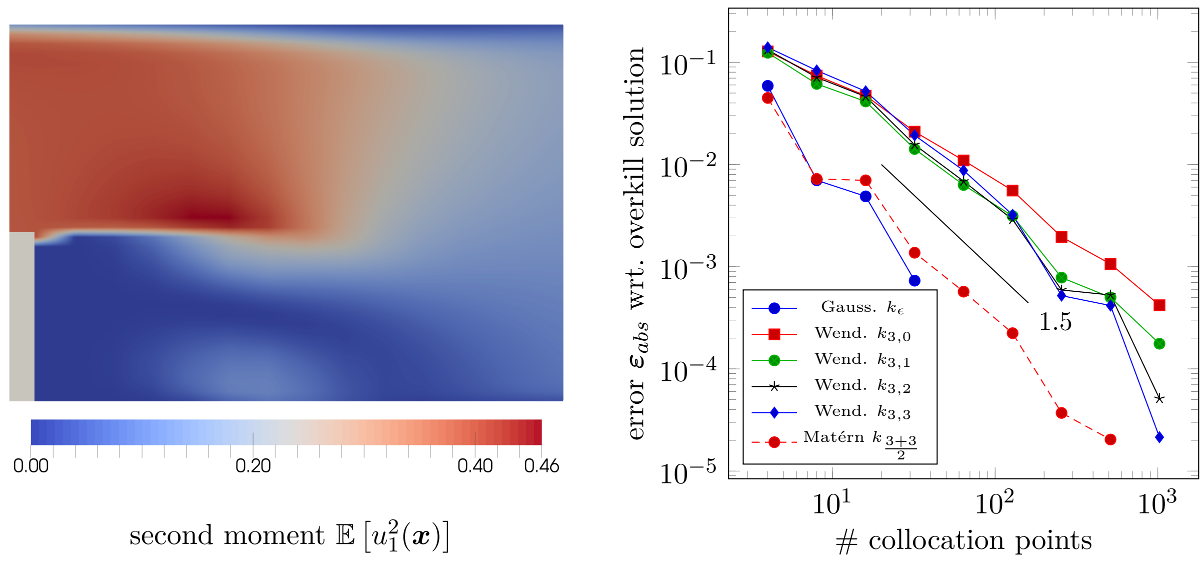


Figure 6.9: Second moment analysis results for the backward facing step model problem with a visualization of a subset of the second moment of the first velocity component colored by the magnitude (left) and error convergence results in the first component of the second moment velocity field (right).

approximate the mean and the second moment of the velocity field of this problem. This rate clearly outperforms the robust Monte-Carlo and current Quasi-Monte-Carlo methods. A comparison to the Polynomial Chaos Expansion (PCE) [GS91] and (sparse) spectral tensor-product approximations [NTW08b, BNT10] will be given in Section 6.2.

Performance results To approximate mean or second moment flow fields in the backward facing step problem, approximately 50 GB of data are generated (including intermediate time steps). The average runtime to compute each PDE realization is about 5.33 minutes. Figure 6.10 outlines on the left-hand side the time required to approximate the mean velocity field up to a fixed error tolerance of $\varepsilon_{abs} < 10^{-3}$. The given timings include the time to compute the PDE solutions on a 32 GPU cluster, with assumed optimal scaling due to the independence of each flow realization. Furthermore, runtime measurements of the stochastic collocation method are included. Approximation by Gaussian and Matérn kernels is by far the fastest approach here, with a total runtime of about 5.5 minutes on a 32 GPU cluster. Due to a higher preasymptotic error, approximation by the the other kernel functions (with fixed error tolerance), is orders of magnitude slower. The right-hand side of Figure 6.10 further gives an indicator on the composition of the full approximation runtime. Note that this diagram has a *logarithmic* vertical axis. It becomes obvious that the runtime of the stochastic collocation method is only a small fraction of the total runtime. Between 7 and 13 seconds are measured in the three shown examples. These timings do not show a clear dependence on the number of collocation points, since timings are largely dominated by file read and write operations with fluctuating response time on an actively used GPU cluster.

Overall, the achieved approximation error of less than 10^{-3} within only 5.5 minutes on a

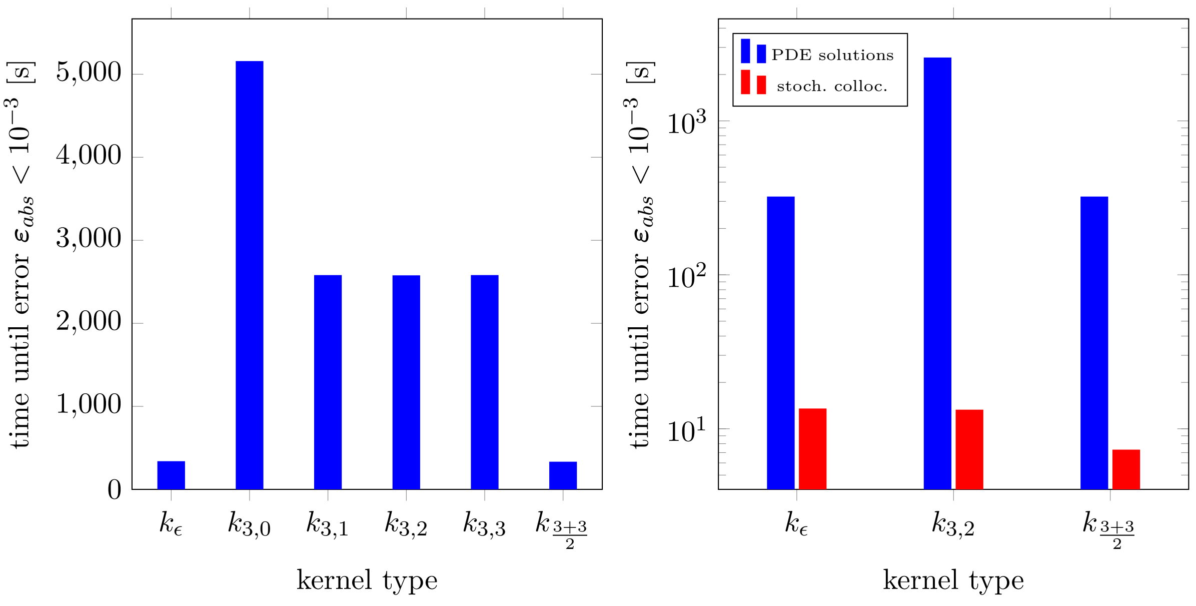


Figure 6.10: Runtime results on 32 GPUs to approximate the mean velocity field of the backward facing step problem at $t = 10$ seconds up to an error threshold of $\varepsilon_{abs} < 10^{-3}$, including PDE solution time for all kernels (left) and with a decomposition on the different algorithmic parts for several kernels and *logarithmic* scale (right).

32 GPU cluster is an optimal result, showing that uncertainty quantification for smaller flow problems can now be done in an almost interactive way.

Bubble flow with random volume force modeled by Karhunen-Loève expansion

The second flow problem is a two-phase flow problem, which is discretized in space by a mesh of $N_\mathcal{D} = 100^3$ grid points. Time discretization is done with a second-order Adams-Bashforth method and adaptive time step size control. The problem is discussed in Section 3.3.3, in detail. A stochastic influence is introduced by a random volume force in which the infinite-dimensional stochastic parameter is approximated by a truncated Karhunen-Loève expansion of $N_{KL} = 3$ terms. The correlation length is set to $L_c = 2.0$. Figure 6.11 shows visualizations of four solution realizations at $t = 0.2$ seconds. The bubble is shown by extracting the iso-surface of the level-set function for a value of zero. Furthermore, a slice of the velocity field with coloring by the magnitude and velocity field streamlines are visualized.

We first consider the mean approximation. Quadrature follows the default of tensor-product quadrature with an approximation level of $l_q = 7$. The norm in the radial basis function construction is $\|\cdot\| := \sigma\|\cdot\|_2$, with $\sigma = 0.1$ for the Gaussian kernel and all Wendland kernels and $\sigma = 1.0$ for the Matérn kernel. The approximation of the mean velocity field by the Gaussian kernel with $N_\Gamma = 1024$ collocation points is considered as reference solution. Its visualization by a two-dimensional slice of streamlines is given in Figure 6.12 on the left-hand side. The right-hand side diagram outlines error convergence results for different kernel functions with respect to the reference solution. Measured convergence rates in the range of 0.75 are achieved

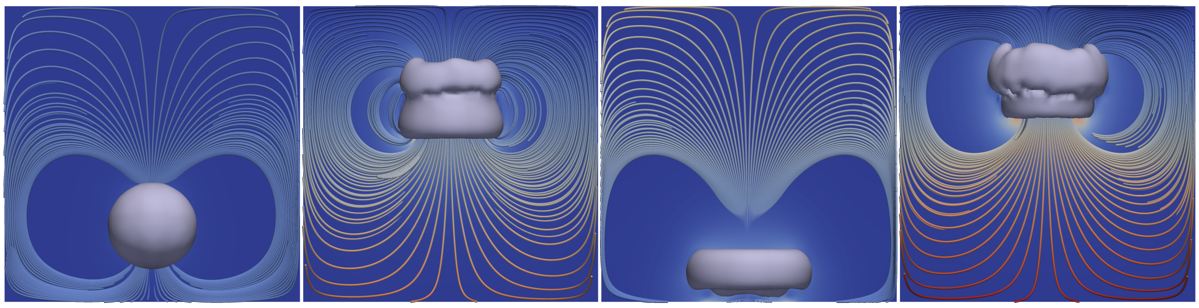


Figure 6.11: Visualizations of flow field solutions of the bubble flow with random volume force and $N_{KL} = 3$ at $t = 0.2$ seconds for stochastic parameters $\boldsymbol{y}_1 \approx (0.0, -0.58, -1.04)^\top$, $\boldsymbol{y}_{12} \approx (-1.08, -1.22, -0.07)^\top$, $\boldsymbol{y}_{19} \approx (0.97, -0.32, 1.45)^\top$ and $\boldsymbol{y}_{411} \approx (1.21, -0.93, -0.72)^\top$ (from left to right).

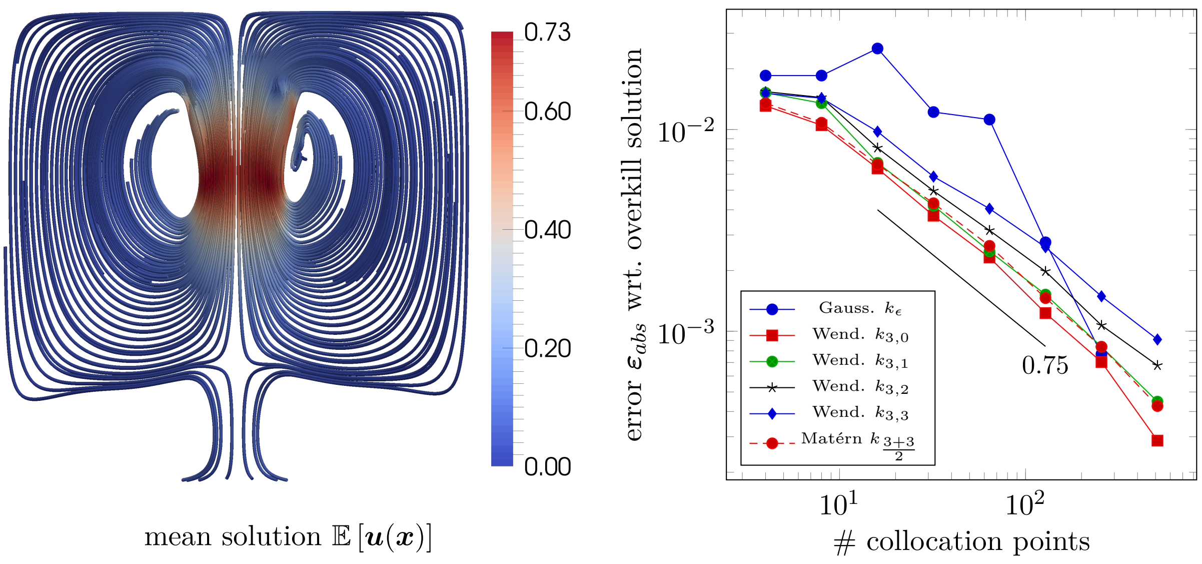


Figure 6.12: Streamline slice visualization of the mean velocity field with color-coded velocity magnitude in the bubble flow problem with random volume forces (left) and error convergence results for the first component of the same mean velocity field (right).

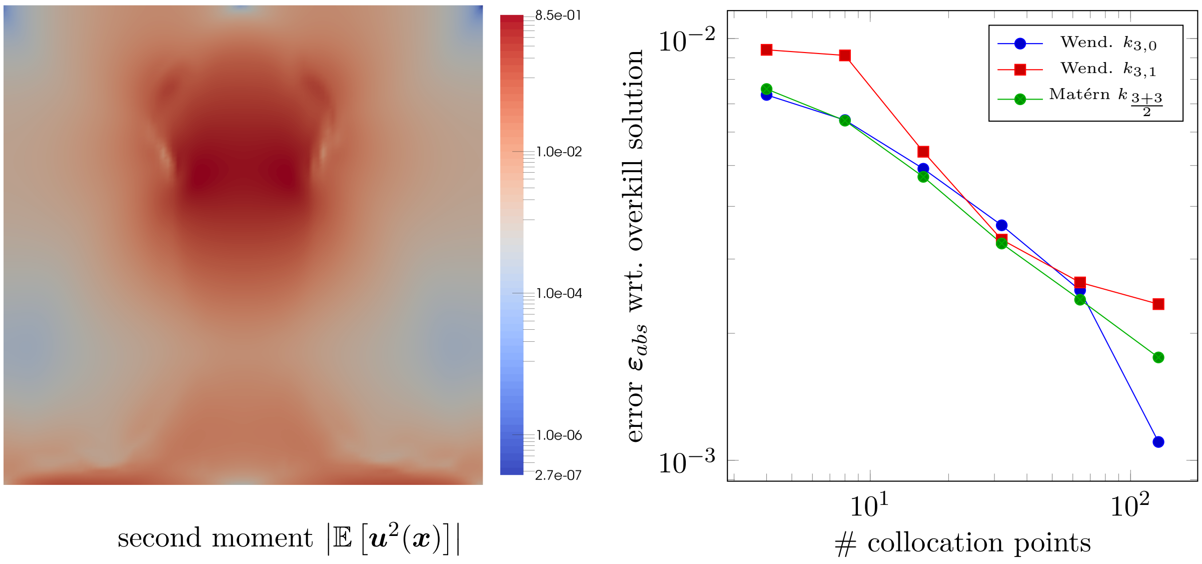


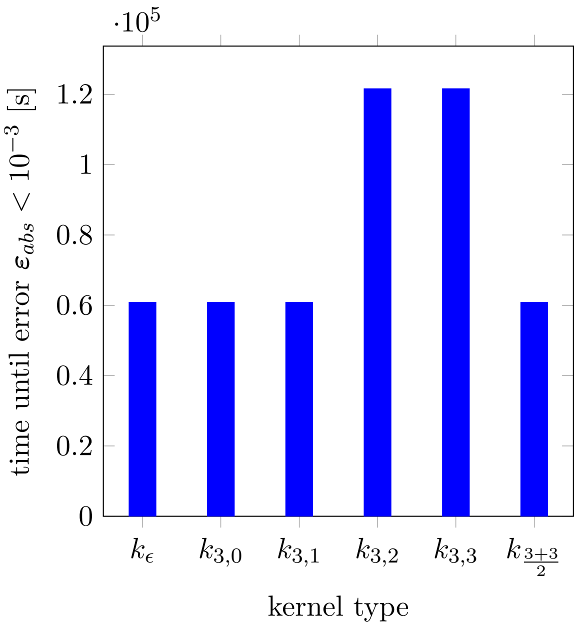
Figure 6.13: Results of the second moment analysis of the bubble flow problem with random volume forces with a logarithmic color-coded slice visualization of the second moment velocity field magnitude (left) and error convergence results for the first component of the full second moment velocity field (right).

by Wendland and Matérn kernels. There is a slight reduction in convergence rate for higher smoothness Wendland kernels. A potential explanation is a lower robustness with respect to numerical outliers in the PDE approximation for higher smoothness kernels. Convergence results, obtained by the Gaussian kernel, tend towards higher-order or even exponential convergence. This might suggest a very smooth dependence of the solution field on the random input.

Approximation of the second moment is also studied. Here, the solution by the Wendland kernel $k_{3,0}$ with $N_\Gamma = 256$ is taken as reference solution. All other parameters remain identical to the mean approximation. A visualization of the magnitude of the second moment velocity field by a colored two-dimensional slice is given on the left-hand side of Figure 6.13. Note that the coloring is chosen with a logarithmic scale, such that details of the second moment field become visible. An empirical error convergence study is given on the right-hand side of the same figure. Results computed with Wendland kernels of order zero and one and the Matérn kernel are shown. The empirical convergence rate is in the range of 0.5. However, this seems to be a preasymptotic result. Furthermore, results of Gaussian and higher-order Wendland kernels are not shown due to a clearly preasymptotic behavior.

Overall, the given rates for mean approximation clearly outperform Monte Carlo methods. A convergence comparison to PCE or (sparse) spectral tensor-product methods is given in Section 6.2.7.

Performance results To approximate the mean of the velocity field by $N_\Gamma = 512$ collocation points at time $t = 0.2$ seconds, a total of about of *one terabyte* of simulation data is generated. These numbers are based on the assumption that a simulation time slice is written each 0.005 seconds of simulated time. The average runtime to compute a single flow realization on one



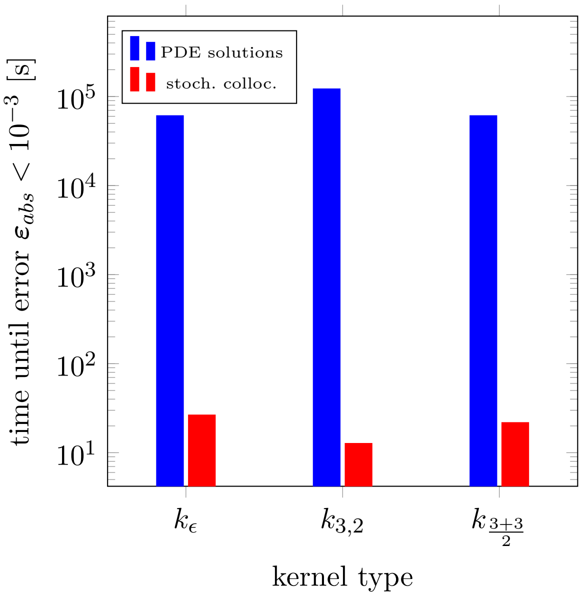


Figure 6.14: Runtime results on 32 GPUs to approximate the mean velocity field of the bubble flow with random volume force at $t = 0.2$ seconds up to an error threshold of $\varepsilon_{abs} < 10^{-3}$ including PDE solution time for all kernels (left) and with a decomposition on the different algorithmic parts for several kernels and *logarithmic* scale (right).

GPU is 2.11 hours. Remember, that this is about three times faster than approximation on equally priced CPUs. Based on these numbers and additional measurements of the time to solve the stochastic collocation problem, we can give idealized measurements for the time that is necessary to get the mean approximation error below a threshold of $\varepsilon_{abs} = 10^{-3}$. Figure 6.14 summarizes these results with the assumption that all independent flow simulations can be run in parallel on a 32 GPU cluster. It turns out that the Gaussian kernel, the Matérn kernel and lower-order Wendland kernels achieve the best possible runtime of about 16.88 hours on a 32 GPU cluster. The right-hand side diagram in Figure 6.14 clearly indicates that the stochastic space approximation needs only a very small fraction of the full computational runtime with about 20 seconds. Overall, it is possible to achieve mean approximations with an error of $\varepsilon_{abs} = 10^{-3}$ within much less than a day of compute time. This is a profound result for such a large-scale problem, which is clearly improved by the GPU parallelization of the flow solver.

Stochastic homogenization for rising bubbles

The last large-scale flow problem is stochastic homogenization for rising bubbles, cf. Section 3.3.3. It is discretized by a $N_{\mathcal{D}} = 100^3$ grid point finite-difference discretization in space and by a second-order Adams-Bashforth method with adaptive time-stepping in time. A total of $N_{FN} = 5$ stochastic dimensions are considered, which include a random three-dimensional initial bubble position random viscosity and density in the liquid phase. Randomness in all quantities is introduced by Karhunen-Loève expansions, truncated after the first term.

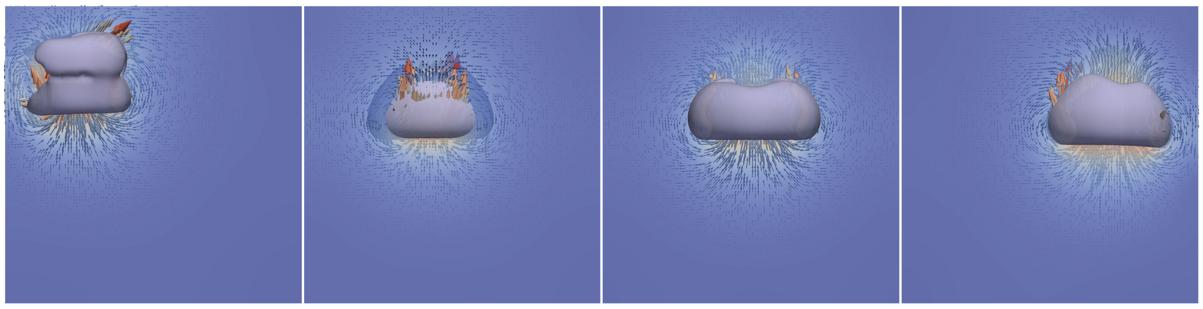


Figure 6.15: Flow field and bubble visualization of solution realizations of the bubble homogenization problem with stochastic parameters $\boldsymbol{y}'_8 \approx (0.05, 0.07, 0.12, 0.02, 863.64)^\top$, $\boldsymbol{y}'_6 \approx (0.09, 0.05, 0.07, 0.09, 772.73)^\top$, $\boldsymbol{y}'_{33} \approx (0.1, 0.05, 0.12, 0.08, 512.4)^\top$ and $\boldsymbol{y}'_{27} \approx (0.14, 0.05, 0.09, 0.09, 735.54)^\top$ (from left to right).

For technical reasons, we again apply the stochastic approximation in the image of the five random input fields, thus a stochastic space $\Gamma' = [0.04, 0.16] \times [0.05, 0.07] \times [0.04, 0.16] \times [0.001, 0.1] \times [500, 1000]$ is considered. To approximate that space uniformly, the norm

$$\|\boldsymbol{y}'\| := \sigma \left\|(5x_1^{init}, 5x_2^{init}, 5x_3^{init}, \mu_1, 10^3\rho_1)\right\|_2$$

is used in the construction of the radial basis functions. Approximation by Gaussian kernels is done with $\sigma = 0.1$, while all other kernels have $\sigma = 1.0$. All other approximation parameters are set as in Section 6.1.1. Four flow field realizations at $t = 0.2\,s$ are shown in Figure 6.15.

Approximation results for the mean of the velocity field at $t = 0.2\,s$ are given in Figure 6.16. The left-hand side image shows a streamline slice visualization of the reference solution, which is approximated by the Gaussian kernel with $N_{\Gamma'} = 512$ collocation points. On the right-hand side diagram, error convergence in the first component of the velocity field with respect to different kernel functions is given. Lower-order Wendland kernels have a measured convergence rate of about 0.7, while the use of higher-order Wendland kernels leads to empirical convergence rates of about 0.6. The highest convergence rates are achieved by the Gaussian and the Matérn kernel with about 0.8. It is interesting to see that increased smoothness of the Wendland kernel gives decreased convergence rates. This might be related to noise in the solution realizations due to numerical errors in the approximation of the two-phase Navier-Stokes equations. In that sense, lower-order methods might be more robust.

The same numerical experiment is carried out for the second moment of the solution velocity field. The reference solution approximated by the Wendland kernel $k_{5,0}$ with $N_{\Gamma'} = 256$ collocation points is given in Figure 6.17 on the left-hand side. Here, the magnitude of the second moment velocity field is visualized in a two-dimensional slice with logarithmic color mapping. The error convergence diagram, on the right-hand side of the same figure, shows results for the first component of the second moment velocity field. A similar convergence behavior as in the mean case is achieved. The Matérn kernel gives the highest convergence rate of about 0.8. Lower-smoothness Wendland kernels lead to convergence rates of about 0.6. Rates are further decreased for the higher-order Wendland kernels. Convergence results for the Gaussian kernel are still in the preasymptotic range. They are skipped here.

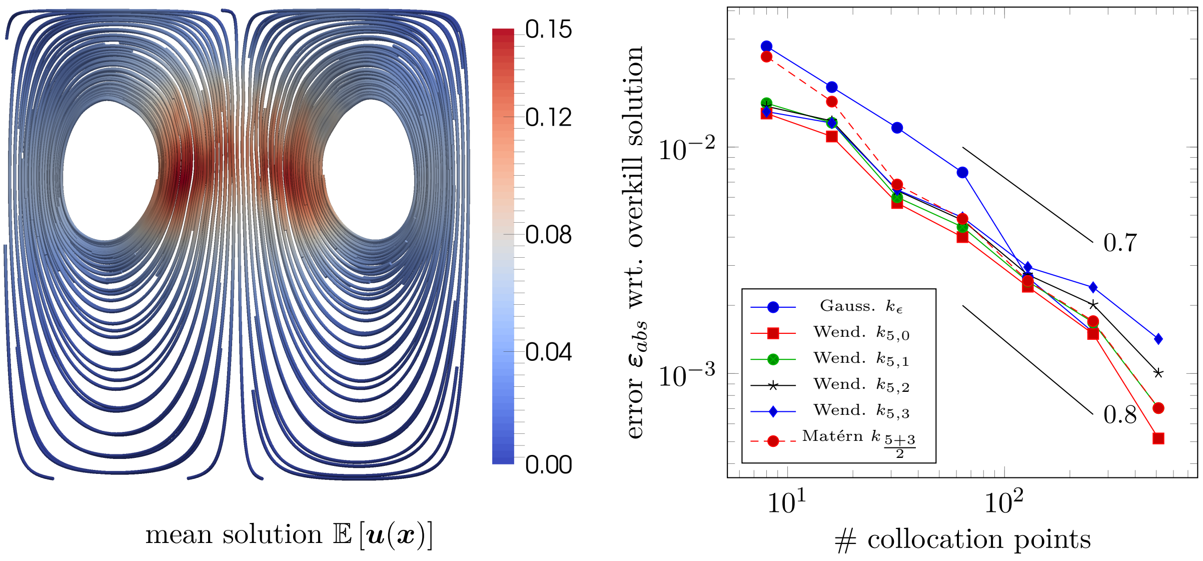


Figure 6.16: Results of the bubble flow stochastic homogenization problem with a streamline slice visualization of the mean velocity field and color-coded velocity magnitude (left) and error convergence results for the first component of the mean velocity field (right).

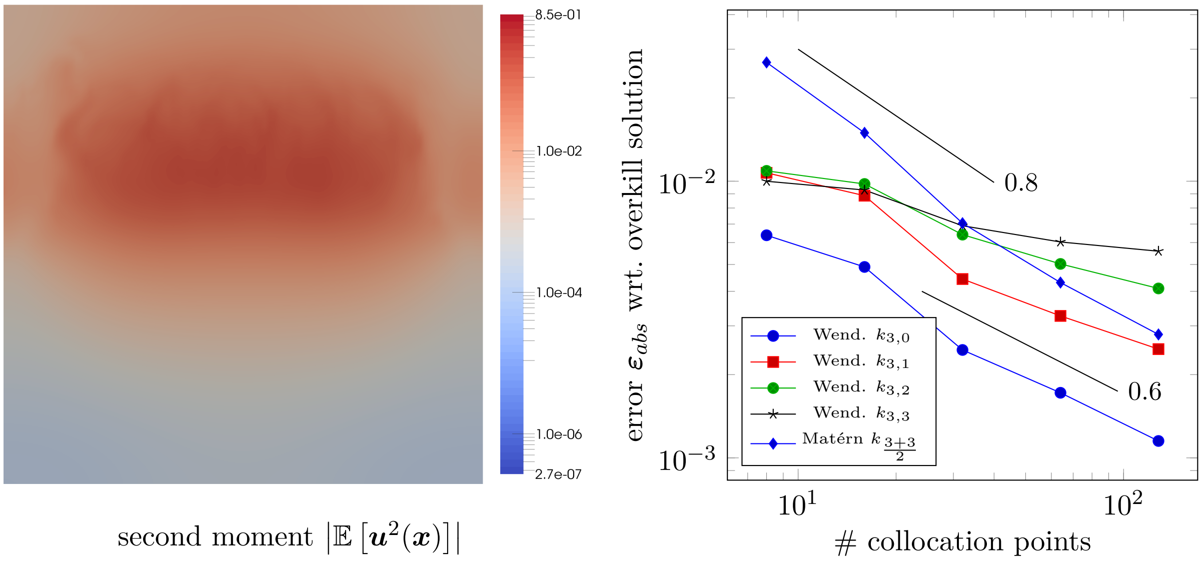
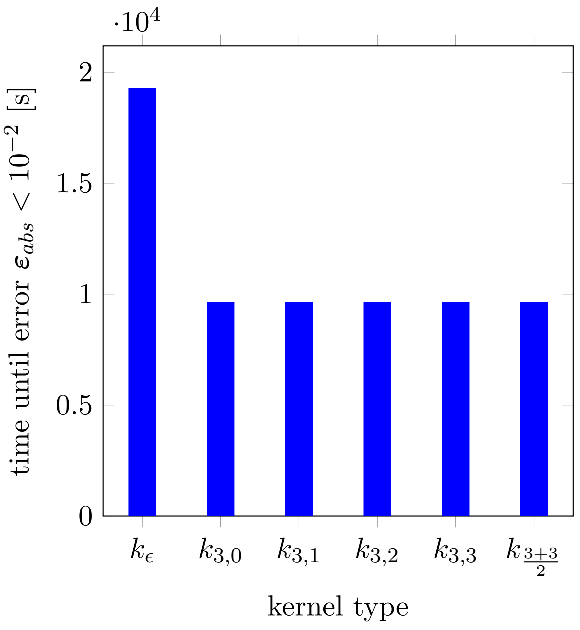


Figure 6.17: Logarithmic color-coded slice visualization of the second moment velocity field magnitude (left) and error convergence results for the first component of the full second moment velocity field (right) in the bubble flow homogenization test case.



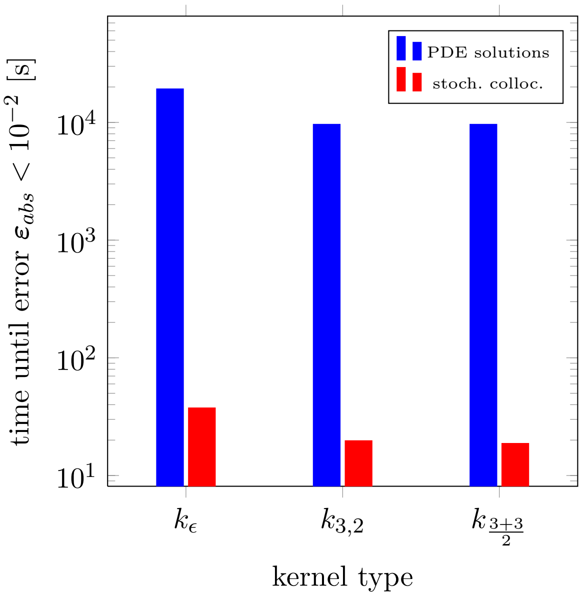


Figure 6.18: Runtime results on 32 GPUs to approximate the mean velocity field of the homogenized bubble flow at $t = 0.2$ seconds up to an error threshold of $\varepsilon_{abs} < 10^{-2}$ including PDE solution time for all kernels (left) and with a decomposition on the different algorithmic parts for several kernels and *logarithmic* scale (right).

To summarize, we observe convergence rates of up to 0.8 for a large-scale stochastic collocation problem with a five-dimensional stochastic space with potential low smoothness. Note furthermore that noise due to numerical errors in the domain space might be available. This is a profound result clearly outperforming Monte Carlo methods.

Performance results We conclude this section, by giving performance results for the application problem of bubble homogenization. To compute up to 512 flow field solutions with solution snapshots every 0.005 seconds of simulated time, about *one terabyte* of simulation data is generated. On a single GPU, the average computation time to compute the flow fields takes about 2.67 hours, due to the efficient GPU parallelization of the flow solver. This leads to idealized total runtimes on 32 GPUs, which are given in Figure 6.18. On the left-hand side, the total runtime to achieve a mean approximation error of at least $\varepsilon_{abs} \leq 10^{-2}$ is given. Here, the averaged runtime of each flow field realization approximation and measured compute times of the stochastic collocatiom method are taken into account. Approximation by the Matérn kernel and all Wendland kernels leads to the lowest runtime of about 2.675 hours on 32 GPUs. The right-hand side of Figure 6.18 compares the runtime of the flow solver to the runtime of the stochastic collocation method in a logarithmically scaled diagram. The stochastic collocation method that runs here on a single GPU, takes only about 20 to 30 seconds, which is a neglectable fraction of the total solution time. Overall, it is possible to approximate the mean solution field of this highly complex application problem, up to two valid digits, in less than three hours on 32 GPUs.

6.1.5 Conclusions

The non-intrusive radial basis function kernel-based stochastic collocation method allows to approximate stochastic moments of random PDEs. Up to exponential convergence order is possible for appropriately smooth problems. For the discussed large-scale flow problems, at least algebraic convergence rates are possible, which clearly outperform Monte Carlo methods. Mean approximation of computational intensive two-phase problems with up to a few valid digits is possible at total runtimes between a few hours and less than a day. This is due to the GPU-based high code performance and low preasymtotic errors of the collocation method. By that way, uncertainty quantification becomes practical for real-world applications with strong time limitations.

6.2 Comparison to (sparse) spectral tensor product approximations

In this section, the proposed kernel-based stochastic collocation method is compared to standard full tensor-product and sparse grid approximations. Stochastic collocation with tensorized global Lagrange polynomials and the (generalized) Polynomial Chaos Expansion (PCE) are considered. Due to prohibitive overall computational runtimes, only a subset of the model and application problems from Chapter 3 are discussed.

6.2.1 Benchmark setup

The numerical studies for the RBF kernel-based stochastic collocation method are evaluated using the GPU-based numerical software from Section 4.5. Discretization in space is identical to Section 6.1

RBF-based stochastic collocation Gaussian kernels with scaling parameter $\epsilon = 1$, Wendland kernels of second order and/or Matérn kernels with $\beta = \frac{N_{KL}+3}{2}$ are applied as standard. The norm in the radial basis function construction is $\|\cdot\| := \sigma\|\cdot\|_2$ with $\sigma = 1$, if not stated otherwise. Regularization is usually set to $\epsilon_{reg} = 10^{-12}$. The default quadrature level is $l_q = 7$ with tensor-product Clenshaw-Curtis quadrature. Collocation points are sampled from a Halton sequence of appropriate dimension.

Stochastic approximation by Dakota Approximations by the kernel-based method shall be compared to results from existing numerical techniques in uncertainty quantification. To this end, model and application problems are solved with the kernel-based method and with the *Dakota* framework [ABB$^+$09a]. Dakota is a parallel software framework developed by the Sandia National Laboratories. It allows e.g. to apply optimization and uncertainty quantification for black-box solvers. Numerical techniques for uncertainty quantification in Dakota, which are of main interest here, are (generalized) Polynomial Chaos Expansion and tensor-product stochastic collocation methods. Dakota features both methods with full tensor product constructions and sparse (grid) constructions, where the latter ones are known to overcome the curse of dimensionality for important model problems.

An introduction to PCE is given in [GS91]. Following [ABB$^+$09a], Dakota implements generalized PCE using the Askey scheme [XK02]. Approximations are made using Legendre or-

Listing 6.1 The part of the Dakota input file, which configures the PCE method.

```
method
  polynomial_chaos
    quadrature_order = <order>
#   sparse_grid_level = <level>
    dimension_preference = <N_KL>*1
    samples = 10000 seed = 12347 rng rnum2
    output silent
```

Listing 6.2 Configuration for the stochastic collocation method in Dakota.

```
method
  stoch_collocation
    quadrature_order = <order>
#   sparse_grid_level = <level>
    dimension_preference = <N_KL>*1
    samples = 10000 seed = 12347 rng rnum2
    output silent
```

thogonal polynomials, since we consider uniform random input variables. To be concise, the interested reader is referred to the technical documentation [ABB+09b] for details of the applied method. We summarize the important numerical methods-related part of the applied Dakota configuration file in Listing 6.1. In case of full tensor product approximations, `<order>` is replaced by the target approximation order for the univariate polynomials. Alternatively, the key word `sparse_grid_level` indicates the use of sparse approximations with `<level>` being replaced by the sparse grid level. Moreover, `<N_KL>` is filled with the dimension in stochastic space.

Classical stochastic collocation uses univariate Lagrange polynomials as Lagrange basis functions [NTW08b, BNT10]. These are collocated in appropriate Gauss points and perform interpolation for function values given in these collocation points. For multi-variate interpolation, thus higher dimensions in stochastic space, either a full tensor product of the univariate Lagrange polynomials or a Smolyak sparse grid construction [Smo63] is used. More details are given in [ABB+09a, ABB+09b]. The standard parameters for stochastic collocation benchmarks are given in Listing 6.2. It turns out that both sets of parameters lead, for fixed level or order and sparsity, to the same numerical results, cf. [ABB+09a, Section 5.4]. Nevertheless, both approximations are computed for the less computational expensive problems, while only (sparse grid) stochastic collocation is used for problems of high computational intensity.

In both approaches, classical stochastic collocation and PCE, it is further possible to define weights for the different stochastic dimensions by `dimension_preference`, leading to weighted approximation spaces, thus e.g. anisotropic sparse grids [NTW08a]. Note that this option is not used here, because the RBF kernel-based stochastic collocation is also discussed without any directional preference. However, the topic of dimension-wise weighting will be outlined in Chapter 10 for kernel-based methods.

Approximation of the mean of quantities of interest A limitation of the Dakota framework with loosely coupled black-box PDE solvers is the lack of a mechanism to address full (discretized) solution fields as quantities of interest in an efficient way. Therefore, in contrast to the previous section, the subsequent convergence studies compare kernel-based results and Dakota-based results only using a single-valued quantity of interest. To be more specific, in those cases, where the only quantity of interest defined in Chapter 3 is a field, e.g. $\boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x})$, we move over to the integral over this quantity, which is replaced by the Monte-Carlo estimator using the discretization points, thus

$$\pi(\boldsymbol{u}(\boldsymbol{y}, \cdot)) := \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} \boldsymbol{u}(\cdot, \boldsymbol{x}_i) \,.$$

As long as a constant number of grid points is used in the convergence studies, this construction is expected to have a limited influence on the error behavior, cf. Section 6.3.1 for an in-depth analysis of related approximations.

Implementation In case of full random PDE problems without analytic solution, the corresponding PDE solvers for elliptic and two-phase Navier-Stokes problems are implemented on GPUs with discretization parameters as discussed in the last section, cf. Section 6.1.1. This also holds for the Dakota benchmarks, in which the GPU codes are called from the CPU-based Dakota control program. Quantities of interest in the flow examples are extracted using the Python interface of Paraview, which might introduce additional approximation errors.

Error analysis structure The structure of the convergence studies will be similar to the previous section. Comparisons between kernel-based results and Dakota-based results are presented for the first stochastic moment. Reference solutions will be defined separately for each model and application problem. Either exact or so-called *overkill* reference solutions will be used. We compute relative errors

$$\varepsilon_{rel} := \frac{|v_{sol} - v_{ref}|}{|v_{ref}|} \,,$$

with v_{sol} the approximate solution and v_{ref} the reference solution. This is trivially possible, since the quantity of interest is always a single number. Convergence plots are given with respect to the number of collocation points, because this is the only relevant quantity in most real-world applications in which each PDE solve is in the range of hours of compute time.

6.2.2 One-dimensional random-coefficient elliptic problem

This model problem is an elliptic random-coefficient problem with one-dimensional stochastic space. As introduced in Section 3.1.1, it has a well-known analytic solution.

Figure 6.19 presents the comparison between kernel-based stochastic collocation and PCE or stochastic collocation in Dakota. The Gaussian kernel is used with regularization parameter $\epsilon_{reg} = 10^{-15}$, while the Wendland kernel $k_{1,2}$ uses the standard of $\epsilon_{reg} = 10^{-12}$. Since there is just one stochastic dimension, only full tensor-product (TP) constructions are considered. As motivated in the benchmark setup description, the quantity of interest $\pi(u(\boldsymbol{y}, \cdot))$ is taken.

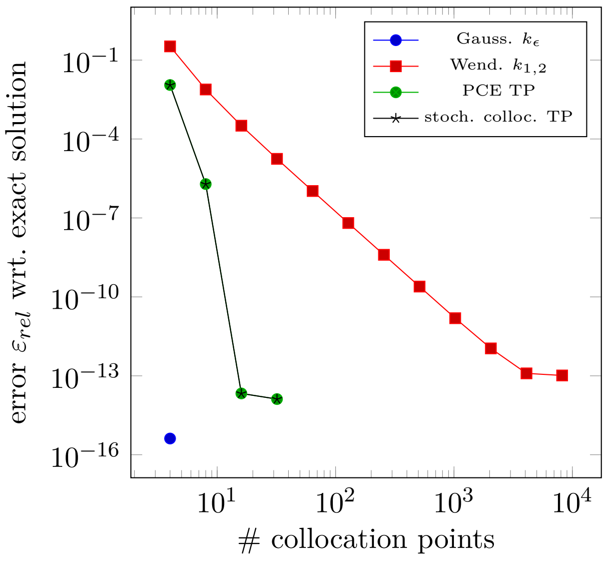


Figure 6.19: Comparison between kernel-based and tensor product-based approximations for the mean of the defined quantity of interest with respect to the random-coefficient Poisson problem with one-dimensional stochastic space.

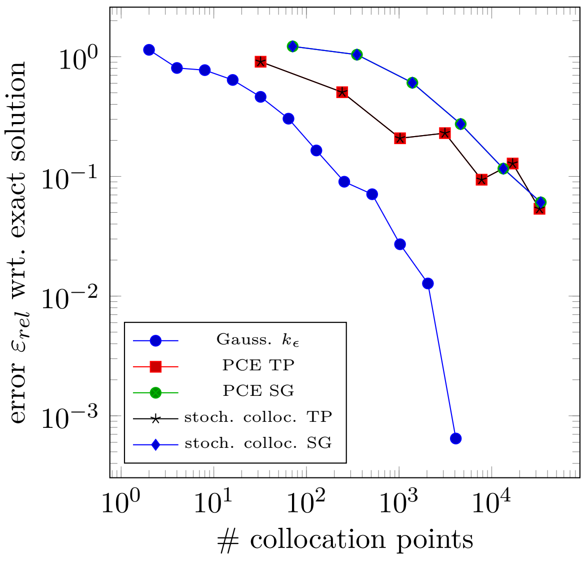
Due to linearity of the mean operator and the linear quantity of interest operator, the exact solution field can be used as reference solution with $u_{ref} := \pi(\mathbb{E}[u_{exact}](\boldsymbol{x}))$. The analysis shows that both Dakota-based tensor-product methods show exponential convergence in this simple model problem. Note that both methods produce exactly the same solutions, here, leading two identical graphs in the figure. These results overcome the algebraic convergence rate of the Wendland kernel. However, the Gaussian kernel clearly outperforms PCE and tensor-product stochastic collocation. While it has the same exponential convergence, it converges with only four collocation points up to machine precision. This is an impressive result, underlining the excellent asymptotic and pre-asymptotic properties of the method.

6.2.3 g function

This study with variable stochastic dimension and analytic solution follows Section 3.1.2. Note that this problem does not fulfill the smoothness requirements of e.g. Smolyak sparse grid constructions [Smo63]. It therefore is representative for problems with limited smoothness.

Comparisons between the kernel-based method and (sparse) tensor-product based methods are given in Figure 6.20 for $N_{FN} = 5$ (left) and $N_{FN} = 11$ (right). In the kernel-based approximation by the Gaussian kernel k_ϵ, the scaling parameter is set to $\epsilon = 2.0$, in contrast to the defined standard. Furthermore, the approximation by the Gaussian kernel is regularized with a regularization parameter $\epsilon_{reg} = 10^{-8}$, which is slightly stricter than the defined standard. Also, the kernel-based method uses tensor-product quadrature with $l_q = 4$ in the five dimensional case and sparse grid quadrature with the same level as in the eleven-dimensional case.

It is obvious that the kernel-based stochastic collocation by Gaussian kernels clearly outperforms both tensor-product (TP) and even both sparse grid-based (SG) methods in the five-dimensional case. Exponential rates are not visible for stochastic collocation by global



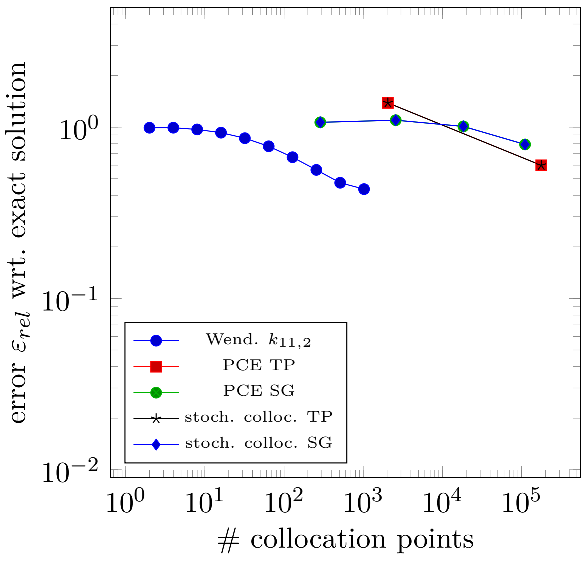


Figure 6.20: Convergence study of the mean of the g function for stochastic dimensions $N_{FN} = 5$ (left) and $N_{FN} = 11$ (right) with respect to different numerical methods.

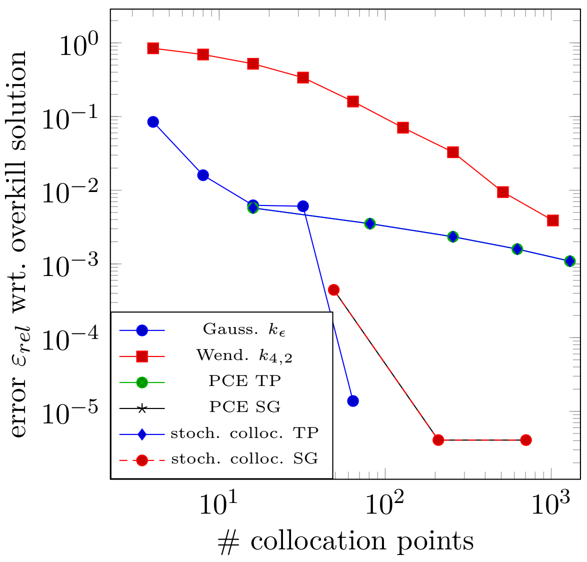
polynomials and for the Polynomial Chaos Expansion. Gaussian kernels have a much better preasymptotic behavior and even a better convergence rate. This is probably due to the limited smoothness of the problem. This is a profound result, indicating the advantage of kernel-based methods for problems with limited smoothness. In the eleven-dimensional case all methods have similar, low convergence rates. However, the Wendland kernel still has a better preasymptotic behavior.

6.2.4 Elliptic problem with piecewise constant random diffusion field

Remember from Section 3.2 that this model with piecewise constant coefficient diffusion field and four stochastic dimensions has no analytical solution. Therefore an overkill solution is considered. The reference solution will be an approximation by stochastic collocation with sparse grids on level 15. In all results, a finite-difference approximation with $N_{\mathcal{D}} = 512^2$ points is used. For kernel-based approximation, Gaussian kernels with $\epsilon = 0.5$ are taken. Full tensor product quadrature on level $l_q = 5$ is applied. All remaining parameters are left as discussed in Section 6.2.1.

Convergence results are given on the left-hand side of Figure 6.21. The Wendland kernel shows algebraic convergence. Gaussian kernels lead to high-order algebraic convergence or even exponential convergence, noting one outlier in the convergence plot. Full tensor-product-based stochastic collocation (TP) outperforms convergence results of the Wendland kernel in the preasymptotic regime. It is well-known that tensor-product stochastic collocation shows exponential convergence for this model problem. Consequently, the given results are still in the preasymptotics.

Exponential convergence is also expected for the *sparse* approximation method (SG). Since a higher interest is put on convergence results with up to thousands of collocation points, it is not possible to show this exponential result here. Nevertheless, further convergence studies



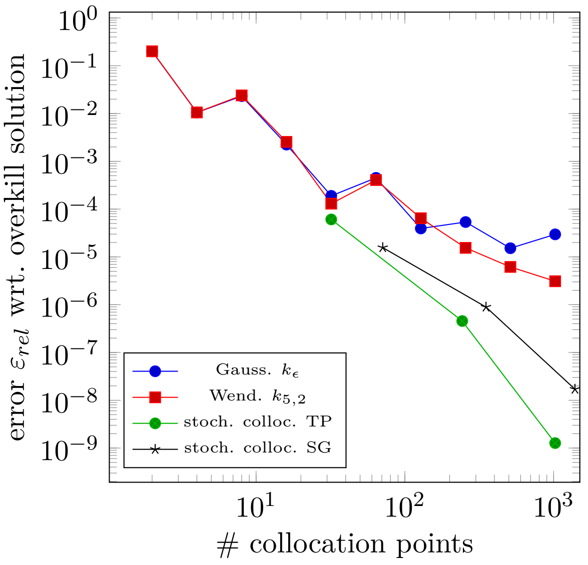


Figure 6.21: Error behavior comparison of the mean of quantity of interest in the two-dimensional piecewise constant random-coefficient Poisson problem (left) and the Karhunen-Loève expansion based random-coefficient elliptic problem (right).

(not shown here) confirm an exponential error drop with several levels of intermediate error stagnation, for the sparse-grid method. Given the results presented in Figure 6.21, on the left hand side, kernel-based stochastic collocation with Gaussians and sparse tensor-product approximation have results in the same error range. Note that Polynomial Chaos Expansion results are given, as well. However, they are identical to results from tensor-product stochastic collocation.

6.2.5 Elliptic problem with Karhunen-Loève expansion-based random diffusion

Next, we consider the two-dimensional elliptic problem with a random diffusion field, which is approximated by a Karhunen-Loève expansion. This expansion is truncated after five terms, thus we have a stochastic dimension of $N_{KL} = 5$. The correlation length is $L_c = \frac{1}{16}$. Further discussion of the problem is given in Section 3.2. As before, we have to compute an overkill solution as reference. An approximation with the sparse grid stochastic collocation method on level seven is used, since a fix-point is reached at this level. In the kernel-based method, radial basis functions are constructed with standard RBF norm and a scaling factor of $\sigma = 0.005$. In contrast to the standard, quadrature is done using full tensor-product quadrature with univariate Clenshaw-Curtis rules on level $l_q = 5$.

Figure 6.21 displays convergence results for this study, on the right-hand side. The Gaussian and Wendland kernel show similar convergence results for the chosen scaling. Sparse and full tensor-product stochastic collocation approaches clearly outperform kernel-based approximation with exponential convergence rates. Nevertheless, the error level seems to be similar on up to 100 collocation points for all methods. Notice here that the shown problem is *the* standard example for good convergence of both tensor product stochastic collocation methods.

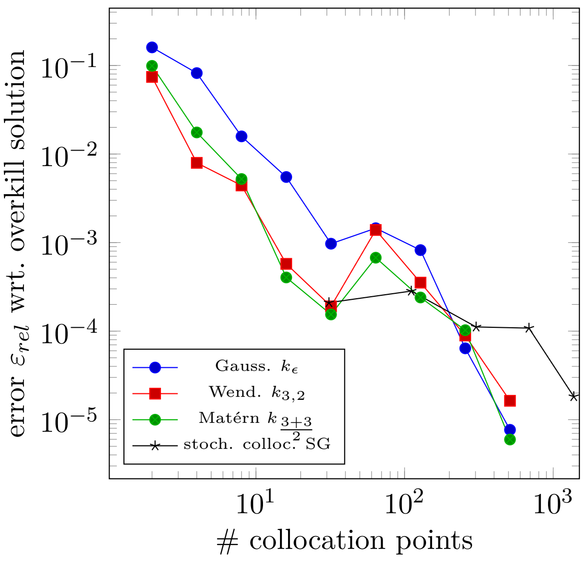


Figure 6.22: Kernel-based approximation of the mean vortex reattachment length for flows over a backward-facing step outperforms classic sparse grid-based stochastic collocation.

6.2.6 Vortex reattachment length in flows over a backward-facing step

In this study, we consider the medium-scale time-dependent flow problem from Section 3.3.3. The quantity of interest is the vortex reattachment length at a physical time of $t = 10$ seconds. Discretization of the Navier-Stokes equations is done with a space grid resolution of $N_{\mathcal{D}} = 50 \times 40 \times 3$. Velocity boundary conditions, density and viscosity are the three random inputs. Remember that they are each modeled by a one-term Karhunen-Loève expansion. We then consider a tensor product of these inputs. Therefore, we have a stochastic dimension of $N_{FN} = 3$. Kernel-based approximation with Gaussian kernels and $N_{\Gamma'} = 2^{10} = 1024$ collocation points is defined as reference overkill solution. This choice is made, since this solution is converged out well, in a reasonable way. Note that another reference solution might give different results. In contrast to the previous experiments, approximations from Dakota were not chosen, since they did not clearly converge to a fix-point. Higher orders or levels of approximation could not solve this issue.

Kernel-based approximation uses tensor-product quadrature of level $l_q = 9$. As in Section 6.1.4, the image space Γ' of the random input fields is considered for kernel-based approximation and Dakota. The Gaussian kernel is applied with standard scaling $\sigma = \epsilon = 1.0$, while Wendland and Gaussian kernels have a scaling of $\sigma = 0.1$ in the RBF norm, which is further defined as $\|\boldsymbol{y}'\| := \left\|\left(u_{\Gamma_{in}}, \mu_1, 10^{-3}\rho_1\right)^\top\right\|_2$.

Figure 6.22 shows convergence graphs for this comparison study. Due to high computational times, only sparse grid (SG) stochastic collocation is considered. Gaussian, Wendland and Matérn kernels show higher-order algebraic convergence rates. For this reason, they clearly outperform the sparse approximation method, which shows (preasymptotic) lower-order algebraic rates. In terms of error with respect to the number of collocation points, Wendland and Matérn kernels have at least the same preasymptotic error (besides one outlier for $N_\Gamma = 64$).

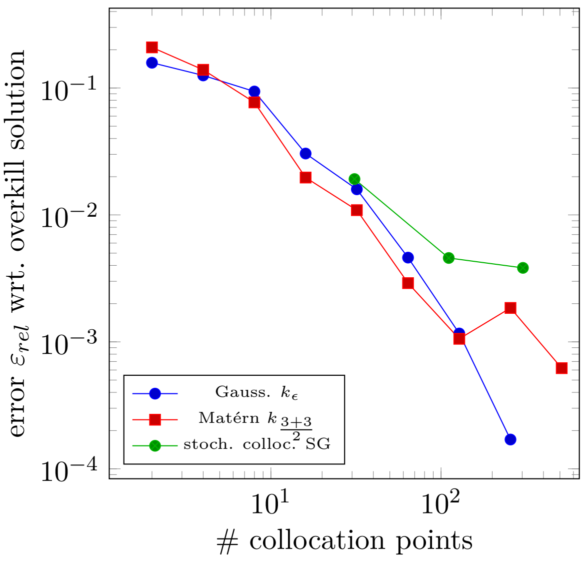


Figure 6.23: The large-scale two-phase flow problem with rising bubbles and the approximation of the mean bubble center position is clearly better solved by the kernel-based stochastic collocation approach.

Beyond approximately 200 collocation points, all kernel-based methods outperform the sparse grid-based method, which is a profound result.

6.2.7 Bubble center in flows with Karhunen-Loève based random volume force

We finally consider the problem of a rising gas bubble in water which is subject to a random volume force, cf. Section 3.3.3. Random input is modeled by a Karhunen-Loève expansion which is truncated after the third term, thus we have $N_{KL} = 3$ stochastic dimensions. A correlation length of $L_c = 2.0$ is used. The quantity of interest is the bubble's center position at a physical time of $t = 0.2$ seconds. As in the previous example, the reference solution is the mean, approximated by a Gaussian kernel, here with $N_\Gamma = 512$ collocation points. Stochastic collocation by sparse tensor product constructions, did not seem to be close to a fix-point solution for a reasonable amount of solution realizations.

In case of kernel-based approximation, the standard RBF norm $\|\cdot\| := \sigma\|\cdot\|_2$ with a modified scaling of $\sigma = 0.1$ is taken. Quadrature is full tensor-product quadrature on level $l_q = 9$. Results computed with the Gaussian kernel are regularized with $\epsilon_{reg} = 10^{-6}$ while Matérn kernel results are regularized with $\epsilon_{reg} = 10^{-5}$.

In Figure 6.23, convergence results are collected for a mean approximation by Gaussian kernels, Matérn kernels and stochastic collocation by sparse grids (SG). It becomes evident that both kernel-based methods show better convergence than the sparse grid-based method. Higher-order algebraic rates are achieved by both kernel methods. Furthermore, errors are always below the results of the classic stochastic collocation method. Remember here that a very computationally challenging problem is considered. Achieving an error with almost four valid digits with only 256 simulations is an impressive result for such a challenging problem.

6.2.8 Conclusions

Classical methods in uncertainty quantification such as Polynomial Chaos Expansion or stochastic collocation on sparse and full tensor product constructions show asymptotically exponential convergence rates on sufficiently smooth problems. Nevertheless, in many large-scale uncertainty quantification problems, the quantity of interest has either limited smoothness with respect to the random input or low preasymptotic errors are of higher importance. This is where kernel-based stochastic collocation has its main advantage. In that sense, the proposed method does not replace classical methods, but gives an additional benefit for specific large-scale random PDE problems.

6.3 Empirical error coupling analysis

In Section 6.1, we studied the convergence properties of the RBF kernel stochastic collocation with a main focus on the convergence of stochastic moment estimates with respect to the number of collocation points and different RBF kernel functions. In the presented results, quadrature and finite difference discretization were chosen as accurate as possible. Furthermore, the number of Karhunen-Loève expansion terms were understood to be fixed, thus approximation was not investigated with respect to the limit case $N_{KL} \to \infty$.

However, now, the different error couplings between the Karhunen-Loève approximation, finite difference approximation, collocation approximation and quadrature approximation shall be investigated. The ultimate goal will thus be to find conditions, such that all involved approximations have a total error below a fixed error tolerance. This will also allow to give (empirical) computational complexities in terms of the error tolerance. Furthermore, relations between approximation parameters are considered.

Remember that we discussed in Section 4.6.2 the error splitting for the evaluation of the first stochastic moment by the proposed kernel-based stochastic collocation method for time-stationary problems. We have the splitting

$$\begin{aligned}\left\|\mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right\|_{L^2(\mathcal{D})} &\leq \varepsilon_{KL} + \varepsilon_{h_\mathcal{D}} + \varepsilon_{N_\Gamma} + \varepsilon_Q \\ &= \|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))} + \|\boldsymbol{u}_{KL} - D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \\ &\quad + \|D_{h_\mathcal{D}}\boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \\ &\quad + \left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right\|_{L^2(\mathcal{D})} .\end{aligned} \tag{6.1}$$

Here, $\boldsymbol{u}$ is the exact solution of a random PDE problem, $\boldsymbol{u}_{KL}$ is its solution for random input fields that are approximated by a Karhunen-Loève expansion, D_h is the projection on the finite difference approximation space, $\mathcal{I}_{N_\Gamma}$ is the stochastic collocation approximation operator and $Q^{l,N_{KL}}$ is the quadrature operator.

The additive nature of the involved approximation errors motivates to study the influence of these errors independently. That is, the empirical error coupling analysis will discuss convergence for the Karhunen-Loève expansion approximation error ε_{KL}, the finite difference discretization error $\varepsilon_{h_\mathcal{D}}$, the stochastic collocation error ε_{N_Γ} and the quadrature error ε_Q separately. Since it is impossible for non-trivial random PDE problems to eliminate all but one approximation error, the main approach will be, to fix all but one approximation error at a reasonably high level and analyze the convergence behavior with respect to the remaining approximation. This, of course, cannot replace a full error analysis. However, it gives rise to a rough empirical relationship between all involved errors. Furthermore, it might be the only way to investigate the error behavior for a rather complex coupled problem without solution theory (in strong formulation) like the two-phase Navier-Stokes equations.

6.3.1 Error measurements in discrete norms

Before we come to the empirical analysis of all involved errors, we have a second look at the above error splitting. All described errors are measured in some continuous norm. To be able to give empirical upper bounds to the moment approximation error, we need two ingredients. The first ingredient are discrete error norms that will approximate the Bochner and Lebesgue norms which are used for continuous functions. The second ingredient are estimates relating the continuous norms to the discrete norms, in terms of error bounds. Combining both, will help to understand the relationship between measured errors and theoretical upper bounds to the error.

Let us start by introducing suitable discrete error norms. The first norm will be a discrete version of the $L^2(\mathcal{D})$ norm, which will be called $\ell^2(\mathcal{D})$ norm. It shall be defined for functions $\boldsymbol{u} \in L^2(\mathcal{D}) \cap C(\mathcal{D})$ and reads as

$$\|\boldsymbol{u}\|_{\ell^2(\mathcal{D})} := \left(\frac{1}{N_\mathcal{D}} \sum_{i=1}^{N_\mathcal{D}} \left((\mathcal{E}_{h_\mathcal{D}} \boldsymbol{u})_i \right)^2 \right)^{1/2} = \left(\frac{1}{N_\mathcal{D}} \sum_{i=1}^{N_\mathcal{D}} (\boldsymbol{u}(\boldsymbol{x}_i))^2 \right)^{1/2} .$$

The operator $\mathcal{E}_{h_\mathcal{D}}$ samples the function $\boldsymbol{u}$ at the $N_\mathcal{D}$ finite difference discretization grid points $X_{\mathcal{D},N_\mathcal{D}} = \{\boldsymbol{x}_i\}_{i=1}^{N_\mathcal{D}}$, thus is a mapping $\mathcal{E}_{h_\mathcal{D}} : L^2(\mathcal{D}) \cap C(\mathcal{D}) \to \mathbb{R}^{N_\mathcal{D}}$. In fact, for uniform finite difference grids, the $\ell^2(\mathcal{D})$ norm is a rectangular quadrature rule approximating the full $L^2(\mathcal{D})$ norm with respect to abscissas chosen identical to the finite difference grid points.

The second norm shall be defined for functions $\boldsymbol{u} \in L^2(\Gamma) \cap C(\Gamma)$, with

$$\|\boldsymbol{u}\|_{\ell^2_{MC}(\Gamma)} := \left(\frac{\mathrm{Vol}(\Gamma)}{N_{MC}} \sum_{j=1}^{N_{MC}} (\boldsymbol{u}(\boldsymbol{\xi}_j))^2 \rho(\boldsymbol{\xi}_j) \right)^{1/2} , \quad \left(\boldsymbol{\xi}_j\right)_{j=1}^{N_{MC}} \text{ uniform i.i.d. samples,} \quad \boldsymbol{\xi}_i \in \Gamma .$$

It is the standard Monte-Carlo estimator for the $L^2(\Gamma)$ norm. Note that the $\boldsymbol{\xi}_j$ are not identical to the collocation points $\boldsymbol{y}_j$. By combining both norms, it is possible to introduce the discrete version of the $L^2(\Gamma; L^2(\mathcal{D}))$ norm, namely the $\ell^2(\Gamma; \ell^2(\mathcal{D}))$ norm for functions $\boldsymbol{u} \in L^2(\Gamma; L^2(\mathcal{D})) \cap C(\Gamma; C(\mathcal{D}))$, as

$$\|\boldsymbol{u}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} := \left(\frac{\mathrm{Vol}(\Gamma)}{N_{MC}} \frac{1}{N_\mathcal{D}} \sum_{j=1}^{N_{MC}} \sum_{i=1}^{N_\mathcal{D}} \left(\boldsymbol{u}(\boldsymbol{\xi}_j, \boldsymbol{x}_i) \right)^2 \rho(\boldsymbol{\xi}_j) \right)^{1/2}$$

and the $\boldsymbol{\xi}_i$ as before. Next, we want to understand the discrete norm $\ell^2(\Gamma; \ell^2(\mathcal{D}))$ as a numerical approximation to the continuous norm $L^2(\Gamma; L^2(\mathcal{D}))$. This will allow to relate both norms to each other. With the additional requirement of $\boldsymbol{u}(\cdot,\boldsymbol{x}) \in C^2(\mathcal{D})$ and a uniform finite difference grid, we can estimate with *high probability*

$$\begin{aligned}
\|\boldsymbol{u}\|^2_{L^2(\Gamma;L^2(\mathcal{D}))} &= \int_\Gamma \int_{\mathcal{D}} (\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x}))^2 \, d\boldsymbol{x} \rho(\boldsymbol{y}) d\boldsymbol{y} && (6.2)\\
&\leq \int_\Gamma \left(\frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} (\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x_i}))^2 + c_1(\boldsymbol{y})\, h^2_{\mathcal{D}} \|\boldsymbol{u}(\boldsymbol{y},\cdot)\|_{C^2(\mathcal{D})} \right) \rho(\boldsymbol{y}) d\boldsymbol{y} && (6.3)\\
&= \int_\Gamma \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} (\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x_i}))^2 \rho(\boldsymbol{y}) d\boldsymbol{y} + \int_\Gamma c_1(\boldsymbol{y})\, h^2_{\mathcal{D}} \|\boldsymbol{u}(\boldsymbol{y},\cdot)\|_{C^2(\mathcal{D})} \rho(\boldsymbol{y}) d\boldsymbol{y} && (6.4)\\
&\leq \frac{\mathrm{Vol}(\Gamma)}{N_\Gamma} \frac{1}{N_{\mathcal{D}}} \sum_{j=1}^{N_{MC}} \sum_{i=1}^{N_{\mathcal{D}}} \left(\boldsymbol{u}(\boldsymbol{\xi}_j,\boldsymbol{x}_i) \right)^2 + c_2 N_{MC}^{-1/2} \sum_{i=1}^{N_{\mathcal{D}}} \sigma\left((\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x}_i))^2 \right) && (6.5)\\
&\quad + \int_\Gamma c_1(\boldsymbol{y})\, h^2_{\mathcal{D}} \|\boldsymbol{u}(\boldsymbol{y},\cdot)\|_{C^2(\mathcal{D})} \rho(\boldsymbol{y}) d\boldsymbol{y}\\
&= \|\boldsymbol{u}\|^2_{\ell^2(\Gamma,\ell^2(\mathcal{D}))} + c_2 N_{MC}^{-1/2} \sum_{i=1}^{N_{\mathcal{D}}} \sigma\left((\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x}_i))^2 \right) && (6.6)\\
&\quad + h^2_{\mathcal{D}} \int_\Gamma c_1(\boldsymbol{y}) \|\boldsymbol{u}(\boldsymbol{y},\cdot)\|_{C^2(\mathcal{D})} \rho(\boldsymbol{y}) d\boldsymbol{y} \,.
\end{aligned}$$

In (6.3) a tensor product quadrature with a univariate rectangle rule and with second-order convergence is introduced by adding a zero. This of course might require some additional smoothness. Remember that we have $\Gamma \subset \mathbb{R}^{N_{KL}}$. The point-wise given constant $c_1(\boldsymbol{y})$ is the usual constant of that quadrature rule. Afterwards, (6.5) introduces a Monte-Carlo quadrature rule in stochastic space. Following [Caf98, Theorem 2.1], the statement in (6.5) has to be understood probabilistically. We furthermore have for $f \in L^2(\Gamma)$ the variance of f given as

$$(\sigma(f))^2 := \frac{1}{\mathrm{Vol}(\Gamma)} \int_\Gamma \left(f(\boldsymbol{y}) - \int_\Gamma f(\boldsymbol{y}')d\boldsymbol{y}' \right)^2 d\boldsymbol{y} \,.$$

Overall, we get with the abbreviations

$$c'(\boldsymbol{u}) := \left(\int_\Gamma c_1(\boldsymbol{y}) \|\boldsymbol{u}(\boldsymbol{y},\cdot)\|_{C^2(\mathcal{D})} \rho(\boldsymbol{y}) d\boldsymbol{y} \right)^{1/2} \quad \text{and} \quad c''(\boldsymbol{u}) := \left(c_2 \sum_{i=1}^{N_{\mathcal{D}}} \sigma\left((\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x}_i))^2 \right) \right)^{1/2}$$

the approximation estimate for the continuous norm by

$$\|\boldsymbol{u}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \leq \|\boldsymbol{u}\|_{\ell^2(\Gamma,\ell^2(\mathcal{D}))} + c'(\boldsymbol{u}) N_{MC}^{-1/4} + c''(\boldsymbol{u})\, h_{\mathcal{D}} \,. \tag{6.7}$$

While classical literature sometimes uses error measurement norms like $\|\cdot\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))}$, the above estimate clearly shows that these measurements might be strongly overlaid by some not necessarily fast decaying error. Very often, there is no other way to measure errors in numerical studies, however it should be kept in mind that an error measured in a discrete norm does not necessarily reflect the error in the original continuous norm.

We finally apply the above result to the error estimates for the first stochastic moment in equations (4.22) and (4.23). By doing that, we get with the abbreviation $\boldsymbol{u}_{h_\mathcal{D}} := D_{h_\mathcal{D}}\boldsymbol{u}_{KL}$ and using linearity of integration and of the first stochastic moment

$$\left\|\mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right\|_{L^2(\mathcal{D})} \tag{6.8}$$

$$\begin{aligned}
&\overset{(6.1)}{\leq} \|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))} + \|\boldsymbol{u}_{KL} - D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))} \qquad (6.9)\\
&\quad + \|D_{h_\mathcal{D}}\boldsymbol{u}_{KL} - \mathcal{I}_{N_\Gamma}D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\|_{L^2(\Gamma;L^2(\mathcal{D}))}\\
&\quad + \left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma}D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}D_{h_\mathcal{D}}\boldsymbol{u}_{KL}\right\|_{L^2(\mathcal{D})}\\
&\leq \|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))} + \|\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \qquad (6.10)\\
&\quad + \|\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} + \left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right\|_{\ell^2(\mathcal{D})}\\
&\quad + c'\left(\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\right)N_{MC}^{-1/4} + c''\left(\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\right)h_\mathcal{D}\\
&\quad + c'\left(\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right)N_{MC}^{-1/4} + c''\left(\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right)h_\mathcal{D}\\
&\quad + c'''\left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right\|_{C^2(\mathcal{D})} h_\mathcal{D}\,.
\end{aligned}$$

In (6.10), an additional approximation by a quadrature rule in space $\mathcal{D}$ is introduced, with a constant c'''. The first four terms will be the errors under consideration in the next paragraphs. Note that the error in the Karhunen-Loève expansion is still given in its corresponding natural norm $\|\cdot\|_{L^2(\Omega;L^2(\mathcal{D}))}$. This will be discussed in Section 6.3.3.

6.3.2 Benchmark setup

In the following paragraphs, numerical estimates for the different discretized error terms will be given. This is done with respect to two model problems.

The first model problem is the random-coefficient elliptic problem in which the random input field is approximated by a Karhunen-Loève expansion, cf. Section 3.2.2. Discretization in space is done with finite differences on a uniform grid with $N_\mathcal{D} = 512^2$ unknowns. The solution is approximated by a Jacobi-preconditioned CG solver, which is said to have converged if the norm of the residual drops below 10^{-15}. The quantity of interest is the full solution field.

As second model problem, the two-phase Navier-Stokes equations with random forcing term are solved. Random volume forces are modeled by a truncated Karhunen-Loève expansion, cf. Section 3.3.3. Correlation length $L_c = 2.0$ is assumed. A uniform grid with $N_\mathcal{D} = 100^3$ unknowns is used in space discretization. Time discretization is done by a second-order Adams-Bashforth method with adaptive time-stepping. Other involved discretization methods follow Section 5.1. As proposed in Section 3.3.3, the quantity of interest is the bubble position at time $t = 0.2$. It is evaluated by *Paraview*.

All error measurements are performed with the GPU-based framework introduced in Section 4.8. Also, if not stated otherwise, we use $N_{KL} = 3$ for both model problems.

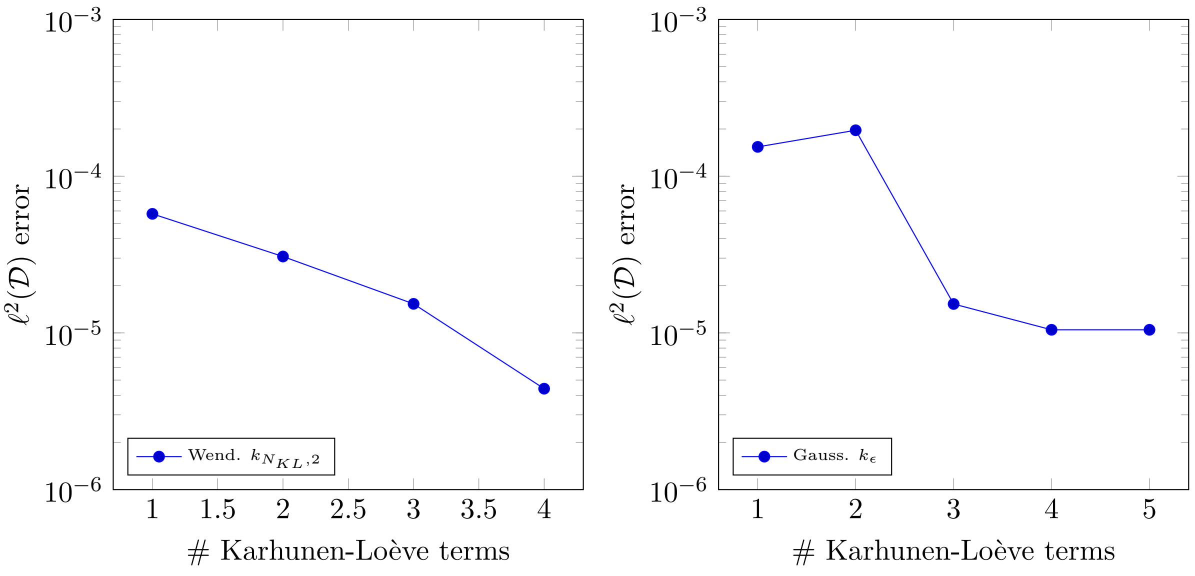


Figure 6.24: Measurement of the Karhunen-Loève expansion error for the elliptic random PDE problem (left) and for the random two-phase Navier-Stokes problem (right).

6.3.3 Karhunen-Loève expansion error

While some of the other errors are often discussed by numerical means in classical literature, the Karhunen-Loève expansion error is usually neglected, already expecting a finite-dimensional description of the random PDE problem. For many model problems, it is well-known from theory, cf. [BNT10], that the solution error decay in the number of Karhunen-Loève terms is exponential, if the input random field is modeled by an exponentially convergent Karhunen-Loève expansion. An empirical estimate of the Karhunen-Loève expansion error thus seems to be of moderate importance. However, if non-linear problems with little to no knowledge on solvability, such as the two-phase Navier-Stokes equations, come into play, theoretical results do not exist. Therefore, empirical studies become the only way to understand the properties of the involved equations (see also Part III of this thesis).

Due to missing similar empirical studies in the literature, it is not obvious, how to measure the Karhunen-Loève expansion error with respect to a random input approximated by a Karhunen-Loève expansion. Also, from a theoretical point of view, it is not clear, how to relate the error norm $L^2(\Omega; L^2(\mathcal{D}))$ to some easily computable discrete analogue. In the following, the Karhunen-Loève expansion error will be evaluated in terms of the approximation of the first moment. Therefore the original error $\|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))}$ is replaced by

$$\left\| Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \left(\boldsymbol{u} - \boldsymbol{u}_{KL}\right) \right\|_{\ell^2(\mathcal{D})}$$

with suitable modifications to the involved operators to be defined on the infinite-dimensional solution $\boldsymbol{u}$.

On the left-hand side of Figure 6.24, a convergence study of the above defined Karhunen-Loève expansion error is given for the elliptic problem with a growing number of expansion terms, i.e. stochastic dimensions, in the input. The Wendland kernel $k_{N_{KL},2}$ is used for approximation with $N_\Gamma = 2^{10}$ collocation points, regularization $\epsilon_{reg} = 10^{-12}$, Clenshaw-Curtis sparse grid quadrature at level $l_q = 9$ and a dimension-dependent norm in the kernel construction with $\|\cdot\| := 0.5\sqrt{0.1}^{N_{KL}}\|\cdot\|_2$. Displayed error results reflect the differences in $l^2(\mathcal{D})$ norm between consecutive approximation levels. The convergence graph suggests exponential error decay in the early asymptotic regime. Notice however that e.g. the collocation error might become dominant with growing dimension.

A second study is given for the random two-phase Navier-Stokes equations. Here, a Gaussian kernel k_ϵ with $\epsilon = 0.5\sqrt{0.1}^{N_{KL}}$, the standard Euclidean norm in kernel construction, approximation with $N_\Gamma = 2^8$ collocation points, regularization $\epsilon_{reg} = 10^{-8}$ and Clenshaw-Curtis sparse grid quadrature at level $l_q = 5$ is used. The error in the quantity of interest is computed with respect to the solution for $N_{KL} = 5$. A convergence graph is given on the right-hand side of Figure 6.24. This error result might be overlaid by the stochastic collocation or interpolation error, cf. Section 6.3.5. Note that there is even a small increase in error for two Karhunen-Loève terms. Nevertheless, there seem to be two error levels for approximation with one or two terms and approximation with three or more terms. The error drop might be caused by the rather large correlation length in the studied problem. Convergence stagnation could be related to dominant auxiliary errors in higher dimensions. However, this is clearly a preasymptotic result.

Overall, showing convergence with respect to a growing number of terms in the Karhunen-Loève expansion of the input random field is challenging. Dominant approximation errors in higher dimensions might influence the given results. Nevertheless, one can expect an exponential Karhunen-Loève error decay for the elliptic problem and potentially similar behavior for the Navier-Stokes case leading to

$$\|\boldsymbol{u} - \boldsymbol{u}_{KL}\|_{L^2(\Omega;L^2(\mathcal{D}))} \lesssim c\, e^{-c_{KL}(\boldsymbol{u})\, N_{KL}}\, \nu_{KL}(\|\boldsymbol{u}\|)\,,$$

where c is always a generic constant. The symbol $\lesssim$ shall indicate that the above result is only given by empirical means, at least for the Navier-Stokes equations. Measurements do not allow to quantify a suitable norm-dependence on the right-hand side. This (unknown) dependence is described by the term $\nu_{KL}(\|\boldsymbol{u}\|)$. Moreover, $c_{KL}(\boldsymbol{u})$ models problem-dependent convergence speed.

6.3.4 Finite difference approximation error

To be concise, the finite difference approximation error will not be evaluated here. In case of constant-coefficient Poisson problems with Dirichlet boundary condition and the standard five point stencil, classical literature gives an estimate for the error between the exact solution $v \in C^4(\bar{\mathcal{D}})$, $\mathcal{D} = (0,1)^2$ and the finite difference approximation v_h as

$$|v(\boldsymbol{x}) - v_h(\boldsymbol{x})| \leq c\, h_{\mathcal{D}}\, \|v\|_{C^4(\bar{\mathcal{D}})} \quad \forall \boldsymbol{x} \in X_{\mathcal{D},N_{\mathcal{D}}}\,.$$

Here, a numerical analysis would need to discuss exactly the given (parametrized) problem. Obviously, requirements on finite upper bounds of the solution with respect to the parametric input would be necessary. This is future work. However, from practice, we know that upper bounds might be expressed like

$$\|\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \lesssim c\, c_\mathcal{D}(\boldsymbol{u}_{KL})\, {h_\mathcal{D}}^2\, \nu_\mathcal{D}(\|\boldsymbol{u}_{KL}\|)\,.$$

The term $c_\mathcal{D}(\boldsymbol{u}_{KL})$ shall hide the potentially difficult dependence on the parametrized input and on e.g. discretization parameters, while $\nu_\mathcal{D}(\|\boldsymbol{u}_{KL}\|)$ models norm-dependence of the upper bound. The operator $\lesssim$ highlights that this result is only based on empirical knowledge.

Experimental studies in e.g. [CGS09] further suggest finite difference approximation error bounds of the form

$$\|\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \lesssim c\, c_\mathcal{D}(\boldsymbol{u}_{KL})\, {h_\mathcal{D}}^{p_\mathcal{D}}\, \nu_\mathcal{D}(\|\boldsymbol{u}_{KL}\|), \qquad p_\mathcal{D} \in [1,2]$$

for the velocity field of stationary random two-phase Navier-Stokes problems discretized as in Section 5.1, with the same restrictions as before. Both above empirical estimates will be used in the final error coupling.

6.3.5 Stochastic collocation approximation error

Next, the stochastic collocation approximation error, thus the term

$$\|\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))}$$

is evaluated. This is again done for the two Karhunen-Loève based random PDE problems. Convergence is compared for the Gaussian kernel k_ϵ with $\epsilon = 1$, the Wendland kernels $k_{N_{KL},1}$ and $k_{N_{KL},2}$ and the Matérn kernel k_β with $\beta = \frac{N_{KL}+3}{2}$. The kernel norm is a scaled Euclidean norm $\|\cdot\| := \sigma\|\cdot\|_2$. Kernel regularization is set to $\epsilon_{reg} = 10^{-12}$. The evaluation of the $\ell^2(\Gamma;\ell^2(\mathcal{D}))$ norm involves the computation of solutions of the random PDE as deterministic parametric problem and samplings of the response surface of the full random PDE, which is computed by the kernel-based stochastic collocation method. The sampling is done by N_{MC} appropriately distributed Monte Carlo samples $\boldsymbol{\xi}_j \in \Gamma$.

The elliptic random PDE problem is considered first. Remember that a stochastic dimension of $N_{KL} = 3$ is fixed and we set $\sigma = 1$ and $N_{MC} = 1024$. The results of this study are given on the left-hand side of Figure 6.25. As expected, Wendland and Matérn kernels give algebraic convergence rates. Exponential convergence can be achieved by the Gaussian kernel, due to the high regularity of the elliptic random PDE problem. We can conclude to have

$$\|\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \lesssim c\, e^{-c_\Gamma(\boldsymbol{u}_{h_\mathcal{D}})N_\Gamma}\, \nu_\Gamma(\|\boldsymbol{u}_{h_\mathcal{D}}\|)$$

for the elliptic problem and Gaussian kernels, with $c_\Gamma(\boldsymbol{u}_{h_\mathcal{D}})$ problem- and (stochastic) dimension-dependent convergence speed and $\nu_\Gamma(\|\boldsymbol{u}_{h_\mathcal{D}}\|)$ some general norm term. Note again that this is no real upper bound, but a rough empirical estimate.

The second problem under consideration is the random two-phase Navier-Stokes problem. Stochastic dimension is set to $N_{KL} = 3$. A scaling of $\sigma = 0.1$ is applied. Due to the extreme computational effort, only $N_{MC} = 2^7$ Monte Carlo samples are used to evaluate the norm. On the right-hand side of Figure 6.25, the convergence results for the different kernel choices are given. These results suggest a limited regularity of the mapping between the stochastic space

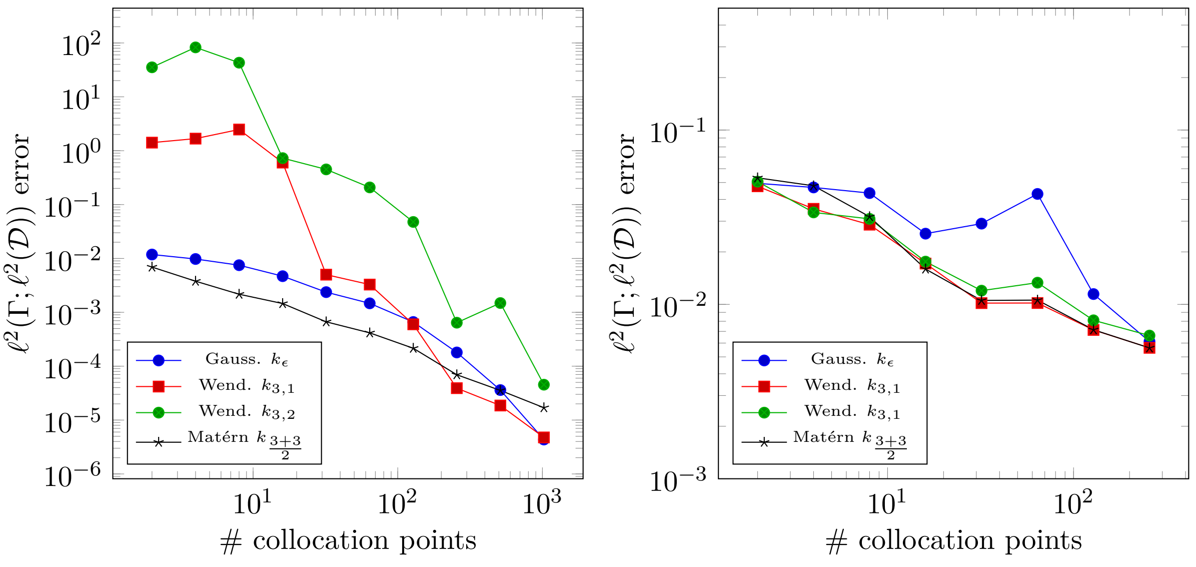


Figure 6.25: The stochastic collocation approximation error for the elliptic problem (left) depends on the applied kernel function, while the error decay is fixed to a slow algebraic rate for all kernels in the Navier-Stokes case (right), due to limited regularity.

and the quantity of interest of the solution of the two-phase Navier-Stokes equations. In fact, all kernel functions show a similar, low algebraic convergence rate, with some stability issues for the Gaussian kernel. Therefore, we get for the Navier-Stokes case an empirical stochastic collocation error estimate of the form

$$\|\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \lesssim c\, {N_\Gamma}^{-p_\Gamma(\boldsymbol{u}_{h_\mathcal{D}})}\, \nu_\Gamma(\|\boldsymbol{u}_{h_\mathcal{D}}\|)\,,$$

with $p_\Gamma(\boldsymbol{u}_{h_\mathcal{D}})$ a potentially complicated term with describes a dimension-, discretization- and problem-dependence algebraic convergence rate in stochastic space. Also, some solution norm term on the right-hand side is formally given by $\nu_\Gamma(\|\boldsymbol{u}_{h_\mathcal{D}}\|)$. However, its structure cannot be measured here.

6.3.6 Quadrature error

Finally, we have a look at the quadrature error

$$\left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right\|_{\ell^2(\mathcal{D})}\,.$$

Both random PDE problems are solved with a Gaussian kernel k_ϵ, $\epsilon = 1.0$ and the standard Euclidean kernel norm. Regularization is set to $\epsilon_{reg} = 10^{-12}$. Since we have analytic formulas (in terms of the error function) to evaluate the mean of Gaussian kernels, cf. Section 4.5.1, the above error can be evaluated up to (almost) machine precision.

For the elliptic problem with $N_{KL} = 3$, stochastic collocation is done with $N_\Gamma = 2^{10}$ collocation points. Figure 6.26 gives on the left-hand side the error decay for full tensor-product

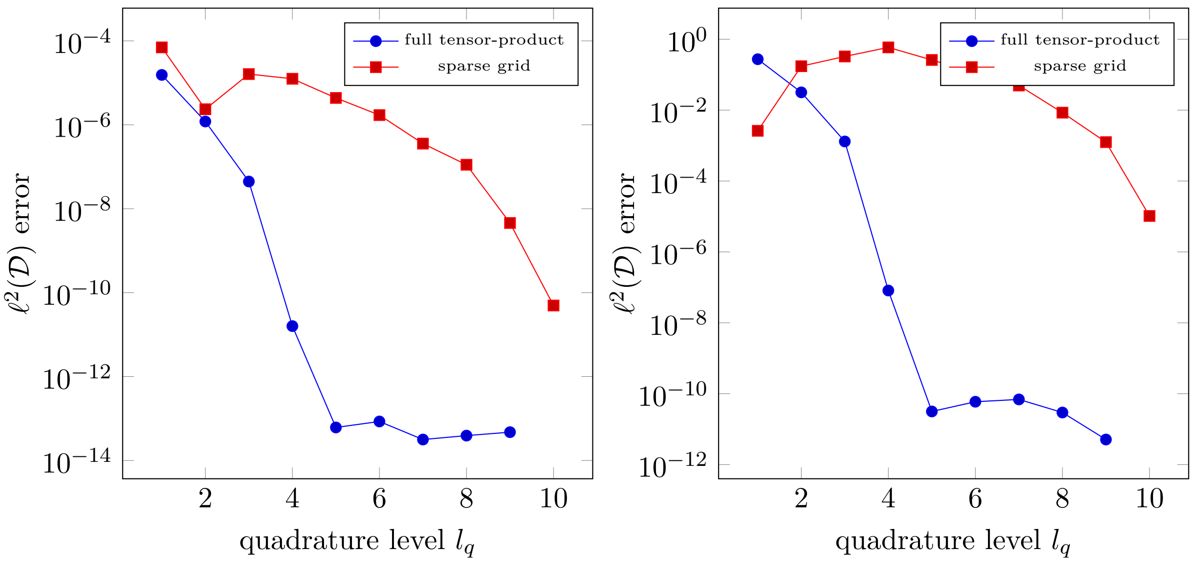


Figure 6.26: The error in the full tensor-product and sparse grid quadrature based on univariate Clenshaw-Curtis quadrature rules shows similar behavior for the elliptic problem (left) and the two-phase Navier-Stokes problem (right).

and sparse grid quadrature using univariate Clenshaw-Curtis rules with growing levels l_q. In both cases, exponential convergence is achieved. Since a rather low-dimensional problem is investigated, the full tensor-product quadrature rule still outperforms the sparse grid method, with error stagnation at machine precision. Obviously, this will not hold for high dimensions in stochastic space.

The same study is repeated for the flow problem, again with $N_{KL} = 3$, thus a three-dimensional stochastic space. Stochastic collocation is done with $N_\Gamma = 2^9$. In Figure 6.26, on the right-hand side, an almost identical error convergence behavior can be observed, compared to the elliptic case. The only difference is an offset in the overall error level, since no relative error measure is used. This result suggests that the convergence in quadrature is independent of the underlying PDE problem. For both applied quadrature techniques, it reads as

$$\left\|\mathbb{E}\left[\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}}\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\right\|_{\ell^2(\mathcal{D})} \lesssim c\, e^{-c_Q(\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}})\, l_q}\, \nu_Q(\|\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|)\,,$$

with $c_Q(\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}})$ a dimension-, method- and solution-dependent constant and $\nu_Q(\|\mathcal{I}_{N_\Gamma}\boldsymbol{u}_{h_\mathcal{D}}\|)$ some norm term, which both cannot be quantified or identified by the presented empirical convergence study. Again, the operator $\lesssim$ stresses that this is no upper bound in terms of numerical analysis, but a rough empirical guess for the dependence of the error on the quadrature level l_q.

6.3.7 Empirical error estimate and coupling

We now have all information to give empirical error estimates for the approximation of mean solutions or the means of a quantity of interest in the random elliptic and random two-phase

Navier-Stokes case. Let us remember that we had in Section 6.3.1 the estimate

$$\begin{aligned}
&\left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \\
&\overset{(6.10)}{\leq} \left\| \boldsymbol{u} - \boldsymbol{u}_{KL} \right\|_{L^2(\Omega;L^2(\mathcal{D}))} + \left\| \boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}} \right\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} \\
&\quad + \left\| \boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}} \right\|_{\ell^2(\Gamma;\ell^2(\mathcal{D}))} + \left\| \mathbb{E}\left[\mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}} \right\|_{\ell^2(\mathcal{D})} \\
&\quad + c' \left(\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\right) N_{MC}^{-1/4} + c'' \left(\boldsymbol{u}_{KL} - \boldsymbol{u}_{h_\mathcal{D}}\right) h_\mathcal{D} \\
&\quad + c' \left(\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}\right) N_{MC}^{-1/4} + c'' \left(\boldsymbol{u}_{h_\mathcal{D}} - \mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}\right) h_\mathcal{D} \\
&\quad + c''' \left\| \mathbb{E}\left[\mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}} \right\|_{C^2(\mathcal{D})} h_\mathcal{D} \,.
\end{aligned}$$

To be able to use discrete error measurements as rough empirical upper bounds to the full approximation error, we would normally need to do a rigorous analysis of the last five terms, to show that they are small and bounded from above. However, this is future work and they are expected to be neglectable. Moreover, we introduce abbreviations

$$c_{KL} := c_{KL}(\boldsymbol{u}), \quad c_\mathcal{D} := c_\mathcal{D}(\boldsymbol{u}_{KL}), \quad c_\Gamma := c_\Gamma(\boldsymbol{u}_{h_\mathcal{D}}), \quad c_Q := c_Q(\mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}), \quad p_\Gamma := p_\Gamma(\boldsymbol{u}_{h_\mathcal{D}})$$

and expect to have for all norm terms $\nu_{KL}(\|\boldsymbol{u}\|)$, $\nu_\mathcal{D}(\|\boldsymbol{u}_{KL}\|)$, $\nu_\Gamma(\|\boldsymbol{u}_{h_\mathcal{D}}\|)$ and $\nu_Q(\|\mathcal{I}_{N_\Gamma} \boldsymbol{u}_{h_\mathcal{D}}\|)$ a common upper bound of $\nu(\|\boldsymbol{u}\|)$.

Random-coefficient elliptic problem

In the elliptic case, we collect the measured results and get the purely empirical error estimate

$$\left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \lesssim c \left(e^{-c_{KL} N_{KL}} + c_\mathcal{D} {h_\mathcal{D}}^2 + e^{-c_\Gamma N_\Gamma} + e^{-c_Q l_q} \right) \nu(\|\boldsymbol{u}\|) \,. \tag{6.11}$$

Assuming an identical contribution of all measured approximation errors, we can now give conditions under which the upper bound for the approximation error in the elliptic case on the right-hand side of (6.11) drops below a given tolerance ε_{tol}. These read as

$$N_{KL} \gtrsim -\frac{1}{c_{KL}} \ln \left(\frac{1}{4\, c\, \nu(\|\boldsymbol{u}\|)} \varepsilon_{tol} \right) , \tag{6.12}$$

$$h_\mathcal{D} \lesssim \left(\frac{1}{4\, c\, c_\mathcal{D}\, \nu(\|\boldsymbol{u}\|)} \varepsilon_{tol} \right)^{1/2} \Leftrightarrow N_\mathcal{D} \gtrsim 4\, c\, c_\mathcal{D}\, \nu(\|\boldsymbol{u}\|) {\varepsilon_{tol}}^{-1} , \tag{6.13}$$

$$N_\Gamma \gtrsim -\frac{1}{c_\Gamma} \ln \left(\frac{1}{4\, c\, \nu(\|\boldsymbol{u}\|)} \varepsilon_{tol} \right) \quad \text{and} \quad l_q \gtrsim -\frac{1}{c_Q} \ln \left(\frac{1}{4\, c\, \nu(\|\boldsymbol{u}\|)} \varepsilon_{tol} \right) . \tag{6.14}$$

The modified inequality operators again indicate that these results stem back from a purely empirical analysis.

Remember that a complexity analysis for the full stochastic collocation method was given in Section 4.7. However, this analysis could not show a complexity estimate in the achievable upper error bound ε_{tol}. This is now possible by combining conditions (6.12) to (6.14) with

$$C^{elliptic}(N_\Gamma, N_\mathcal{D}, l_q, N_{KL}) = O\left(N_\Gamma {N_\mathcal{D}}^2 + N_\Gamma \left(l_q 2^{l_q}\right)^{N_{KL}} + {N_\Gamma}^3\right),$$

leading for $\varepsilon_{tol} \to 0$ to

$$\begin{aligned} & C^{elliptic}(\varepsilon_{tol}) \qquad (6.15) \\ & \approxeq O\left(-\frac{1}{c_\Gamma} \ln(\varepsilon_{tol}){\varepsilon_{tol}}^{-2} - \frac{1}{c_\Gamma} \ln(\varepsilon_{tol}) \left(-\frac{1}{c_Q} \ln(\varepsilon_{tol}){\varepsilon_{tol}}^{-\frac{1}{c_Q}} \right)^{-\ln(\varepsilon_{tol})} - \frac{1}{{c_\Gamma}^3} \ln(\varepsilon_{tol})^3 \right). \end{aligned}$$

Notice that the terms c_Γ and c_Q are kept in the estimate, since they depend on the stochastic dimension N_{KL}, thus their complexity depends on $-\ln(\varepsilon_{tol})$. Other, still (at least) problem dependent (norm) constants were dropped, due to the asymptotic nature of this statement. All results are given on an empirical base, which is underlined by $\approxeq$.

The first term in (6.15) stands for the solution of the deterministic elliptic problems, the second term describes quadrature and the last term is the complexity in the solution of the kernel interpolation problem. Based on previously given results, the complexity in quadrature is usually small or can even be neglected, due to exact quadrature evaluation formulas. However, the first and last complexity terms are dominant.

Therefore, besides of giving conditions on the different approximations to achieve a fixed total error tolerance, another interest in error coupling is the relationship between the PDE discretization parameter $h_\mathcal{D}$ and the required number N_Γ of collocation points to achieve a balanced total error. Revisiting the error estimate (6.11), a purely empirically motivated error balance between collocation error and PDE error is achieved for

$$N_\Gamma \approxeq O\left(-\frac{1}{c_\Gamma} \ln\left(c_\mathcal{D}\, {h_\mathcal{D}}^2\right)\right).$$

This empirically suggests that an algebraic reduction in PDE error only requires a logarithmic increase of the number of collocation points to achieve an overall balanced error.

Two-phase Navier-Stokes equations with random forces

We can do the same analysis for the two-phase Navier-Stokes case. With respect to the discussed quantity of interest, the purely empirical error estimate is

$$\left\| \mathbb{E}\left[\boldsymbol{u}\right] - Q^{l,N_{KL}} \mathcal{I}_{N_\Gamma} D_{h_\mathcal{D}} \boldsymbol{u}_{KL} \right\|_{L^2(\mathcal{D})} \lesssim c\left(e^{-c_{KL} N_{KL}} + c_\mathcal{D} {h_\mathcal{D}}^{p_\mathcal{D}} + N_\Gamma^{-p_\Gamma} + e^{-c_Q l_q}\right) \nu(\|\boldsymbol{u}\|). \tag{6.16}$$

The right-hand side of this empirical error shall now be bounded from above by ε_{tol}. Assuming identical contributions of all error parts gives conditions

$$N_{KL} \gtrsim -\frac{1}{c_{KL}} \ln\left(\frac{1}{4c\,\nu(\|\boldsymbol{u}\|)} \varepsilon_{tol}\right), \tag{6.17}$$

$$h_{\mathcal{D}} \lesssim \left(\frac{1}{4c\, c_{\mathcal{D}}\, \nu(\|\boldsymbol{u}\|)}\varepsilon_{tol}\right)^{1/p_{\mathcal{D}}} \Leftrightarrow N_{\mathcal{D}} \gtrsim \left(\frac{1}{4c\, c_{\mathcal{D}}\, \nu(\|\boldsymbol{u}\|)}\varepsilon_{tol}\right)^{-3/p_{\mathcal{D}}}, \tag{6.18}$$

$$N_{\Gamma} \gtrsim \left(\frac{1}{4c\, \nu(\|\boldsymbol{u}\|)}\varepsilon_{tol}\right)^{-1/p_{\Gamma}} \quad \text{and} \quad l_q \gtrsim -\frac{1}{c_Q}\ln\left(\frac{1}{4c\, \nu(\|\boldsymbol{u}\|)}\varepsilon_{tol}\right), \tag{6.19}$$

to achieve the empirical upper error bound of ε_{tol}. Combining these conditions with our knowledge on the overall complexity of the stochastic collocation method to solve the random Navier-Stokes equations

$$C^{Navier}(N_{\Gamma}, N_{\mathcal{D}}, N_{\mathcal{T}}, l_q, N_{KL}) = O\left(N_{\Gamma} {N_{\mathcal{D}}}^2 N_{\mathcal{T}} + N_{\Gamma}\left(l_q 2^{l_q}\right)^{N_{KL}} + {N_{\Gamma}}^3\right),$$

leads to an empirical computational complexity estimate for a fixed target tolerance with $\varepsilon_{tol} \to 0$ as

$$\begin{aligned} &C^{Navier}(\varepsilon_{tol}, N_{\mathcal{T}}) \\ &\approxeq O\left({\varepsilon_{tol}}^{-\frac{1}{p_{\Gamma}}-\frac{6}{p_{\mathcal{D}}}} N_{\mathcal{T}} + \varepsilon^{-\frac{1}{p_{\Gamma}}}\left(-\frac{1}{c_Q}\ln(\varepsilon_{tol}){\varepsilon_{tol}}^{-\frac{1}{c_Q}}\right)^{-\ln(\varepsilon_{tol})} - {\varepsilon_{tol}}^{-\frac{3}{p_{\Gamma}}}\right). \end{aligned} \tag{6.20}$$

Here, $N_{\mathcal{T}}$ is the required number of time steps. The convergence rate p_{Γ} and the constant c_Q depend on the stochastic dimension N_{KL} and thus asymptotically on $-\ln(\varepsilon_{tol})$. Due to the asymptotic nature of this complexity estimate, the problem-dependent term $c_{\mathcal{D}}$, which is connected to the finite difference discretization, and the norm term $\nu(\|\boldsymbol{u}\|)$ are dropped.

As in the elliptic case, the second term, i.e. the complexity in the quadrature, is usually not dominant or can be sometimes evaluated up to machine precision by exact formulas. Therefore, the first and last term, which are the computational complexity in the solution of deterministic PDEs and the complexity of kernel approximation are of dominant nature. We can relate their corresponding parameters $h_{\mathcal{D}}$ and N_{Γ} in terms of a purely empirical error balance by

$$N_{\Gamma} \approxeq O\left({c_{\mathcal{D}}}^{-\frac{1}{p_{\Gamma}}} {h_{\mathcal{D}}}^{-\frac{p_{\mathcal{D}}}{p_{\Gamma}}}\right)$$

As a result, the two-phase Navier-Stokes problem empirically requires a polynomial increase in the number of collocation points if the error in the PDE approximation is reduced.

Conclusion

The empirical error analysis allows to give rough error estimates for the kernel-base stochastic collocation approximation of the mean in case of elliptic and two-phase Navier-Stokes problems. Classical estimates for the random two-phase Navier-Stokes equations are currently impossible due to open questions with regard to existence and uniqueness of solutions of their deterministic counterparts. A measurement-based error coupling analysis allows to give empirical computational complexity estimates in terms of a fixed objective error and gives hints on the balanced choice of discretization parameters in practical applications. Note however that these relationships and results are only given in an asymptotic sense and are subject to many simplifications. Derived knowledge on the relation of stochastic and PDE approximation might give rise to some multi-level approximation scheme. However, this is future work.

Even though the previously given analysis allows to have an improved approximation parameter choice, it does not overcome the problem of high computational complexity for the solution of all deterministic PDE realizations and of the kernel approximation. This issue will be solved in the next part of this thesis. The exponential increase of the number of collocation points for growing dimension in stochastic space will be addressed in Part III.

Part II

Parallel preconditioning in uncertainty quantification

7 Local preconditioning for kernel approximation

In this chapter, preconditioning strategies for kernel-based interpolation and collocation will be discussed. Solving the interpolation or stochastic collocation problem with general positive definite kernels, thus not necessarily compactly supported kernels, usually requires the solution of dense linear systems with the number of unknowns equal to the number of collocation points. Naive approaches use some direct factorization method such as LU decomposition with $O(N_\Gamma^3)$ computational complexity to achieve this. This certainly leads to prohibitive run-times for hundreds of thousands or millions of collocation points (even on larger HPC clusters). However, such amounts of points arise in many large-scale interpolation problems. But they are also necessary for highly resolved (higher-dimensional) stochastic collocation problems with short correlation lengths, thus several dominant stochastic dimensions. The objective is to reduce the computational complexity to $O(N_\Gamma^2)$ operations by the introduction of an effectively preconditioned iterative Krylov subspace method for *dense* matrices. To this end, a preconditioner for kernel interpolation is necessary.

A suitable preconditioner will be constructed by the use of approximate Lagrange basis functions, cf. [Wen04, Fas07, FHN$^+$13]. In [FHN$^+$13], kernel interpolation problems on spheres are solved in terms of Lagrange interpolation. For special *conditionally* positive definite kernels, i.e. thin plate splines, it is possible to introduce locally constructed Lagrange basis functions as approximations to standard Lagrange basis functions. On the sphere, i.e. for boundary-free interpolation problems, a proof of convergence with the same convergence rate for the localized interpolation as for standard interpolation is available. Interestingly, these local Lagrange bases allow to construct powerful preconditioners for iterative Krylov subspace solvers, which require an optimal, problem-size independent number of iterations for the boundary-free case [FHN$^+$13]. By that way, kernel interpolation with quadratic costs becomes possible.

This approach shall be carried over to special strictly positive definite kernel functions, namely Matérn kernels. It will be shown empirically that the optimal preconditioning property still holds for these kernel functions, as long as interpolation is done on a sphere. Furthermore, numerical results will outline that the presence of a boundary does not affect the preconditioning too much. Since the given approach shall be applied to higher-dimensional problems, some numerical studies will allow to analyze the dimension-influence on the preconditioner. These promising preliminary results will motivate to apply the proposed numerical approach to higher-dimensional RBF kernel-based stochastic collocation problems with large point counts. Therefore, a multi-GPU implementation of the preconditioner and solver is given. It will allow to solve collocation problems with hundreds of thousands to millions of collocation points with almost optimal complexity $O(N_\Gamma^2)$. Note that it is even possible to get a $O(N_\Gamma \log N_\Gamma)$ solution method by introducing fast multipole or tree algorithms, cf. e.g. [BN92, BL97, BCM99, Yin06, Wen06, GD07]. However, this is future work for higher-dimensional problems in the multi-GPU context. An alternative approach for preconditioning is discussed in [SCM12, Che13].

This chapter starts with the introduction of local Lagrange bases for interpolation. Thereafter, the application in context of preconditioning is outlined. Furthermore, this kind preconditioning is reinterpreted as a special kind of restricted additive Schwarz preconditioner. Numerical results underlining optimal preconditioning for Matérn kernels on spheres and almost optimal preconditioning in presence of boundaries and in higher dimensions are given. Details on a full multi-GPU parallel implementation of both the Krylov solver and the preconditioner are discussed. Finally it will be shown that this implementation has almost perfect multi-GPU strong scaling in large-scale RBF kernel-based stochastic collocation problems.

7.1 Quasi-interpolation by local Lagrange bases

Let us introduce the basic methodology of interpolation by local Lagrange basis functions based on standard textbooks [Fas07, Wen04]. Note that the whole framework is described for *strictly positive definite* kernel functions, in contrast to [FHN+13], which also covers conditionally positive definite kernels.

In Section 4.3, we discussed the interpolation problem to find for a given function $f : \Omega \to \mathbb{R}$ and a set of points

$$X := \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_N\} \subset \Omega$$

the interpolant $s_{f,X}(\boldsymbol{y})$ with

$$s_{f,X}(\boldsymbol{y}) := \sum_{j=1}^{N} \alpha_j k(\boldsymbol{y}, \boldsymbol{y}_j) \quad \forall \boldsymbol{y} \in \Omega, \quad \text{s.th.} \quad s_{f,X}(\boldsymbol{y}_j) = f(\boldsymbol{y}_j) \quad 1 \leq j \leq N\,. \tag{7.1}$$

In terms of Lagrange interpolation, cf. Section 4.3, this reads as

$$s_{f,X}(\boldsymbol{y}) := \sum_{i=1}^{N} f(\boldsymbol{y}_i) L_i(\boldsymbol{y}) \quad \text{with} \quad L_i(\boldsymbol{y}) = \sum_{j=1}^{N} \alpha_j^i k(\boldsymbol{y}, \boldsymbol{y}_j) \quad \forall i = 1 \dots N \tag{7.2}$$

the Lagrange basis of the native space of kernel k. We will now formalize the concepts of *approximate Lagrange basis functions* $\widetilde{L}_i$ and a *quasi interpolants* $\widetilde{s}_{f,X}$ which become the local Lagrange basis functions and the respective interpolants for special point choices.

Definition 7.1 (Approximate Lagrange basis and quasi interpolant)**.** *Let be a kernel interpolation problem given as in Definition 4.7 with the alternative Lagrange basis formulation from* (7.2)*. For each $i \in \{1, \dots, N_\Gamma\}$ and collocation points $X := \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_{N_\Gamma}\}$, we introduce subsets*

$$\widetilde{X}_i \subseteq X, \qquad N_i := |\widetilde{X}_i|, \qquad \widetilde{X}_i := \{\boldsymbol{y}_{i_1}, \dots, \boldsymbol{y}_{i_{N_i}}\}, \qquad \textit{s.th.} \qquad \boldsymbol{y}_i \in \widetilde{X}_i\,.$$

The approximate Lagrange basis $\left\{\widetilde{L}_i\right\}_{i=1}^N$, $\widetilde{L}_i \in \mathcal{N}_k(\Gamma)$ *has to fulfill*

$$\widetilde{L}_i(\boldsymbol{y}) = \begin{cases} 1 & \textit{if } \boldsymbol{y} = \boldsymbol{y}_i \\ 0 & \textit{if } \boldsymbol{y} \in \widetilde{X}_i \setminus \boldsymbol{y}_i \end{cases} \qquad \textit{with} \qquad \widetilde{L}_i(\boldsymbol{y}) = \sum_{j=1}^{N_i} \widetilde{\alpha}_j^i k(\boldsymbol{y}, \boldsymbol{y}_{i_j})\,. \tag{7.3}$$

Furthermore, the quasi interpolant $\widetilde{s}_{f,X}$ *is given as*

$$\widetilde{s}_{f,X}(\boldsymbol{y}) := \sum_{i=1}^{N} f(\boldsymbol{y}_i)\widetilde{L}_i(\boldsymbol{y})\,.$$

It is obvious that such Lagrange bases and quasi interpolants exist, if their counterparts L_i and $s_{f,X}$ exist. According to the above definition, approximate Lagrange basis functions are each constructed on subsets of the full collocation or interpolation point set. Each subset is chosen such that it contains at least the interpolation point $\boldsymbol{y}_i$ associated to the corresponding Lagrange basis $\widetilde{L}_i$.

To introduce a localization and thus *local Lagrange bases*, [FHN+13] propose two different approaches. In the first approach, locality of the approximate Lagrange basis functions is introduced by fixing a radius in which neighboring collocation points are included into the local approximation. In that case, we have

$$\widetilde{X}_i := \{\boldsymbol{y} \in X \mid \|\boldsymbol{y} - \boldsymbol{y}_i\| \leq K h_{X,\Gamma} |\log h_{X,\Gamma}|\}\ ,$$

with $h_{X,\Gamma}$ the fill distance, cf. Definition 4.8. Another locality measure can be given by taking a fixed number of nearest neighbor nodes. Following [FHN+13], in case of interpolation on a three-dimensional sphere and sufficiently well distributed collocation points, a point neighborhood of radius $K h_{X,\Gamma}|\log h_{X,\Gamma}|$ is filled by roughly $K^2(\log(N_\Gamma))^2$ equally distributed collocation points with fill distance $h_{X,\Gamma}$. A similar argument leads to the upper bound of $K^d|\log h_{X,\Gamma}|^d$ collocation points in a ball of radius $K h_{X,\Gamma}|\log h_{X,\Gamma}|$ within a d-dimensional tensor-product domain Γ. This motivates to define a local Lagrange basis and a localized interpolant as follows.

Definition 7.2 (Local Lagrange basis and localized interpolant)**.** *With the requirements and nomenclature of Definition 7.1, the* local Lagrange basis $\left\{L_i^{loc}\right\}_{i=1}^N$, $L_i^{loc} \in \mathcal{N}_k(\Gamma)$, $\Gamma \subset \mathbb{R}^d$ *is an approximated Lagrange basis with the subsets*

$$X_i^{loc} := \operatorname{argmin}_{X' \subset X, |X'| = N^{loc}} \sum_{\boldsymbol{y} \in X'} \|\boldsymbol{y} - \boldsymbol{y}_i\|_2 \quad \textit{with} \quad N^{loc} := K^d |\log h_{X,\Gamma}|^d\,.$$

With that, we get the localized interpolant $s_{f,X}^{loc}$ *as as*

$$s_{f,X}^{loc}(\boldsymbol{y}) := \sum_{i=1}^{N} f(\boldsymbol{y}_i) L_i^{loc}(\boldsymbol{y})\,.$$

Figure 7.1 visualizes the proposed (not necessarily uniquely determined) construction for two collocation points $\boldsymbol{y}_i$, $\boldsymbol{y}_j$ and their respective neighborhoods X_i^{loc}, X_j^{loc}.

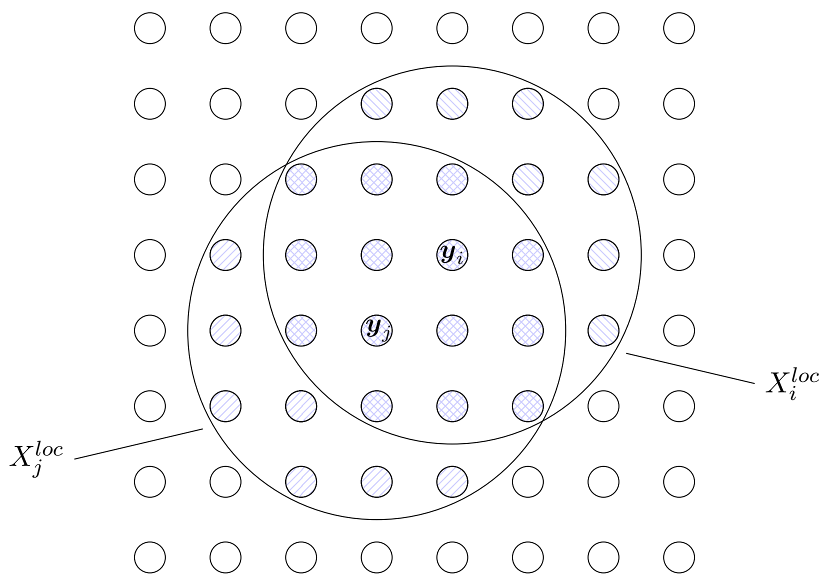


Figure 7.1: Local Lagrange basis functions at collocation points $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are constructed based on overlapping local neighborhoods X_i^{loc} (ruled from left to right) and X_j^{loc} (ruled from right to left), here exemplified for $N^{loc} = 21$.

7.2 Preconditioning using local Lagrange bases

Krylov subspace solvers for kernel interpolation systems shall be preconditioned using the approximate or local Lagrange bases. While some of the literature in kernel-based methods motivates this from an approximation or interpolation perspective, which will be outlined first, it will also be shown that the proposed method is a special restricted additive Schwarz preconditioner.

7.2.1 Motivation by improved basis functions

In [FHN$^+$13], the authors motivate the introduction of the local Lagrange basis as a right-preconditioner by observing that the standard kernel basis is often not the best possible basis in terms of conditioning of the resulting linear system. Instead, they propose to use the local Lagrange basis to replace the standard kernel basis resulting in a new interpolation problem of the form: Find interpolant $\hat{s}(\boldsymbol{y})$, such that

$$\hat{s}(\boldsymbol{y}_k) := \sum_{i=1}^{N} \mu_i L_i^{loc}(\boldsymbol{y}_k) = f(\boldsymbol{y}_k) \qquad \forall k = 1, \ldots, N\,.$$

With the definition of the local Lagrange basis, we get for all $k = 1, \ldots, N$

$$\sum_{i=1}^{N} \mu_i L_i^{loc}(\boldsymbol{y}_k) = \sum_{i=1}^{N} \sum_{j=1}^{N_i} \mu_i \alpha_j^{loc,i} k(\boldsymbol{y}_k, \boldsymbol{y}_{i_j}) = \sum_{j=1}^{N_i} \sum_{i=1}^{N} k(\boldsymbol{y}_k, \boldsymbol{y}_{i_j}) \left(\alpha_j^{loc,i} \mu_i\right) = f(\boldsymbol{y}_k)\,. \tag{7.4}$$

Let now $A_{L^{loc}}$ be the matrix which represents the coefficients $\alpha_j^{loc,i}$ in the defining equation (7.3) for approximate Lagrange basis functions. Formally, matrix $A_{L^{loc}}$ is of the form

$$A_{L^{loc}} := (a_{ik})_{i,k=1}^{N}, \qquad \text{with} \qquad a_{i,k} := \begin{cases} \alpha_j^{loc,i} & \text{if } k = i_j \\ 0 & \text{otherwise} \end{cases}. \tag{7.5}$$

It is thus the row-wise concatenation of the coefficients related to the approximate basis functions. Rewriting (7.4) in matrix notation with $\boldsymbol{\mu} = (\mu_i)_{i=1}^{N}$ and $\boldsymbol{f} = (f(\boldsymbol{y}_k))_{k=1}^{N}$ finally leads to the linear system

$$A_{k,X}(A_{L^{loc}})^{\top}\boldsymbol{\mu} = \boldsymbol{f}\,. \tag{7.6}$$

The resulting coefficients $\boldsymbol{\mu}$ have to be transfered back to the coefficients $\boldsymbol{\alpha}$ of the linear system (4.5). We therefore need

$$\boldsymbol{\alpha} = (A_{L^{loc}})^{\top}\boldsymbol{\mu}\,. \tag{7.7}$$

Equations (7.6) and (7.7) taken together, are exactly the definition of a right-preconditioner $(A_{L^{loc}})^{\top}$ for the interpolation problem linear system.

Instead of using $(A_{L^{loc}})^{\top}$ as a right-preconditioner, we can also use $A_{L^{loc}}$ as a left-preconditioner. This can be easily seen by the spectrum of the right-preconditioned linear system (7.6). There, we have by standard linear algebra arguments and knowing that $A_{k,X}$ is symmetric,

$$\sigma\left(A_{k,X}(A_{L^{loc}})^{\top}\right) = \sigma\left(\left(A_{L^{loc}}(A_{k,X})^{\top}\right)^{\top}\right) = \sigma\left(A_{L^{loc}}(A_{k,X})^{\top}\right) = \sigma\left(A_{L^{loc}}A_{k,X}\right)\,.$$

Hence, the spectra of $A_{k,X}(A_{L^{loc}})^{\top}$ and $A_{L^{loc}}A_{k,X}$ are identical. The left-preconditioned interpolation problem

$$A_{L^{loc}}A_{k,X}\boldsymbol{\alpha} = A_{L^{loc}}\boldsymbol{f}$$

should therefore have similar properties. In the following, the left-preconditioner is applied which is more usual in classical numerical linear algebra.

7.2.2 Relations to the restricted additive Schwarz method

Until now, preconditioning by local Lagrange basis functions has been mainly motivated by arguments stemming from approximation or interpolation. However, it is also possible to reinterpretate the given method in terms of Schwarz domain decomposition methods [Sch90, SBG96], which are more closely related to classical preconditioner constructions in numerical linear algebra. As it turns out, the local Lagrange preconditioning approach discussed above, is a special form of a restricted additive Schwarz (RAS) method [CS99].

Let us make the argument more specific by briefly introducing the *restricted additive Schwarz* method following [CS99]. Further information is given in [SBG96]. With the linear system of equations

$$A_{k,X}\boldsymbol{\alpha} = \boldsymbol{f}\,,$$

given, thus the classic interpolation problem as discussed before, we can introduce non-overlapping subsets $\mathcal{X}_i^0$, such that

$$\mathcal{X}_i^0 \subset X \quad \forall i = 1, \dots, N_{\mathcal{X}}, \quad \text{and} \quad X = \bigcup_{i=1}^{N_{\mathcal{X}}} \mathcal{X}_i^0, \ \bigcap_{i=1}^{N_{\mathcal{X}}} \mathcal{X}_i^0 = \emptyset \,.$$

Furthermore, we can define subsets $\mathcal{X}_i^\delta$ with $\mathcal{X}_i^0 \subset \mathcal{X}_i^\delta$, which also cover the full domain, but have some overlap with "neighboring" subsets and further all have the same size. Modifying the notation of [CS99], restriction matrices $R_i^\delta \in \mathbb{R}^{N_{\mathcal{X}_i} \times N}$ with

$$R_i^\delta = \left(r_{jk}^{i,0} \right)_{j,k}, \quad r_{jk}^{i,0} = \begin{cases} 1 & \text{if } \boldsymbol{y}_k \in \mathcal{X}_i^\delta, j = v_i(k) \\ 0 & \text{else} \end{cases}$$

are constructed. The function v_i is a mapping of the indices in the full point set X to the indices in subset $\mathcal{X}_i^\delta$.

An analogous definition is given for restriction matrices R_i^0 only covering the non-overlapping point sets. Finally, matrices

$$A_i = R_i^\delta A_{k,X} (R_i^\delta)^\top$$

are introduced. Based on this notation, it is then possible to introduce the classical *additive Schwarz preconditioner* $P_{AS} \approx A_{k,X}^{-1}$ as

$$P_{AS} = \sum (R_i^\delta)^\top A_i^{-1} R_i^\delta \,.$$

Applying the preconditioner thus corresponds to restricting the original linear problem to the subspace belonging to each subset. There, the (smaller) problem is solved with matrix A_i and projected back onto the full problem. Partial solutions for all subsets are finally added up. The restricted additive Schwarz preconditioner P_{RAS} is given as

$$P_{RAS} = \sum (R_i^0)^\top A_i^{-1} R_i^\delta \,.$$

It thus only applies corrections from each subset $\mathcal{X}_i^\delta$ to the *inner* variables or points $\mathcal{X}_i^0$. Therefore, each overlap $\mathcal{X}_i^\delta \setminus \mathcal{X}_i^0$ remains untouched.

Let us now set

$$\mathcal{X}_i^0 = \{\boldsymbol{y}_i\}, \ \mathcal{X}_i^\delta = X_i^{loc} \qquad \forall i = 1, \dots, N \,, \tag{7.8}$$

i.e. the subsets $\mathcal{X}_i^0$ are the collocation points and subsets with overlap are the local regions X_i^{loc}. It is trivial to see that the resulting restricted additive Schwarz preconditioner is identical to the local Lagrange preconditioner defined in (7.5). Consequently, the local Lagrange preconditioner is a special restricted additive Schwarz preconditioner.

7.2.3 Properties of local Lagrange bases for Matérn kernels

In this chapter, preconditioning shall be investigated for Matérn kernel functions

$$k_\beta(\boldsymbol{y}, \boldsymbol{y}') := \frac{K_{\beta - \frac{d}{2}}(\|\boldsymbol{y} - \boldsymbol{y}'\|) \|\boldsymbol{y} - \boldsymbol{y}'\|^{\beta - \frac{d}{2}}}{2^{\beta - 1} \Gamma(\beta)}, \quad \beta > \frac{d}{2} \,,$$

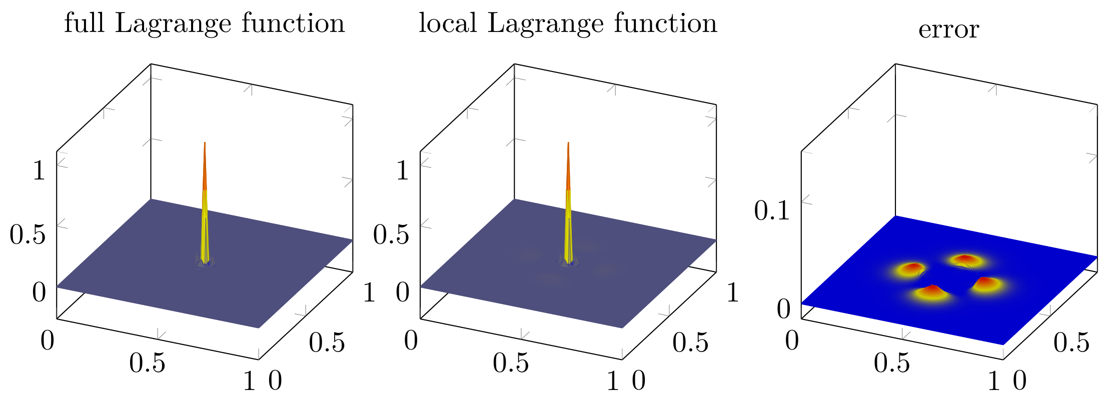


Figure 7.2: A full Lagrange basis function for Matérn kernels at an inner collocation point (left) looks almost identical as the corresponding local Lagrange basis function (middle) with a very low error (right).

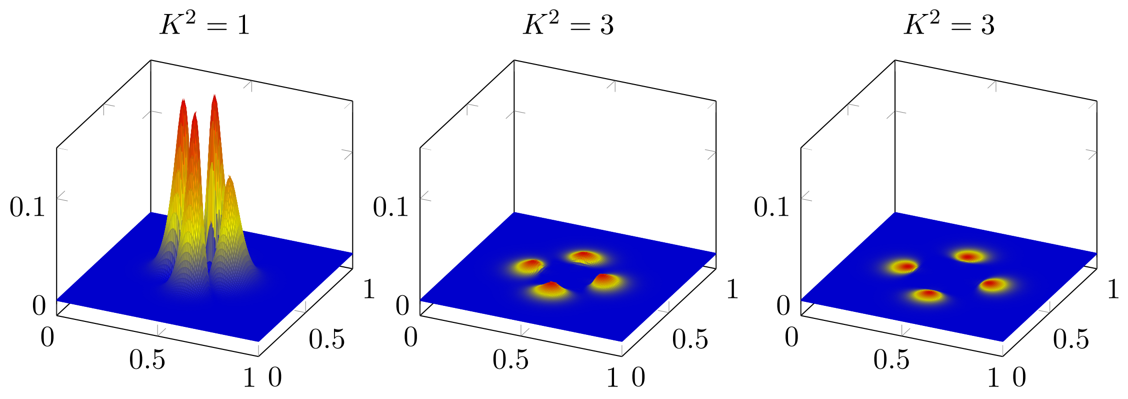


Figure 7.3: The approximation error of a local Lagrange basis function using Matérn kernels at inner collocation points becomes smaller for growing subset size parameters K^2.

cf. Section 4.3.2. The local Lagrange preconditioner is strong, if the local Lagrange functions are a decent approximation for the Lagrange functions derived by solving the full interpolation problem. Figure 7.2 shows on the left-hand side a Lagrange function for Matérn kernels with parameters $d = 2$, $\beta = 3.5$ and $\|\cdot\| := 8\|\cdot\|_2$, constructed by solving the full interpolation problem. This specific Lagrange function is centered far away from the boundary of domain $\Gamma = [0,1]^2$. In the middle of Figure 7.2, the corresponding local Lagrange function is plotted for $N^{loc} := K^2 |log(N)|^2$ and $K^2 = 3$. A difference to the standard Lagrange basis function is hardly visible. The error between the localized and the global Lagrange function is given on the right-hand side of the same figure. The ℓ_∞ error is in the range of 10^{-3}, indicating that the overall approximation quality of the preconditioner for Matérn kernels should be very good.

An important quantity in the local Lagrange preconditioning approach is the size of the local neighborhood of each local Lagrange basis or equivalently the overlap size in the restricted additive Schwarz preconditioner. Figure 7.3 highlights the influence of the local neighborhood

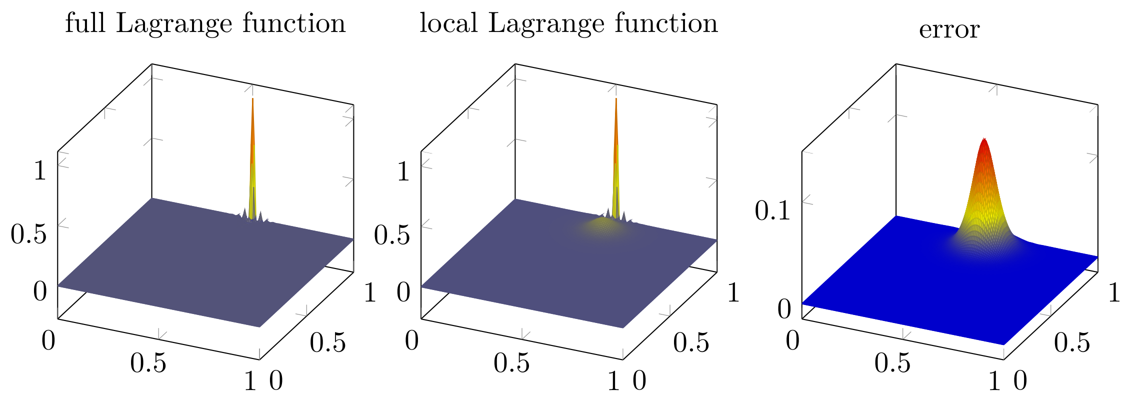


Figure 7.4: A local Lagrange function (middle) close to the domain boundary introduces an additional error (right) in contrast to the full Lagrange basis function (left).

size on the approximation quality of the local Lagrange basis. Analogously to the results on the right-hand side of Figure 7.2, we show Lagrange basis function approximation errors for subset size parameters $K^2 = 1$, $K^2 = 3$ and $K^2 = 5$ (from left to right). It becomes evident that – as expected – larger subset sizes lead to much smaller errors. In fact, if the subset size is equal to the full collocation point set, the local Lagrange preconditioner is identical to the inverse interpolation matrix ${A_{k,X}}^{-1}$.

A known issue (cf. [BCM99]) of local Lagrange functions is an increased approximation error close to the boundary. This becomes evident in Figure 7.4. Here, the numerical study from Figure 7.2 is repeated for a Lagrange function close to the boundary. Obviously, there is a much larger approximation error of the Lagrange function. The impact of this issue on preconditioning will be reviewed in Section 7.4.

7.3 Multi-GPU parallel implementation

To solve large- to extreme-scale kernel interpolation and stochastic collocation problems, a multi-GPU parallel implementation of the preconditioner setup and the preconditioned iterative solver is done. In the following, the most important technical details of this implementation are given.

7.3.1 Domain decomposition

A distributed memory multi-GPU parallelization requires a domain decomposition of the collocation point set. Each processor shall handle point subsets of similar size with a minimum of neighborhood communication for an optimal load balancing. Also, similar to the finite difference case, cf. Section 5.3, a layer of *ghost* or *halo points* is needed, since the computation of local Lagrange basis functions requires local point neighborhoods defined by the subsets X_i^{loc}. Figure 7.5 illustrates this. Here, the point set belonging to processor P_1 has an additional ghost point layer, which is indicated by the ruled circles.

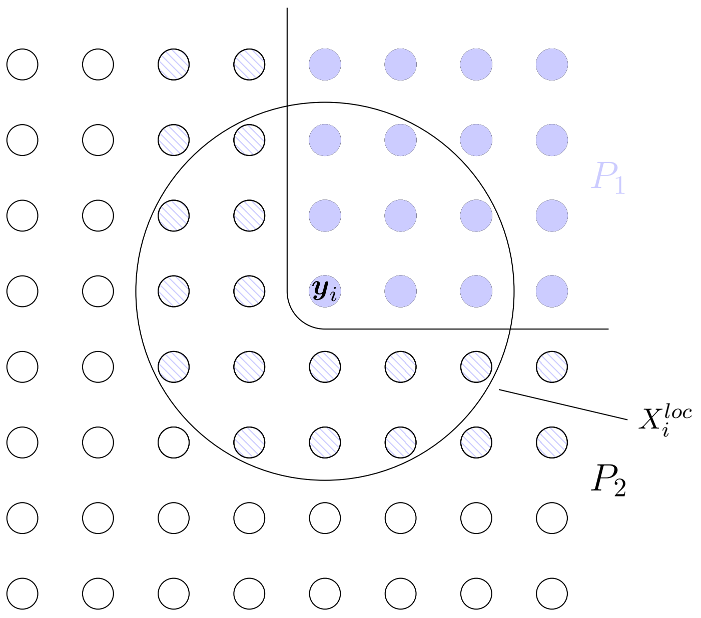


Figure 7.5: Local Lagrange basis functions that are computed on a given processor P_1 require a collocation point neighborhood from adjacent processors. This *ghost point layer* (ruled) has the size or radius of the local Lagrange subsets.

Note that generating domain decompositions of given arbitrary point sets in higher dimensions with an optimal load balancing is a research topic on its own. This shall not be solved here. Therefore, domain decomposition is considered as a pre-processing step, which is performed by a simplistic CPU implementation. It uses the machine learning library *MLPACK* [CCS+13]. To be more specific, the assignment of collocation points to processors is done by k-means clustering [Llo82] applied to the point locations with the number of clusters equal to the number of processors. Furthermore, ghost point layers are obtained by a k-nearest-neighborhood search. Overall, this approach results in similar-sized point subsets that are suitable for the given application. Figure 7.6 presents an example of domain decomposition for a spherical point set.

7.3.2 Preconditioner setup

To construct the preconditioning matrix, it is necessary to solve for each local Lagrange basis function L_i^{loc} a linear system of the form

$$A_{k,X_i^{loc}} \boldsymbol{\alpha}^{loc,i} = \boldsymbol{e}_i \qquad \forall i = 1 \dots N \,,$$

with $e_i \in \mathbb{R}^{N^{loc}}$ the ith unit vector and $A_{k,X_i^{loc}}$ the kernel interpolation matrix restricted to the local interpolation point subset. If we assume, as above, to have a fixed subset size N^{loc}, we have to solve N dense linear systems of size $N^{loc} \times N^{loc}$ with an overall computational complexity of $O(N \cdot (N^{loc})^3)$.

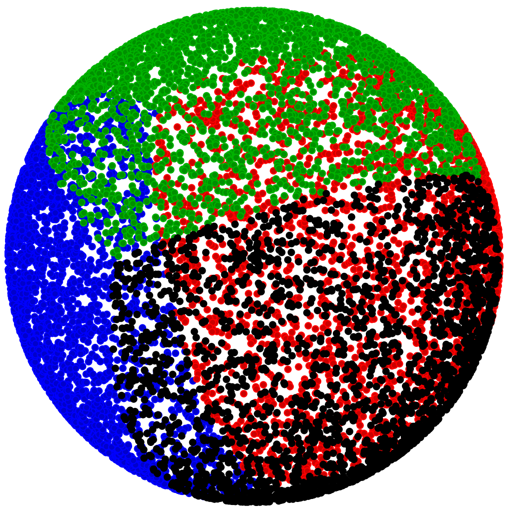


Figure 7.6: The multi-GPU implementation needs a domain decomposition of the given point cloud, here exemplified for a point set on a sphere and different colors for the four processor subdomains.

Due to the domain decomposition approach described before and the structure of the preconditioning matrix – thus it is the row-wise concatenation of the coefficients corresponding to the local Lagrange basis functions – it is possible to fully decouple the construction of the preconditioner for each subset. In the multi-GPU implementation, the coefficients for each local Lagrange basis function are independently approximated by an LU factorization given by the GPU library CULA. The subsets themselves are for now located by a GPU-based brute-force neighborhood search within each processor point subset. To the author's knowledge and at the time of writing this thesis, there is no GPU-based k-nearest-neighborhood search library available, which runs on higher-dimensional point sets and which could replace this approach. Implementing such a method is future work. Instead of constructing the full preconditioning matrix, only the local Lagrange basis coefficients are stored, resulting in a storage complexity of $O(N \cdot N^{loc})$ instead of $O(N^2)$.

7.3.3 Preconditioned Krylov subspace solver

In Section 4.5.3, we discussed Krylov iterative linear solvers for the solution of kernel interpolation systems with symmetric kernel matrices. The symmetry of the interpolation matrices stemmed back from strictly positive definite kernels. However, if we apply a non-symmetric preconditioner, as the proposed one, it is necessary to move over to a Krylov subspace method for non-symmetric problems. The preconditioned *conjugate gradient squared* (CGS) method [Son89] is chosen for this application, since, in contrast to the classical bi-conjugate gradient (BiCG) method [BBC$^+$94], it does not require the transposition of the preconditioning matrix. Transposing this matrix, or implementing the application of its transpose, would destroy the decoupling of the preconditioning for each subdomain.

The implementation of the multi-GPU parallel preconditioned CGS method follows the ideas described in Section 4.8. It is again based on the newly implemented multi-GPU parallel iterative dense linear algebra library *parla*. A naive application of the full preconditioning matrix would require $O(N^2)$ operations. This can be circumvented by a problem-specific matrix-vector implementation, which uses the structure of the preconditioning matrix. A combination of gathering and scalar product operations delivered by the GPU library *thrust*, allows the local application of each of the preconditioner's matrix rows. Thereby, the cost of each preconditioning step is reduced to $O(N \cdot N^{loc})$ operations.

Performing the interpolation matrix dense matrix-vector product costs $O(N^2)$ operations, as long as no fast matrix-vector product implementations are used, cf. [BN92, BL97, Yin06, Wen06]. While computing $O(N^2)$-matrix-vector-products is still feasible for hundreds of thousands to millions of collocation points on multiple GPUs, storing those matrices is impossible. Therefore, a GPU matrix-vector product with $O(N)$ storage complexity is used. To achieve this, small subsets of the dense interpolation matrix are constructed on-the-fly and in parallel in shared-memory of each GPU multi-processor. The application of these matrix sub-blocks to the vector is done immediately on the device by a shared-memory reduction algorithm made available in the GPU device code library CUB [Mer]. Afterwards, the old sub-block is discarded and a new sub-block is considered, within the same kernel. By that way, storing the full matrix is no longer necessary.

7.3.4 Complexities and efficiency

To summarize, the current implementation uses a preconditioner construction or setup with a theoretical $O(N \cdot (N^{loc})^3)$ computational complexity and $O(N \cdot N^{loc})$ storage complexity, noting that the local subset search is still a brute-force method that has to be replaced. Furthermore the preconditioned Krylov subspace solver is expected to have an almost constant number of iterations with $O(N^2 + N \cdot N^{loc}) = O(N^2)$ operations per iteration. The $O(N^2)$ complexity of the dense matrix-vector product, might be replaced by a $O(N \log N)$ method in the future. Candidates for this are the fast multipole method or tree algorithms [BN92, BL97, Yin06, Wen06].

The resulting numerical method for solving kernel interpolation or stochastic collocation problems will be of optimal computational complexity. However, it is crucial to note that the subset size N^{loc} might represent a quite large fraction of the overall collocation point count, in the pre-asymptotic regime. This leads to a potentially non-optimal method for small kernel interpolation problems. Efficiency, i.e. low overall runtime at optimal complexity, is only possible if the subset size is a small fraction of the overall problem size. Furthermore, the computation time for the initial preconditioner setup and the subsequent preconditioner applications in each iteration, has to be significantly smaller than the time spend for the additional iterations in the non-preconditioned iterative solver. Results, showing efficiency and parallel scalability for the given implementation, will be outlined in Section 7.4.

7.3.5 Properties for Exascale

As mentioned in Chapter 1, an important research topic in high performance computing is the discovery of numerical methods, which are well-designed for future Exascale parallel comput-

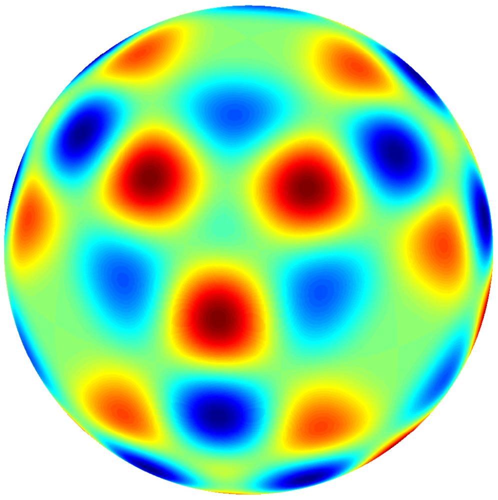


Figure 7.7: The test function for interpolation on a sphere is given on a cut surface through a tensor product of trigonometric polynomials.

ing platforms. It turns out that the proposed preconditioned kernel interpolation method is an optimal candidate in this field. The preconditioner setup completely decouples into compute-intensive, local problems, knowing that locality and an increase of compute-intensity are some of the important challenges in future Exascale clusters. Preconditioner setup and application is a massively parallel operation, which is a further important requirement for future parallel systems. Finally, the iterative nature of the proposed method gives error-resilience by construction, which is crucial due to increased expected error rates on Exascale machines.

Note that this approach can be also extended to a much broader range of applications such as high-order quadrature or PDE solvers.

7.4 Numerical results and scalability

In the following a series of numerical studies are given to analyze the convergence properties of the proposed preconditioner. For the Matlab-based studies, the Matérn kernel function

$$k_\beta(\boldsymbol{y},\boldsymbol{y}') := \frac{K_{\beta-\frac{d}{2}}(\|\boldsymbol{y}-\boldsymbol{y}'\|)\|\boldsymbol{y}-\boldsymbol{y}'\|^{\beta-\frac{d}{2}}}{2^{\beta-1}\Gamma(\beta)}, \quad \beta > \frac{d}{2},$$

is applied. Since the modified Bessel function of second kind is singular in the origin, $K_{\beta-\frac{d}{2}}(\|\boldsymbol{y}-\boldsymbol{y}'\|+\epsilon_m)$ is evaluated instead of the original Bessel function, with ϵ_m the machine precision. Due to a missing modified Bessel function of second kind on GPUs, the special case of a Matérn kernel for $\beta=\frac{d+3}{2}$, thus

$$k_{\frac{d+3}{2}}(\boldsymbol{y},\boldsymbol{y}') := (1+\|\boldsymbol{y}-\boldsymbol{y}'\|)\, e^{-\|\boldsymbol{y}-\boldsymbol{y}'\|}$$

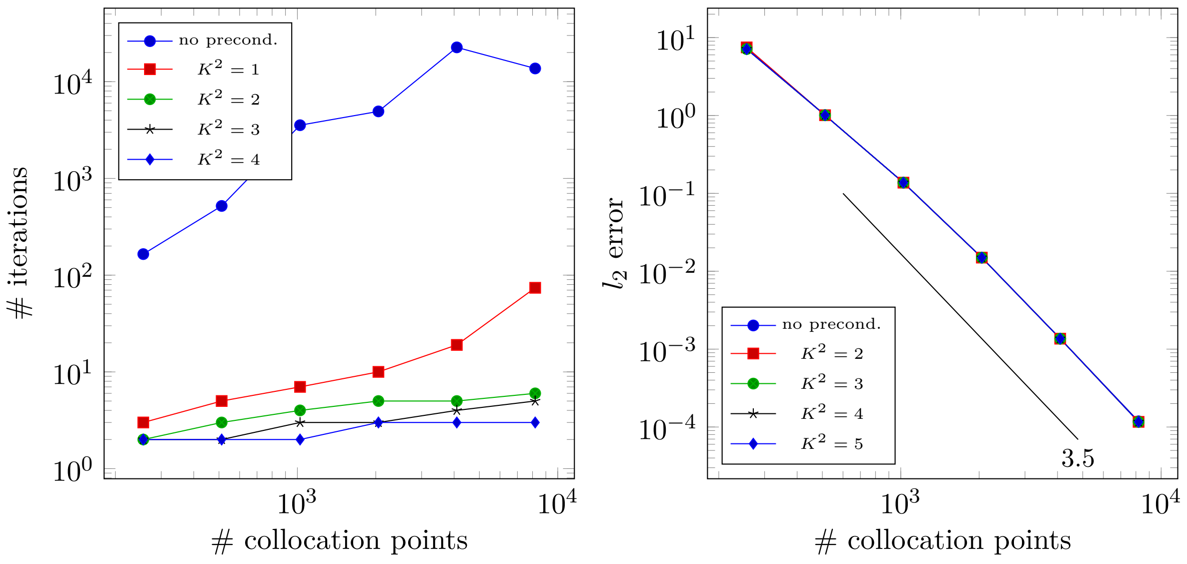


Figure 7.8: The application of a local Lagrange preconditioner allows to solve the given interpolation problem on a sphere with optimal iteration count (left) and at unchanged interpolation convergence rates (right) for $K^2 \geq 3$.

is applied on GPUs. Note that both kernels are identical up to a (dimension-dependent) constant scaling factor. Furthermore, depending on the problem, a uniform scaling is introduced to the kernel by defining the above given general norm as $\|\cdot\| := \sigma\|\cdot\|_2$

7.4.1 Preconditioning on a sphere

The first numerical results shall underline the optimality of the proposed preconditioner in case of interpolation on spheres. Here, the results are obtained with a simple Matlab implementation. We use $d = 2$, $\beta = 3.5$ and $\sigma = 8$. The test function, which is interpolated, is defined on the unit sphere $S^2 = \{\boldsymbol{y} \in \mathbb{R}^3 | \|\boldsymbol{y}\|_2 = 1\}$ by

$$\begin{aligned} f : S^2 &\to \mathbb{R} \\ \boldsymbol{y} &\mapsto \sin(2\pi y_1)\sin(2\pi y_2)\sin(2\pi y_3) \end{aligned}$$

Figure 7.7 displays the resulting function. The collocation point set on the sphere is generated by the Matlab function *RandSampleSphere* with stratified sampling. This function is provided in [Sem12] and bases on [SB96].

Figure 7.8 shows on the left-hand side the number of iterations the CGS solver needs to solve the preconditioned linear system for a growing number of collocation points. CGS is said to have converged, if the relative residual drops below 10^{-8}. As initial solution guess, the zeros vector in $\mathbb{R}^3$ is used. Results are given for growing values of K^2, i.e. growing local neighborhoods, and for the unpreconditioned solution process. For $K^2 \geq 2$, the preconditioner becomes effective. Starting from $K^2 = 3$ with subset size $N^{loc} = 244$ for $N = 8192$, it stabilizes such that less than 10 iterations are required to solve the interpolation problem. This is an optimal result, noting that an unpreconditioned CGS solver requires 13688 iterations, to solve

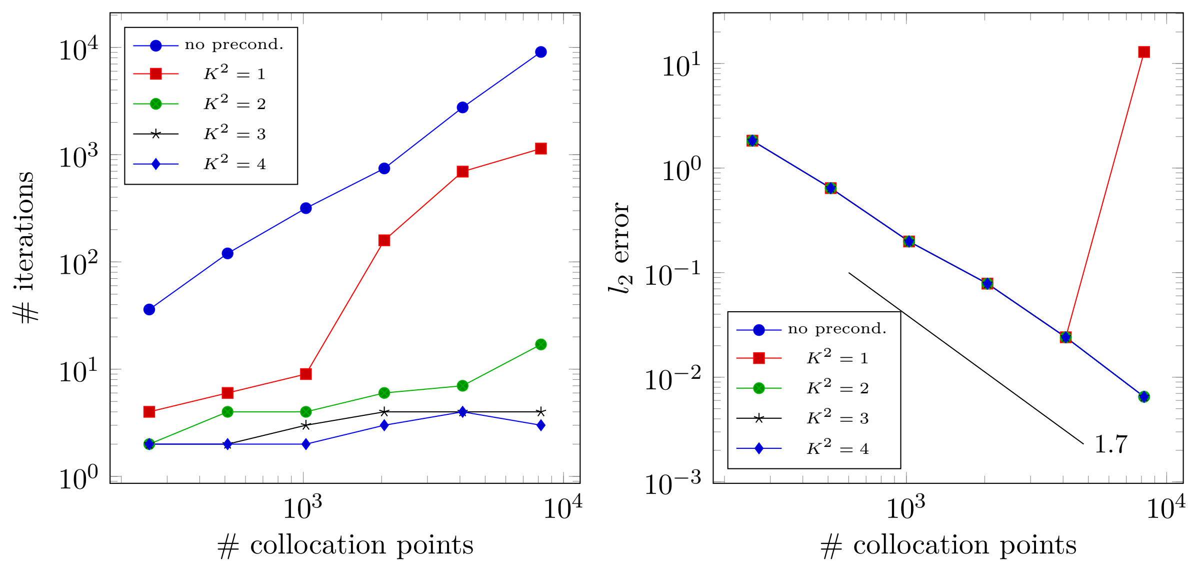


Figure 7.9: A two-dimensional interpolation problem with boundary can be efficiently preconditioned with (almost) constant iteration counts for appropriately sized subsets (left) and with constant convergence rates for $K^2 \geq 2$ (right).

the linear system for the same $N = 8192$ collocation points. On the right-hand side of Figure 7.8 the l_2 error for interpolation of the test function is given. The error is evaluated by sampling at a point set of 2^{13} points generated by *RandSampleSphere*. However, here, uniform sampling is used, cf. [Sem12], to get different evaluation points. A convergence rate of 3.5 in the number of collocation points is achieved.

7.4.2 Parameter- and dimension-dependence for problems with boundaries

Next, preconditioning for an interpolation problem on domain $\Gamma = [0,1]^d$ for growing dimensions d is discussed based on the Matlab implementation. This problem has a boundary. The test function is

$$f_d(\boldsymbol{y}) = \prod_{i=1}^{d} \sin(2\pi y_i)\,.$$

Parameters for the CGS solver are chosen as before. Collocation points are now elements of a d-dimensional Halton sequence, while error evaluation is done at 2^{14} random d-dimensional points generated by the Matlab command *rand* with no further options.

The first results are given for $d = 2$ in Figure 7.9. A Matérn kernel with $d = 2$, $\beta = 3.5$ and $\sigma = 40$ is applied. The diagram on the left-hand side gives very similar results as in the case of interpolation on a sphere. Thus, an optimal preconditioning is achieved in the two-dimensional case, starting from $K^2 = 4$ and a subset size of $N^{loc} = 325$ for $N = 8192$. The constant error convergence rate of 1.7 is usually not affected by the preconditioner. Only in case of $K^2 = 1$ and $N = 8192$ convergence breaks down, due to a too small neighborhood size. Convergence results are summarized on the right-hand side of Figure 7.9.

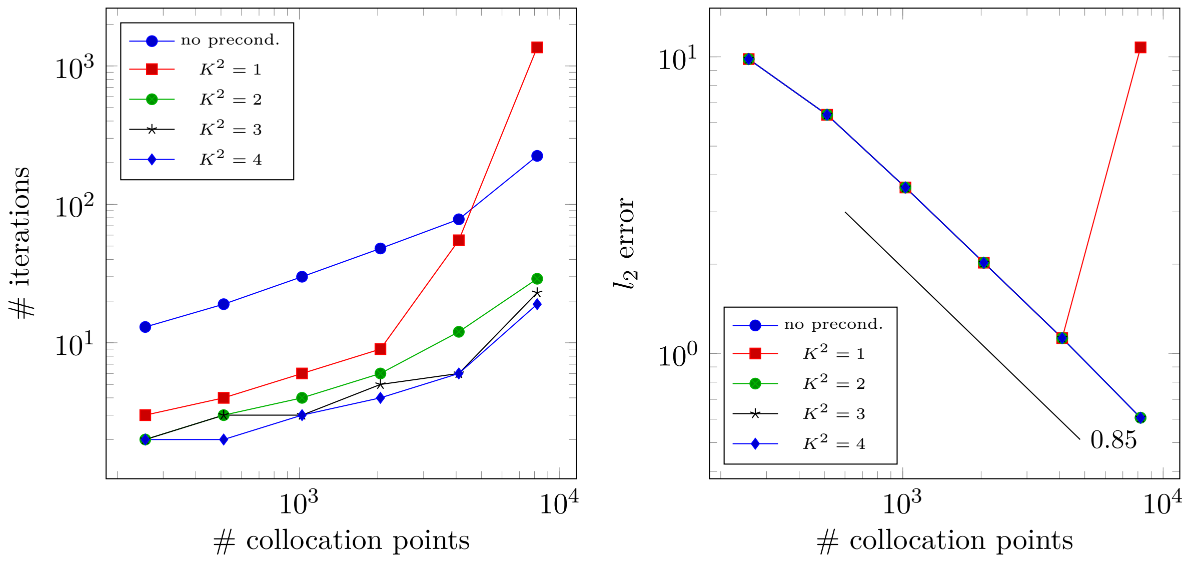


Figure 7.10: Preconditioning the interpolation problem in four-dimensional space gets weaker due to a higher boundary influence (left). Textbook convergence rates are still achieved for $K^2 \geq 2$ (right).

Next, the same study is carried out for a four-dimensional problem, thus $d = 4$. In this case the Matérn kernel with $d = 4$, $\beta = 4.5$ and $\sigma = 20$ is applied. To keep the size of the subsets small, i.e. to construct an efficient preconditioner, we choose

$$X_i^{loc} := \operatorname{argmin}_{X' \subset X, |X'| = N^{loc}} \sum_{\boldsymbol{y} \in X'} \|\boldsymbol{y} - \boldsymbol{y}_i\|_2 \quad \text{with} \quad N^{loc} := K^{\mathbf{2}} |\log h_{X,\Gamma}|^{\mathbf{2}} .$$

This is the same number of collocation points per subset as for the two-dimensional case. With this modification, the preconditioner becomes a bit weaker, as shown on the left-hand side of Figure 7.10. However, the preconditioned method is still in the range of 20 iterations for 2^{13} collocation points, $K^2 = 3$ and a subset size of $N^{loc} = 243$ for e.g. $N = 8192$, while the unpreconditioned CGS solver requires 224 iterations for the same problem size. The convergence order is reduced to 0.85 due to the increased dimension, cf. Section 4.3.2.

Finally, Figure 7.11 gives results for a six-dimensional test case, thus $d = 6$, with the Matérn kernel for $d = 6$, $\beta = 5.5$ and $\sigma = 10$. As before, subsets of size $N^{loc} := K^2 |\log h_{X,\Gamma}|^2$ are applied. While the error convergence rate on the right-hand side still follows textbook results, the preconditioning fails because of the increased boundary influence in higher dimensions, if the preconditioner still uses the same subset sizes as before. However, by introducing larger subsets sizes, some of the preconditioning capabilities can be recovered. This can be observed in Figure 7.12, where the left-hand diagram shows iteration count results for $K^2 \geq 5$. In case the subset parameter $K^2 = 10$ applied, thus subsets of size $N^{loc} = 812$ for problems of size $N = 8192$ are used, the approximated solution is attained with 20 preconditioned CGS iterations, while the unpreconditioned solution method requires 171 iterations. An optimal constant number of iterations for growing problem size could be achieved for subset size $N^{loc} := K^6 |\log h_{X,\Gamma}|^6$, based on the upper bound on the number of collocation points in a d-dimensional ball. However, in that case, preconditioning would not be computationally efficient.

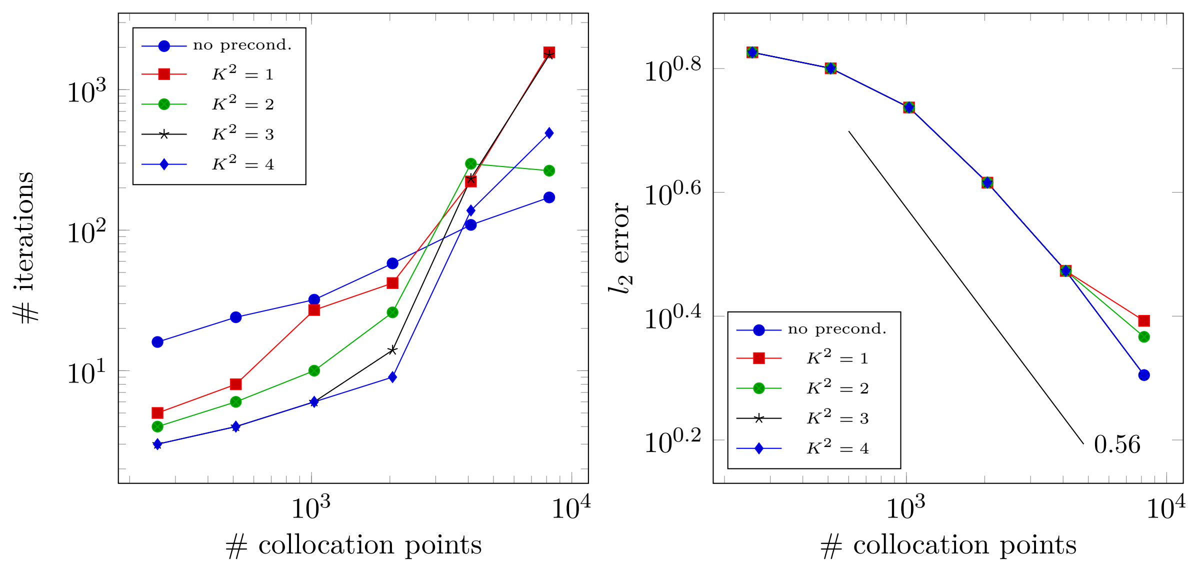


Figure 7.11: The local Lagrange preconditioner fails in case of the six-dimensional problem, if the same subset sizes are taken as in the two- or four-dimensional case (left). However convergence in the interpolation error is still achieved (right).

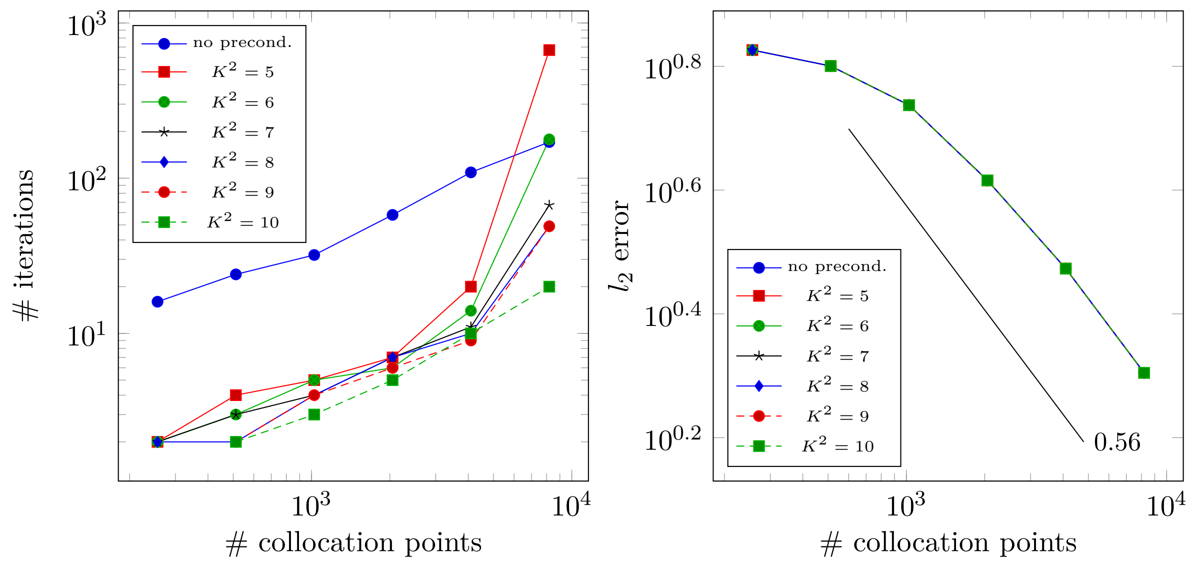


Figure 7.12: Larger subsets allow to recover some preconditioning capabilities for the six-dimensional problem (left) with correct convergence rates (right). A constant number of iterations can be expected for large subsets of size $N^{loc} := K^6 |\log N|^6$.

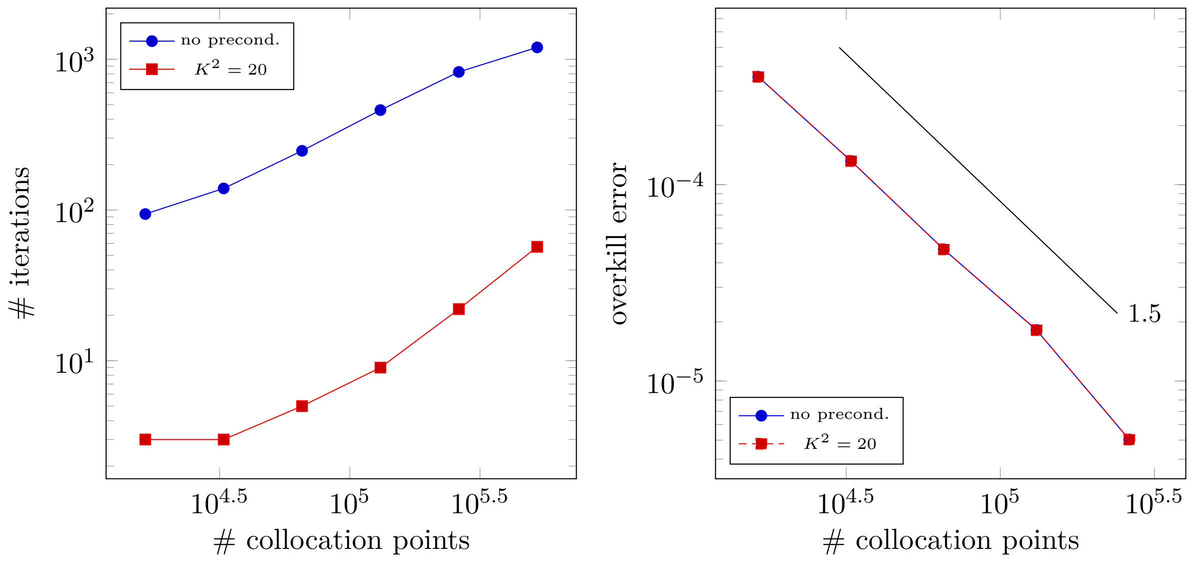


Figure 7.13: Iteration counts (left) and error convergence (right) for the elliptic random PDE test problem.

One can further observe that the number of iterations drops in the unpreconditioned case for fixed N and growing dimension. This is a dimensionality phenomenon: In higher dimensions, a much higher collocation point count is necessary to decrease the fill distance. Therefore, we have a bigger fill distance at fixed problem size. The fill distance, on the other hand, influences the condition of the interpolation problem matrix. A bigger fill distance then results in a lower condition number and thus lower iteration counts. Overall, this phenomenon leads to a reduced (relative) preconditioning efficiency in the pre-asymptotic sense for higher-dimensional problems.

7.4.3 Parallel multi-GPU solution of a large-scale RBF kernel-based stochastic collocation problem

The beforehand discussed preconditioning approach is finally applied to a large-scale kernel-based stochastic collocation problem based on a multi-GPU parallel implementation. Here, mean approximation for the elliptic problem with piecewise constant random diffusion field, stochastic dimension $N_{FN} = 4$ and the quantity of interest $\pi(u(\boldsymbol{y}, x)) := \frac{1}{N_{\mathcal{D}}} \sum_{i=1}^{N_{\mathcal{D}}} u(\boldsymbol{y}, \boldsymbol{x}_i)$ is used, cf. Section 3.2.1. The model problem under consideration represents uncertainty quantification problems with a dominant computational time for kernel interpolation due to a high necessary collocation point count and small PDE solution times. Beside of error convergence results, a parallel scalability analysis shall exemplify the almost perfect parallel scaling of the proposed method. The paragraph is closed by remarks on the efficiency of the preconditioner.

To approximate the given random PDE problem, the Matérn kernel with parameter $\beta = \frac{N_{FN}+3}{2} = 3.5$ and $\sigma = 20$ is applied. Clenshaw-Curtis tensor product quadrature with quadrature level $l_q = 6$ is used to compute the mean of the kernel functions centered at the collocation points. Furthermore a weak kernel interpolation regularization with parameter $\epsilon_{reg} = 10^{-13}$ is applied. Each realization of the elliptic problem is solved by finite differences with a resolution

		timings in seconds				
N_Γ	#GPUs	std. solver	precond. setup	precond. solver	PDE solutions	quadrature
2^{15}	1	234.7	2663.6	15.3	72089.6	1245.0
2^{15}	2	116.1	1327.6	7.9	36044.8	622.5
2^{15}	4	58.2	662.0	4.0	18022.4	311.2
2^{15}	8	29.5	329.8	2.0	9011.2	155.6
2^{15}	16	16.4	126.5	1.1	4505.6	77.8
2^{15}	32	9.3	61.7	0.6	2252.8	38.9
2^{17}	4	3312.2	4195.6	943.0	288358.4	1245.0
2^{17}	8	1526.4	2078.8	47.1	144179.2	622.5
2^{17}	16	862.9	775.1	25.8	72089.6	311.2
2^{17}	32	464.1	460.7	14.1	36044.8	155.6
2^{18}	32	3189.4	1183.8	112.6	18022.4	311.2
2^{19}	32	20191.2	2767.5	1098.5	36044.8	622.5

Table 7.1: Runtime results for the solution of the elliptic random PDE problem for growing problems sizes and GPU counts without and with preconditioning. The term *std. solver* stands for the unpreconditioned iterative CGS solver.

of $N_{\mathcal{D}} = 512 \times 512$. As usual, collocation points to solve the random PDE are given by a Halton sequence of appropriate dimensionality. The subset size in the preconditioner is fixed to $N^{loc} = 20|log(N_\Gamma)|^2$. This choice balances the cost of the preconditioner with its effectivity, noting that it does not necessarily lead to an optimal preconditioning method in sense of fixed iteration counts. Problem sizes of $N_\Gamma = 2^{15}, 2^{16,}2^{17}, 2^{18}, 2^{19}$ will be considered, thus collocation problems with up to 524288 collocation points, will be solved. The corresponding subset sizes are $N^{loc} = 2162, 2460, 2777, 3113, 3469$. The underlying preconditioned Krylov subspace solver, namely CGS, is said to have converged if the l_2 norm of the residual drops below 10^{-8}.

Figure 7.13 gives iteration counts on the left-hand side and error convergence on the right-hand side, as in the previous examples. Remember here that a non-optimal subset size has been chosen. This is why the number of iterations in the preconditioned case still grows with the number of collocation points. Nevertheless, by applying the discussed preconditioner, the number of iterations is reduced by more than an order of magnitude, which is a rather impressive result. For the largest problem size of $N_\Gamma = 2^{19}$, the preconditioned CGS solver needs 57 iterations, while the unpreconditioned solver needs 1199 iterations. Convergence results on the right-hand side of Figure 7.13 are based on error measurements with respect to the overkill solution at $N_\Gamma = 2^{19}$. The achieved rate is in the range of 1.5. It is obvious that the convergence is not affected by preconditioning.

After considering the convergence properties of the preconditioned kernel-based stochastic collocation method, we now turn towards the discussion of scalability and quality of the parallel preconditioner for this large-scale problem. Remember that the largest problem size is $N_\Gamma =$

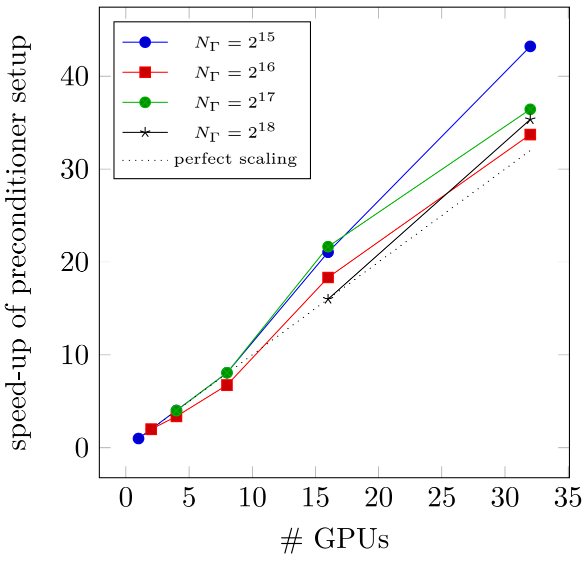


Figure 7.14: The construction of the stochastic collocation preconditioner has *perfect linear strong scaling* or speed-up, even with super-linear parallel scalability due to problem size effects.

$2^{19} = 524288$, thus in the range of more than halve a million collocation points. Solving the kernel interpolation problem with non-compactly supported Matérn kernels and *direct* solvers is almost infeasible in a reasonable amount of time, even on larger HPC clusters.

Throughout this paragraph, runtime results for the *preconditioned* approach are averages over three measurements, while results for the *unpreconditioned* approach are based on a single measurement, due to excessive runtimes. Each measurement takes the minimum runtime for the respective measured component over all involved parallel processes. Timings are given in seconds and represent wall-clock times delivered by the `gettimeofday` method. All experiments are carried out on a GPU cluster consisting of up to 36 nodes equipped each with a four-core Intel Xeon E5620 CPU at 2.4 GHz, 12 GB DDR3 RAM, an Nvidia Tesla M2090 card with 6 GB DDR5 RAM and Mellanox ConnectX-2 QDR InfiniBand 40Gbps interconnect. The operating system is Ubuntu 12.04. Furthermore, CUDA 5.0, CULA R16a and OpenMPI 1.7.5 are in use.

Table 7.1 gives timing results for the various parts of the solution process at growing problem size and changing GPU count. To avoid the recalculation of quadrature and all PDE realizations, these are precomputed once on 32 GPUs. Runtimes for the PDE solution are based on a Jacobi-preconditioned CG solver converged to a residual norm of 10^{-15} and up to 4000 iterations. In average, the solution process per PDE takes about 2.2 seconds. This number is extrapolated to the values given in Table 7.1 expecting perfect scalability for the decoupled problems. Quadrature timings are also extrapolated from the 32 GPU run. Instead of discussing all results of Table 7.1, a focus will be set on a strong scaling or speed-up analysis and an assessment of the quality of the preconditioner.

Figure 7.14 gives strong scaling or speedup results for the preconditioner setup for different problem sizes. In case a problem does not fit in GPU memory for a low GPU count, perfect scalability is assumed until the first existing result. As said before, the construction of the

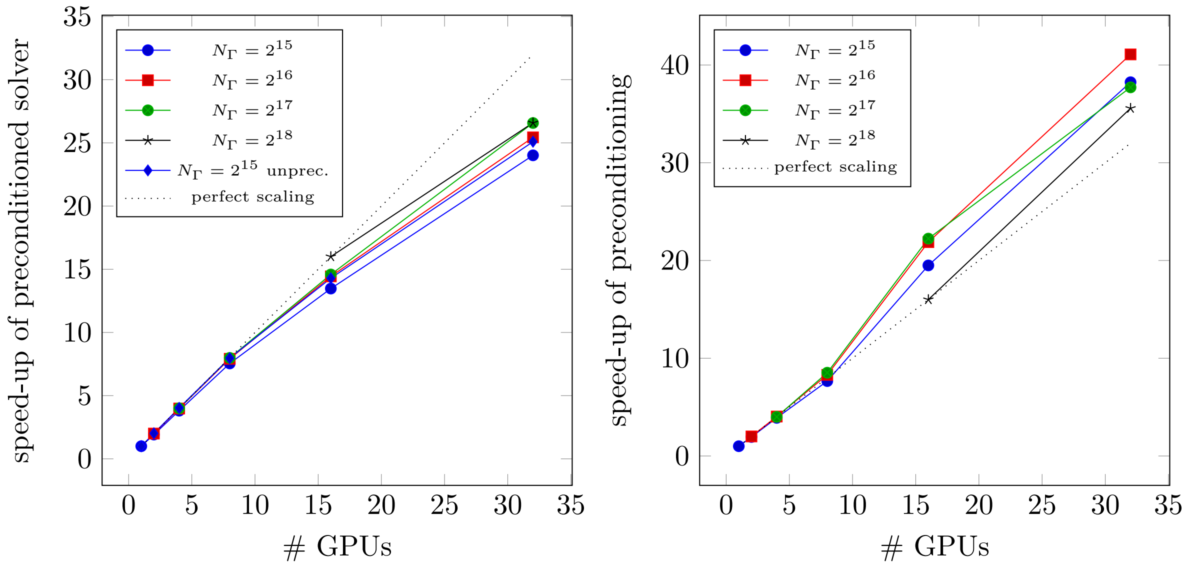


Figure 7.15: Strong scaling results for the (preconditioned) iterative CGS solver (left) and the pure preconditioner application (right) in the elliptic random PDE model problem.

local Lagrange preconditioner is a completely decoupled process, after domain decomposition. This is why perfect strong scaling is expected. Figure 7.14 confirms this excellent behavior. In fact, there is even a super-linear scaling in the number of processors. This effect might be due to per-GPU problem sizes which are more favorable for caches and memory accesses, in case of growing GPU counts.

The scalability of the preconditioned solution process is studied in Figure 7.15. On the left-hand side, strong scaling results are given for different problem sizes. Beyond 8 GPUs, there is a drop in strong scaling efficiency to about 78%. Note that this is still an almost excellent result, knowing that strong scaling parallel scalability is usually a hard problem for GPU-based codes. To rule out that the drop in performance is caused by the preconditioner, the left-hand diagram also shows a strong scaling result for the unpreconditioned case. It has the same scaling behavior as the preconditioned results. Furthermore, the right-hand side diagram of Figure 7.15, shows strong scaling results for the pure preconditioner application within the iterative method. As in the preconditioner setup, this part of the numerical method shows excellent parallel scalability, with super-linear speed-ups. It is future work, to further increase parallel scalability of the whole method.

Finally, we would like to have a look at the quality and effectivity of the preconditioner and its relation to the full solution method of the uncertainty quantification problem. This is exemplified for the case $N_\Gamma = 2^{19}$ on 32 GPUs. On the left-hand side of Figure 7.16, runtimes are compared between the unpreconditioned iterative and the preconditioned approach. The preconditioned method has a solution process of 1098.5 seconds and a setup time of 2767.5 seconds, cf. Table 7.1, being more than five times faster than the unpreconditioned method with 209191.2 seconds. This is an excellent result which underlines the quality and effectivity of the preconditioner. The preconditioner is also effective for $N_\Gamma = 2^{18}$, cf. Table 7.1, and would be effective for smaller problem sizes after adaption of the subset sizes.

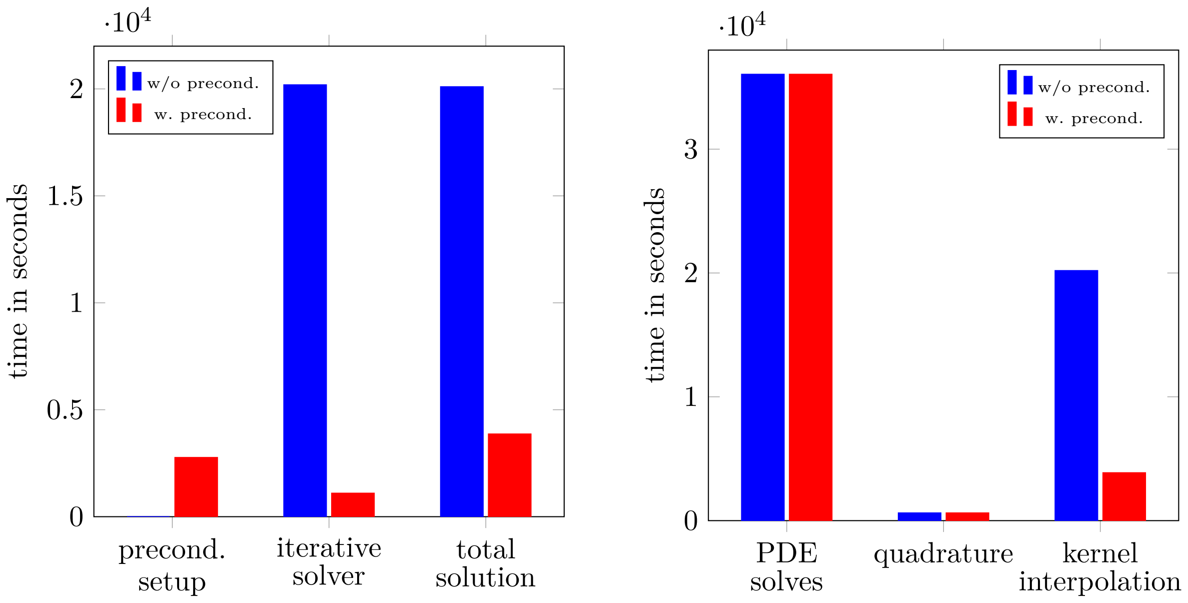


Figure 7.16: The preconditioner is highly effective for the large-scale stochastic collocation problem with $N_\Gamma = 2^{19}$ (left) and reduces the relative computing time of the kernel interpolation from more than 33% to less than 7%.

On the right-hand side of Figure 7.16, runtimes of all parts of the solution process of the elliptic random PDE problem are compared. While quadrature is comparably cheep, the solution of the kernel interpolation problem requires more than one third of the total processing time without preconditioner. Due to the preconditioner, this is impressively reduced to less than 7 percent. Now, the solution of the PDE realization again dominates the whole approach. This problem will be solved in the next chapter.

8 Algebraic multigrid for random PDE problems

In this chapter, we discuss how to achieve optimal complexity of PDE solvers for the uncertainty analysis. Since some of the involved model problems are elliptic problems, cf. Section 3.2, and the two-phase Navier-Stokes solver has its non-linear complexity part in a variable-coefficient Poisson problem with large coefficient jumps, cf. Section 5.1.2, there is a clear need for a fast linear solver for symmetric positive definite linear systems. Multigrid methods [TS01] are used intensively, to achieve optimal complexity for these kinds of elliptic problems. This class of iterative methods can be either used as preconditioner or as solver for linear systems. The algorithm starts with an grid-based approximate solution of some (elliptic) PDE problem whose high-frequency residual error is smoothed to some extend by e.g. a Gauss-Seidel relaxation. The residual (error) can be transfered to a coarser grid level on which it becomes a high-frequency error again. After solving the coarse grid problem, its solution is transfered back to the fine level where it corrects the fine grid solution. This approach can be repeated in a recursive fashion. Using that strategy, it is often possible to achieve a constant amount of iterations of the iterative solver with respect to the mesh width of the underlying elliptic problem. This is an optimality result which is highly desirable for an overall optimal complexity of the proposed uncertainty analysis. In fact, this kind of result allows to reduce the complexity of the PDE problems outlined in Chapter 3 from $O(N_\Gamma N_t {N_\mathcal{D}}^2)$ to $O(N_\Gamma N_t N_\mathcal{D})$ in the Navier-Stokes case and from $O(N_\Gamma {N_\mathcal{D}}^2)$ to $O(N_\Gamma N_\mathcal{D})$ in the elliptic case, cf. Section 4.7.

The standard approach to construct multigrid methods is to apply information from the known geometry of the problem and is called *geometric* multigrid, cf. [TS01]. It is rather easy to implement and has provable convergence rates. However, geometric multigrid methods tend to struggle for e.g. elliptic PDE problems on complex geometries. In this case and also in many applications with the need for a black-box–like multigrid method, the so-called *algebraic* multigrid is (AMG) preferred. Instead of using geometric information from the PDE problem, algebraic multigrid relies purely on the matrix entries to define the coarse level linear problems. The drawback of this approach is a more complicated algorithm and largely missing convergence theory.

Our intention is now to have an algebraic multigrid method available on GPUs, to be able to achieve optimal convergence even on this massively parallel type of hardware. One distinguishes *classical* (Ruge-Stüben) AMG [TS01, Appendix A] and *aggregation*-type AMG (e.g. smoothed or unsmoothed aggregation) [TS01, Appendix A.9]. Aggregation-type AMG for GPUs has been studied before [BDO12, BCHZ13, Vra12, Luk14], showing clear performance improvements over CPUs due to its rather simple construction method. Classical AMG on the other hand has some purely sequential construction part which is hard to parallelize on many threads [CFH$^+$99, Yan06]. Nevertheless, this approach is well-known to be more robust than smoothed aggregation AMG versions. Due to its complex algorithmic structure, there is only very limited literature and results available for purely classical AMG on GPUs [HLDP10, ENS12, KF12]. Moreover, at least to the authors knowledge, there is no freely availably implementation of classical AMG for GPUs, at the point of writing this thesis.

Therefore, in the following, an almost fully GPU-parallelized classical AMG method is proposed. Contrasting e.g. [KF12], almost the full setup phase, thus the part of the algorithm in which the multigrid hierarchy is constructed, is parallelized on GPUs. Only the purely sequential part of the the coarse/fine splitting remains on CPU until an appropriate truly parallel and fast replacement is found. The solve phase is also parallelized, as usual. Note that recently, two fully GPU-based *commercial* classical AMG implementations became available. These are AmgX [Cor] and GAMPACK [ENS12]. Even though AmgX can be downloaded as registered Nvidia developer (as a binary version), self-measured performance results are subject to a non-disclosure agreement. On the other hand, the authors of GAMPACK have published some of their performance results [ENS12]. Towards the end of this chapter, a rough comparison against this second product is given.

We start this chapter by introducing the overall idea of algebraic multigrid without further discussion of aggregation-type AMG. Subsequently, some basic terminology, thus coarsening, interpolation and smoothing is introduced. Here, the clear focus is on the algorithmic part. The important theory for this topic has been intensively discussed in e.g. [TS01, Appendix A]. In the implementation section, all important constructions for GPUs are discussed. This is followed by numerical results and performance results for model problems. Performance results will be compared to the state-of-the-art (CPU-)parallel AMG library BoomerAMG [HY02] which is available in *hypre-2.9.0b*. Furthermore the performance comparison framework for GAMPACK in [ENS12] is duplicated based on *hypre-2.8.0b*. A special paragraph is dedicated to runtime improvements for the elliptic random PDE problems from Section 3.2.

8.1 Algebraic multigrid overview

In the following, we will closely follow [TS01, Appendix A] to give a short overview of classical AMG. Notation from [TS01] will be partially adapted for improved readability.

We want to solve linear systems

$$A\boldsymbol{x} = \boldsymbol{b} \tag{8.1}$$

with $A \in \mathbb{R}^{N\times N}$ a sparse symmetric M-matrix, thus a symmetric positive definite matrix with non-positive off-diagonal entries, cf. [TS01, Sec. A.2.2], and $\boldsymbol{x}, \boldsymbol{b} \in \mathbb{R}^N$. These linear problems arise e.g. in some of the model elliptic problems and in the pressure correction of the two-phase Navier-Stokes solver.

8.1.1 Two-level method

Let us repeat the basic idea of an (algebraic) multigrid method from the introduction. We start with the original linear system on what we call the *fine level* or fine grid. Therefore, we rewrite our original problem (8.1) as

$$A_f \boldsymbol{x}^f = \boldsymbol{b}^f \quad \Leftrightarrow \quad \sum_{j\in\mathcal{D}^f} a^f_{ij} x^f_j = b^f_i \quad \forall i \in \mathcal{D}^f \,,$$

with $N_f := N$, $A_f := A$, $\boldsymbol{x}^f := \boldsymbol{x}$, $\boldsymbol{b}^f := \boldsymbol{b}$ and $\mathcal{D}^f := \{1, \ldots, N_f\}$. On the fine grid, we introduce the *smoother* $S_f \in \mathbb{R}^{N_f \times N_f}$. The overall idea of the smoother is to remove all highly oscillatory modes from the error $\boldsymbol{e}_f := \boldsymbol{x}^f_{approx} - \boldsymbol{x}^f_{exact}$. Starting from an approximation $P \approx A_f^{-1}$ to the inverse of the system matrix and the usual iterative method

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i + P(\boldsymbol{b} - A\boldsymbol{x}^i)\,,$$

the smoothing operator becomes (with I_f the identity matrix)

$$S_f := (I_f - P_f A_f)$$

and one step of the smoother generates from $\boldsymbol{x}^f$ a smoothed $\bar{\boldsymbol{x}}^f$ with

$$\bar{\boldsymbol{x}}^f := S_f \boldsymbol{x}^f + (I_f - S_f) A_f^{-1} \boldsymbol{b}^f \quad \Leftrightarrow \quad \bar{\boldsymbol{x}}^f := \boldsymbol{x}^f + P_f(\boldsymbol{b}^f - A_f \boldsymbol{x}^f)\,. \tag{8.2}$$

Examples for smoothers are a Jacobi-smoother with $P := D_f^{-1}$ (D_f is the diagonal of A_f) and a Gauss-Seidel relaxation with $P_f := L_f^{-1}$ (L_f^{-1} is the lower triangular part of A_f including all diagonal entries).

Having damped away all highly oscillatory error modes on the fine grid, it is then necessary to construct a *coarse level* or coarse grid. In classical AMG, the variables are split up in coarse grid variables (which are used on the coarse and the fine grid) and pure fine grid variables. Therefore, we introduce index sets C and F with $\mathcal{D}^f = C \cup F$, $C \cap F = \emptyset$. Using this definition, we can now define the coarse grid problem as

$$A_c \boldsymbol{x}^c = \boldsymbol{b}^c \quad \Leftrightarrow \quad \sum_{j \in \mathcal{D}^c} a^c_{ij} x^c_j = b^c_i \quad \forall i \in \mathcal{D}^c\,, \tag{8.3}$$

with the obvious choice of $\mathcal{D}^c := C$ and we have with $N_c := |C|$ that $A_c \in \mathbb{R}^{N_c \times N_c}$, $\boldsymbol{x}^c, \boldsymbol{b}^c \in \mathbb{R}^{N_c}$.

The right-hand side of the coarse grid problem will be the restriction of the residual of the fine grid problem to the coarse grid. To formulate this, we need a way to transfer solutions between the different levels. Consequently, an *interpolation operator* $I^f_c \in \mathbb{R}^{N_f \times N_c}$ is introduced. It is often also called *prolongation operator*. The opposite mapping is done by a *restriction operator* $I^c_f \in \mathbb{R}^{N_c \times N_f}$. Together, they allow to define the coarse grid system matrix A_c as

$$A_c := I^c_f A_f I^f_c\,,$$

which we call the *Galerkin operator*. Furthermore, we usually set

$$I^c_f := (I^f_c)^\top\,,$$

thus once we have defined how to perform interpolation, we know how to do the restriction operation. The restriction of the fine grid residual to construct the right-hand side of the coarse grid problem now reads as

$$\boldsymbol{b}^c := I^c_f(\boldsymbol{b}^f - A_f \boldsymbol{x}^f)\,.$$

In a pure two-level method, the coarse-level problem is solved directly. Its solution is transfered

Algorithm 4 Two-level algorithm

```
Require: $A \in \mathbb{R}^{N\times N}$ symmetric M matrix
 1: function TwoLevel($A$, $\boldsymbol{b}$)
 2:     $A_f := A, \boldsymbol{b}^f := b, \boldsymbol{x}_0^f := 0, \mathcal{D}_f = \{1,\dots,N\}$
 3:     $C \dot\cup F = \mathcal{D}_f$                          ▷ $C/F$ splitting
 4:     construct $I_c^f$, $I_f^c := (I_c^f)^\top$               ▷ transfer operators
 5:     $A_c := I_f^c A_f I_c^f$                               ▷ Galerkin operator
 6:     construct $S_f$                                        ▷ smoother
 7:     for $n = 0, 1, \dots$ do
 8:         for $s = 1, \dots, \nu_1$ do
 9:             $\boldsymbol{x}_n^f = S_f(\boldsymbol{x}_n^f, \boldsymbol{b}^f)$                  ▷ pre-smoothing
10:         $\boldsymbol{b}_n^c = I_f^c(\boldsymbol{b}^f - A_f \boldsymbol{x}_n^f)$                 ▷ residual restriction
11:         solve $A_c \boldsymbol{x}_n^c = \boldsymbol{b}_n^c$                        ▷ coarse grid solve
12:         $\boldsymbol{x}_{n+1}^f = \boldsymbol{x}_n^f + I_c^f \boldsymbol{x}_n^c$                   ▷ update solution
13:         for $s = 1, \dots, \nu_2$ do
14:             $\boldsymbol{x}_{n+1}^f = S_f(\boldsymbol{x}_{n+1}^f, \boldsymbol{b}^f)$             ▷ post-smoothing
15:     return $\boldsymbol{x}_{n+1}^f$
```

back to the fine grid by the coarse grid correction

$$\boldsymbol{x}^f = \boldsymbol{x}^f + I_c^f \boldsymbol{x}^c$$

and another smoothing step is applied. By iterating this approach, one gets an iterative linear two-level solver as stated in Algorithm 4. In the algorithm, the smoother application formula from (8.2) is replaced by a short functional representation which can be repeated ν_1 or ν_2 times, respectively. The algorithm furthermore exposes some structure, which is usual in algebraic multigrid. Lines 3 to 6 are pre-processing operations while the remaining part is an iterative algorithm. In AMG methods we call the pre-processing part *setup phase* and the remaining part *solve phase*. It will turn out, that the setup phase is actually the algorithmically more challenging part.

8.1.2 Multigrid

By carefully choosing all numeric ingredients it is possible to have linear complexity in the degrees of freedom for all operations on the fine grid during one iteration of the solve phase, cf. [TS01, Appendix A] for more details. However, directly solving the coarse problem (8.3) still has a computational complexity of $O({N_c}^3)$. Keeping in mind that the number of unknowns on the coarse grid is usually not much less than $N_f/4$, this results in a non-optimal method.

To get around this, the algorithmic idea of the two-level algorithm can be applied recursively to construct a multigrid method. To do that, we introduce levels $l = 0, 1, \dots, l_{max}$ with nested sets of variables $\mathcal{D}_0 \subset \mathcal{D}_1 \subset \ldots \subset \mathcal{D}_{l_{max}}$, and level l_{max} is what we had as finest level $\mathcal{D}_f$ before. Analogously we introduce per-level coarse/fine grid sets C^l/F^l, matrices A_l, vectors $\boldsymbol{x}^l$, $\boldsymbol{b}^l$, smoothers S_l, interpolation operators I_l^{l+1} and restriction operators I_{l+1}^l. The construction

Algorithm 5 Recursive multigrid algorithm (V-cycle)

```
Require: complete multigrid hierarchy already set up
 1: function AMG($l$, $\boldsymbol{x}^l$, $\boldsymbol{b}^l$)
 2:     if l>0 then                                          ▷ finer levels
 3:         for $s = 1, \dots, \nu_1$ do
 4:             $\boldsymbol{x}^l = S_l(\boldsymbol{x}^l, \boldsymbol{b}^l)$                      ▷ pre-smoothing
 5:         $\boldsymbol{b}^{l-1} = I_l^{l-1}(\boldsymbol{b}^l - A_l \boldsymbol{x}^l)$                 ▷ residual restriction
 6:         $\boldsymbol{x}^{l-1} = 0$
 7:         AMG($l-1$, $\boldsymbol{x}^{l-1}$, $\boldsymbol{b}^{l-1}$)                  ▷ solve on coarse grid
 8:         $\boldsymbol{x}^l = \boldsymbol{x}^l + I_{l-1}^l \boldsymbol{x}^{l-1}$                   ▷ update solution
 9:         for $s = 1, \dots, \nu_2$ do
10:             $\boldsymbol{x}^l = S_l(\boldsymbol{x}^l, \boldsymbol{b}^l)$                      ▷ post-smoothing
11:         return $\boldsymbol{x}^l$
12:     else                                                 ▷ coarsest level
13:         $\boldsymbol{x}^0 = A_0^{-1} \boldsymbol{b}^0$                              ▷ direct solve
14:         return $\boldsymbol{x}^0$
```

of coarser levels is usually stopped as soon as the number of variables per level goes below a certain threshold and the coarsest grid is solved directly.

Algorithms 5 and 6 sum up the multigrid method. The first one highlights one step of the resulting iterative method. Critical is the difference in step 7, in which the algorithm is called recursively instead of solving the coarse grid problem directly. This version of the multigrid construction is also called V-cycle. Note that in practice, there are two ways, to use a multigrid V-cycle in iterative linear solvers. The first approach is highlighted in Algorithm 6. After the setup phase in lines 2–7, a single iterative application of the V-cycle solves the linear system up to some residual error ϵ_{res}. Another approach is to run the setup phase, as before, and then to apply the V-cycle as preconditioner in e.g. a conjugate gradient method. The second approach is often preferred due to higher robustness.

In the next sections, we will shortly discuss the way of choosing the C/F splitting, the interpolation and the smoothers.

8.2 Coarsening, interpolation and smoothing

8.2.1 Coarsening

The crucial sequential part of the classical AMG method is the choice of coarse and fine grid points (C-/F-points), also called *C/F splitting*. To explain the underlying algorithm we need a bit of notation closely following [TS01], as before. Note that it is usual in AMG methods to identify a variable with a node or point in a graph. *Connections* between nodes exist if there is a non-zero non-diagonal entry in the system matrix A relating one variable to the other. This gives rise to the equivalent description of a linear system by a graph with weighted edges having as edge weights the non-diagonal entries of the matrix. Due to this point of view, the usual notation in AMG is closely related to typical notation used in graph theory.

Algorithm 6 Iterative algebraic multigrid solver

```
Require: $A \in \mathbb{R}^{N\times N}$ symmetric M matrix
 1: function AMGsolver($A$, $\boldsymbol{b}$, $\epsilon_{res}$)
 2:     $A_{l_{max}} := A, \mathcal{D}_{l_{max}} := \{1, \ldots, N\}$
 3:     for $l = l_{max}, \ldots, 1$ do                         ▷ multigrid hierarchy setup
 4:         $C^l \dot{\cup} F^l = \mathcal{D}_l$, $\mathcal{D}_{l-1} := C^l$                    ▷ $C/F$ splitting
 5:         construct $I^l_{l-1}$, $I^{l-1}_l := (I^l_{l-1})^\top$                ▷ transfer operators
 6:         $A_{l-1} := I^{l-1}_l A_l I^l_{l-1}$                              ▷ Galerkin operator
 7:         construct $S_l$                                         ▷ smoother
 8:     $\boldsymbol{x}_0 := 0, n := 0$
 9:     repeat                                                  ▷ iterative solution
10:         $x_{n+1} = \mathrm{AMG}(l_{max}, \boldsymbol{x}_n, \boldsymbol{b})$                     ▷ multigrid cycle
11:         $r_{n+1} = \boldsymbol{b} - A\boldsymbol{x}_{n+1}$
12:     until $\|r_{n+1}\|_2 \leq \epsilon_{res}$
13:     return $\boldsymbol{x}_{n+1}$
```

Let us now start by introducing neighboring variables or points. Thus, the neighborhood of a point $i \in \mathcal{D}^l$ is

$$N_i^l := \left\{ j \in \mathcal{D}^l \middle| j \neq i, a_{ij}^l \neq 0 \right\} .$$

We say that a variable i is *strongly negatively coupled* to variable j if for a fixed $0 < \epsilon_{str} < 1$

$$-a_{ij}^l \geq \epsilon_{str} \max_{a_{ik}^l < 0} |a_{ik}^l| .$$

All strong negative couplings of a variable i can be denoted by the set

$$S_i^l = \left\{ j \in N_i^l \middle| i \text{ strongly negatively coupled to } j \right\} .$$

Furthermore, we also need somewhat the transpose of this, thus the set of all variables j which are strongly coupled to i. It is

$$S_i^{l^\top} := \left\{ j \in \mathcal{D}^l \middle| i \in S_j^l \right\} .$$

According to [TS01, Section A.7.1], "an important objective [in coarsening] is to create C/F-splittings which are as uniform as possible with F-variables being surrounded by C-variables to interpolate from." We get better convergence if we have strong couplings from fine grid points to coarse grid points. In [TS01, Section A.7.1.1] the "preliminary C-point choice" algorithm from [RS] is taken as rough idea of a coarsening method. It repeats the choice of a coarse grid point with the definition of all neighboring points which are strongly coupled to the the coarse point as fine grid points until all points are chosen as fine or coarse grid points.

What is called the *standard coarsening* algorithm in the literature [TS01, Section A.7.1.1], is stated in Algorithm 7. This method also introduces sets of undecided variables U^l and importance measures λ_i^l. It results in a rather evenly distributed set of coarse grid points due to the indirectly imposed ordering by the weights λ_i^l.

Algorithm 7 Standard coarsening algorithm

```
Require: level $l$
 1: function AMGstandardCoarsening
 2:     $F^l := \emptyset, C^l := \emptyset, U^l := \mathcal{D}^l$
 3:     for $i \in U^l$ do
 4:         $\lambda_i^l := \left|S_i^{l\top} \cap U^l\right| + 2\left|S_i^{l\top} \cap F^l\right|$
 5:     while $\exists i$ s.th. $\lambda_i^l \neq 0$ do
 6:         find $i_{\max} := \operatorname{argmax}_i \lambda_i^l$
 7:         $C^l := C^l \cup \{i_{\max}\}$
 8:         $U^l := U^l \setminus \{i_{\max}\}$
 9:         for $j \in (S_i^{l\top} \cap U^l)$ do
10:             $F^l := F^l \cup \{j\}$
11:             $U^l := U^l \setminus \{j\}$
12:         for $i \in U^l$ do
13:             $\lambda_i := \left|S_i^{l\top} \cap U^l\right| + 2\left|S_i^{l\top} \cap F^l\right|$
14:     return $C^l, F^l$
```

Besides of standard coarsening, there exists also *aggressive coarsening*, which basically defines strong couplings based on paths of strongly coupled variables and thus allows to introduce a much sparser coarse point set. However, it also requires stronger smoothing. Overall, AMG with aggressive coarsening is a research topic by its own and shall not be discussed further.

Also, there is a large set of parallel algorithms to define the C/F splitting, cf. [Yan06, GMOS06]. By construction, most of them are designed for parallelism known from distributed memory CPU clusters. Thus they usually construct splittings on subdomains of a domain decomposition by the serial standard coarsening algorithm and try to connect these parts with some clever routine. However, most of these methods are not designed for the massive thread-parallelism known from GPUs. One exception is parallel maximal independent set (PMIS) [Yan06], which is e.g. used in GAMPACK. Since PMIS tends to have robustness issues, it usually needs stronger interpolation methods. As a results, in this thesis, the objective is to keep the very robust sequential standard coarsening on CPU and port all other parts to GPU.

8.2.2 Interpolation

While the coarsening defines a set of variables which will be used as coarse grid variables, the interpolation defines the relation between the different levels in AMG. As previously stated, the interpolation operator I_l^{l+1} transfers solutions on the coarse level to the next finer level. Since we use the transpose of the interpolation operator matrix as restriction matrix, it is enough to describe the interpolation operation and operator construction. This is again done closely following [TS01, Appendix A].

We start by introducing some additional notation with

$$C_i^l := C^l \cap N_i^l, \quad F_i^l := F^l \cap N_i^l, \quad \bar{C}_i^l := C^l \cap S_i^l, \quad \bar{F}_i^l := F^l \cap S_i^l\,.$$

Let us now have a look at some of the usual techniques used, to perform interpolation.

Direct interpolation

This type of interpolation uses only the strongly coupled coarse grid points to interpolate a given fine grid point. Thus, for each $i \in F^l$ we use the set of so-called *interpolatory variables* $P_i^l = \bar{C}_i^l$ and interpolate a given fine grid variable e_i^l by

$$e_i^l = \sum_{k \in P_i^l} w_{ik}^l e_k^l, \quad w_{ik}^l = -\alpha_i^l \frac{a_{ik}^l}{a_{ii}^l}, \quad \alpha_i^l = \frac{\sum_{j \in N_i^l} a_{ij}^l}{\sum_{k \in P_i^l} a_{ik}^l}\,.$$

We still assume here the system matrix to be an M-matrix. Therefore, we can neglect the handling of positive non-diagonal entries.

Standard interpolation

A much better convergence can be achieved by *standard interpolation.* It not only considers strongly connected coarse grid nodes but also includes strong connections between fine grid nodes. In order to describe this process, let us have a look at a fine grid point $i \in F^l$. The application of its corresponding matrix row to a vector $\boldsymbol{e}$ reads as

$$a_{ii}^l e_i^l + \sum_{j \in N_i^l} a_{ij}^l e_j^l\,.$$

To apply standard interpolation, we introduce a modified system matrix. There, we replace in those rows that are associated with a fine grid point $i \in F^l$, as above, the variables e_j with $j \in \bar{F}_i^l$, thus the strongly coupled fine grid points, as

$$e_j \longrightarrow - \sum_{k \in N_j^l} a_{jk}^l e_k^l / a_{jj}^l\,.$$

The newly generated matrix $\hat{A}_l$ has entries $\hat{a}_{ij}^l$ and possesses new neigborhood sets $\hat{N}_i^l$. By further setting for all $i \in F^l$ that $\hat{P}_i^l = \bar{C}_i^l \cup (\bigcup_{j \in \bar{F}_i^l} \bar{C}_j^l)$, we can define standard interpolation analogously to direct interpolation as

$$e_i^l = \sum_{k \in \hat{P}_i^l} \hat{a}_{ik}^l e_k^l, \quad \hat{w}_{ik}^l = -\hat{\alpha}_i^l \frac{\hat{a}_{ik}^l}{\hat{a}_{ii}^l}, \quad \hat{\alpha}_i^l = \frac{\sum_{j \in \hat{N}_i^l} a_{ij}^l}{\sum_{k \in \hat{P}_i^l} \hat{a}_{ik}^l}\,.$$

This basically means that we expand direct interpolation to include the neighborhood of the strongly connected fine grid points.

Jacobi interpolation

A way to improve the quality of an existing interpolation is the application of one or more Jacobi relaxation steps. Analogously to the standard interpolation, we increase the set of variables that interpolate a given fine grid variable. However, *Jacobi interpolation* does this in much broader sense. Let us assume to have a given system matrix $A_l^{(\mu-1)}$ to which we already applied $\mu - 1$ times the Jacobi interpolation. Then for all variables $i \in F^l$ we find the $j \in F_i^l$ and replace in the equation associated to the variable i

$$e_j^l \longrightarrow \sum_{k \in P_j^l} w_{jk}^{(\mu-1)} e_k^l \,,$$

resulting in a new system matrix $A_l^{(\mu)}$. Finally we replace the interpolatory variables P_i^l by $P_j^l := C_i^l \cup \left(\bigcup_{j \in F_i^l} P_j^l \right)$ and construct the interpolation as for standard interpolation. By increasing the interpolation quality with Jacobi relaxation steps, we however also run into complexity issues, which might be overcome with *truncation*.

Truncation

The more connections between variables are taken into account when constructing the interpolation operator, the better the interpolation. However, too many connections lead to rather densely populated interpolation operator matrices and thus to denser system matrices on coarser levels. This in turn might heavily affect the overall complexity of the multigrid method.

Therefore, it is often necessary to introduce a *truncation* of the interpolation. One thus drops all entries related to connections with a strength smaller than the largest entry scaled by a threshold ϵ_{tr}. The resulting entries finally have to be rescaled accordingly.

8.2.3 Standard smoothers

We already introduced smoothers $S_l \in \mathbb{R}^{N_l \times N_l}$ to be the crucial numerical tool on a given level of the algebraic multigrid method. For $P_l \approx A_l^{-1}$ one has the smoothing operator

$$S_l := (I_l - P_l A_l)$$

and the smoothing reads as

$$\bar{\boldsymbol{x}}^l := S_l \boldsymbol{x}^l + (I_l - S_l) A_l^{-1} \boldsymbol{b}^l \quad \Leftrightarrow \quad \bar{\boldsymbol{x}}^l := \boldsymbol{x}^l + P_l(\boldsymbol{b}^l - A_l \boldsymbol{x}^l) \,. \tag{8.4}$$

Jacobi smoother

For $P := D_l^{-1}$, $D_l = \operatorname{diag}(A_l)$ we get the Jacobi smoother $S_l^J := I_l - D_l^{-1} A_l$. For a given point i its application thus reads as

$$\bar{x}_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j \right) \,.$$

Due to its trivial nature it can be very easily applied in parallel, even on GPUs. Furthermore, we can introduce a relaxed Jacobi iteration with the parameter $\omega \in \mathbb{R}$ and $P := \omega D_l^{-1}$, thus

$S_l^{\omega J} := I_l - \omega D_l^{-1} A_l$ resulting in the iteration

$$\bar{x}_i = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j \right) + (1 - \omega) x_i \,.$$

By a proper choice of the relaxation parameter, the smoothing can be improved a lot.

Gauss-Seidel smoother

A stronger smoother than Jacobi is the Gauss-Seidel method with $P_l := L_l^{-1}$ and L_l^{-1} the lower triangular part of A_l including all diagonal entries. It results in the smoother $S_l^{GS} = I_l - L_l^{-1} A_l$ which has the iteration rule

$$\bar{x}_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} \bar{x}_j - \sum_{j=i+1}^{N} a_{ij} x_j \right) \,. \tag{8.5}$$

Again, we can introduce relaxation leading to a method called *successive overrelaxation*, cf. [Saa03, Section 4.1] with the iteration rule

$$\bar{x}_i = \frac{\omega}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} \bar{x}_j - \sum_{j=i+1}^{N} a_{ij} x_j \right) + (1 - \omega) x_i \,.$$

By iteration rule (8.5) we can already see that this smoother is purely sequential. A way to overcome this is to introduce a coloring [Luk12, Section 3.6.3] for the variables of the linear system, decoupling subsets of the variables which can be then treated in a parallel way. For more details on coloring for iterative linear solvers, see e.g. [Saa03, Section 12.4]. Note however, that colored Gauss-Seidel versions are not used in this thesis.

8.3 GPU-parallel implementation

The implementation of algebraic multigrid for a single GPU is based on the sparse linear algebra library CUSP [BG12] in version 0.4.0. This library provides a set of linear algebra primitives, iterative solvers and preconditioners for sparse matrices of different sparse matrix formats. *Compressed sparse row* (CSR) [BG09] is the matrix format, which is mainly used throughout the implementation of AMG on top of CUSP. The implemented code integrates withing the CUSP framework, thus the new Ruge-Stüben classical AMG can be used as preconditioner for the standard CUSP solvers. It can be furthermore applied as stand-alone solver.

Due to the available linear algebra primitives, the implementation of a V-cycle in CUSP is straight-forward. The simple structure of the smoother allows for an easy GPU parallelization, too, thus does not need to be further discussed. Applying the restriction and interpolation operator is also done with the existing framework by a sparse matrix-vector product. The construction of the coarse grid operator by the matrix triple product

$$A_{l-1} := I_l^{l-1} A_l I_{l-1}^l$$

is done with the matrix-matrix operations delivered with CUSP. Note that a more efficient code might be achieved by implementing a dedicated triple product for the well-known structure of these matrices. However this is not done for the sake of usability and simplicity of the resulting code. Finally, the (usually direct) solve on the coarsest level is achieved by a CUSP-based Jacobi-preconditioned CG solver iterated to an absolute residual norm of 10^{-20}. All other components, namely all different types of interpolation operator construction and the hybrid GPU splitting between coarse and fine grid nodes are discussed in the following. Note that the described algorithms heavily make use of graph / matrix traversals. For GPUs, this is an important research topic by its own, cf. [MGG12]. Though, the realized graph traversal implementations might not achieve the best possible performance.

8.3.1 Hybrid C/F splitting

The coarsening algorithm is the only part of the code which is only partially implemented on GPU. It follows Algorithm 7. Steps three and four of the algorithm compute the strong influence weights λ_i^l for the current level. The weights are computed on GPU. Here, the first necessary step is to pre-compute the maximum (negative) non-diagonal entry for each matrix row, thus, $\max_{a_{ik}^l<0} |a_{ik}^l|$. This can be efficiently done by parallelizing on GPU over each row and traversing the CSR matrix data structure. Using these maxima, it is possible to identify strongly coupled nodes in an algorithm for evaluating the λ_i^l. The weight evaluation algorithm again traverses the system matrix in parallel for each row on GPU. Adding another node to the strong influence weights, thus increasing one of the λ_i^l by one is done with atomic operations.

The remaining part of Algorithm 7 is implemented on CPU with the necessary copy operations between CPU and GPU. To achieve an optimal computational complexity, the lookup of the largest weight λ_i^l in step 6 of the algorithm is performed with a priority queue data structure [CLRS01, Section 6.5], which is added to the CUSP framework. Since e.g. the STL-based implementation of a priority queue does not allow to have a constant-complexity maximum-find and -removal operation (in fact that implementation has a logarithmic complexity in that case), the data structure is hand-implemented. The proposed implementation uses bucket sort [CLRS01, Section 8.4] as base algorithm and allows to achieve the designated constant complexity removal operation with a linear-complexity data structure setup time. Overall this keeps a linear complexity for the standard coarsening algorithm.

Finally, while skipped in Algorithm 7, it is necessary to identify the mapping between the newly found coarse grid points on the next grid level and their position in the current grid level. This is again done on GPU. Results are stored in a constant-complexity lookup table.

8.3.2 Interpolation operator construction

Interpolation operators are represented by a matrix. Therefore, it is necessary for all types of interpolation operators to construct new sparse matrices. Due to the inefficiency of dynamic memory allocation on GPUs, new sparse matrices have to be generated in two phases. The first phase mimics the matrix construction, only counting the number of (later) generated entries in each row. Afterwards, memory for the to be constructed matrix is allocated. Eventually, the second phase fills the allocated memory with matrix entries. In fact, this often requires twice the amount of compute operations. However, there is no obvious way to avoid this approach.

Direct interpolation

The GPU implementation of direct interpolation starts by precomputing the intermediate coefficients

$$\alpha_i^l = \frac{\sum_{j \in N_i^l} a_{ij}^l}{\sum_{k \in P_i^l} a_{ik}^l} .$$

Thereafter, the sparse interpolation matrix is build as sketched before. All traversal operations are parallelized over the matrix rows. Thus, each GPU thread starts traversing the system matrix from another row. The positions of the interpolation coefficients in the finally constructed matrix I_l^{l+1} are determined by the lookup table created at the end of the C/F splitting as outlined in Section 8.3.1.

Standard interpolation

Implementation of standard interpolation is more complicated. Here, the algorithm starts by finding the set of interpolatory variables $P_i^l = \bar{C}_i^l$ for direct interpolation. Since this is necessary for each row, a sparse matrix is constructed to store this information. It reuses the existing data structure of the sparse system matrix and replaces the array of non-zeros by boolean indicator values. These indicate whether a given neighbor node is part of the interpolatory set or not. Next, the original system matrix is modified such that strongly coupled fine grid points are expanded as

$$e_j \longrightarrow - \sum_{k \in N_j^l} a_{jk}^l e_k^l / a_{jj}^l .$$

Note, that this construction is rather compute intensive and might be replaced by a more sophisticate method in the future. It requires e.g. to implement a sparse matrix row addition. The new set $\hat{P}_i^l = \bar{C}_i^l \cup (\bigcup_{j \in \bar{F}_i^l} \bar{C}_j^l)$ of interpolatory variables is computed, as well. Finally the algorithm reproduces the direct interpolation implementation, which is possible since the interpolatory nodes are are known.

Jacobi interpolation and truncation

Jacobi-interpolation is implemented, too, to be able to approach some system matrices which struggle with direct or standard interpolation. It is possible to reformulate the Jacobi interpolation application in terms of products and sums of some specifically constructed matrices. This allows to express the algorithm mostly in terms of CUSP linear algebra operations.

Truncation is also implemented on GPU. This is usually done within the construction algorithm for interpolation matrices, to avoid building new sparse matrices. However, this algorithm is also available as stand-alone method.

8.4 Numerical results

In the following, a series of convergence results will be presented and discussed to analyse the quality and to assure the correctness of the implemented algebraic multigrid method. We first have to fix a set of parameters for the setup and the solve phase. The splitting of coarse and fine grid points is performed as presented in Section 8.2.1, thus standard coarsening is used. Both, direct and standard interpolation are investigated with the strength parameter $\epsilon_{str} = 0.25$ as usual in the literature [TS01, Appendix A]. In case of standard interpolation, truncation is applied with the standard parameter $\epsilon_{tr} = 0.2$, cf. Section 8.2.2. New coarser levels are generated until less than a fixed, problem-independent number of 100 equations remain for the new level. The smoother is a relaxed Jacobi iteration with relaxation parameter $\omega = 0.8$, which is applied twice as pre-smoother and twice as post-smoother. It is part of a standard V-cycle. To measure the convergence quality, the convergence factor

$$\rho = \left(\frac{\|\boldsymbol{r}_{i_{last}}\|}{\|\boldsymbol{r}_1\|}\right)^{1/i_{last}}$$

is computed, with r the residual and i_{last} the index of the iteration in which the method converges. AMG is either applied directly as solver or as preconditioner for a conjugate gradient method.

The first analysis is based on a standard two-dimensional Poisson problem with homogeneous Dirichlet boundary conditions,

$$\begin{aligned} -\Delta u &= f && \text{on } \mathcal{D}\,, \\ u &= 0 && \text{on } \partial\mathcal{D}\,. \end{aligned}$$

Its underlying domain is the unit square $\mathcal{D} = (0,1)^2$. Here, $u : \mathcal{D} \to \mathbb{R}$ is the solution and $f : \mathcal{D} \to \mathbb{R}$ is the right-hand side with

$$f(x,y) = 8\pi^2 \sin(2\pi x)\sin(2\pi y)\,,$$

such that the analytic solution

$$u(x,y) = \sin(2\pi x)\sin(2\pi y)$$

is known. This problem is discretized with a standard second-order five-point finite difference stencil. The iterative linear solver is said to have converged, if the initial residual is reduced by a factor of 10^{-12}. A random vector is taken as initial guess for the solution.

Figure 8.1 displays on the left-hand side the convergence factors achieved by the AMG method directly used as iterative solver. Problem size $N_{\mathcal{D}}^1$ is always defined as the number unknowns in each of the coordinate directions in the discretization, thus $N_{\mathcal{D}}^1 \sim \frac{1}{h}$, with h the mesh width. Keeping in mind that smaller convergence factors are better, direct interpolation performs here a bit better than standard interpolation. However, this result will not hold for more complex problems. Overall these are textbook convergence rates. Furthermore, the error of the discrete solution has been calculated to make sure that not only the residual converges, but the solution is actually the solution of the original problem. The results on the right-hand side of Figure 8.1 thus show the expected second order convergence with errors evaluated by the discrete maximum norm. Note that direct and standard interpolation result in almost the same result, therefore the line plotted for direct interpolation in the results figure is hardly visible.

Next, we have a series of rather generic convergence studies for two- and three-dimensional

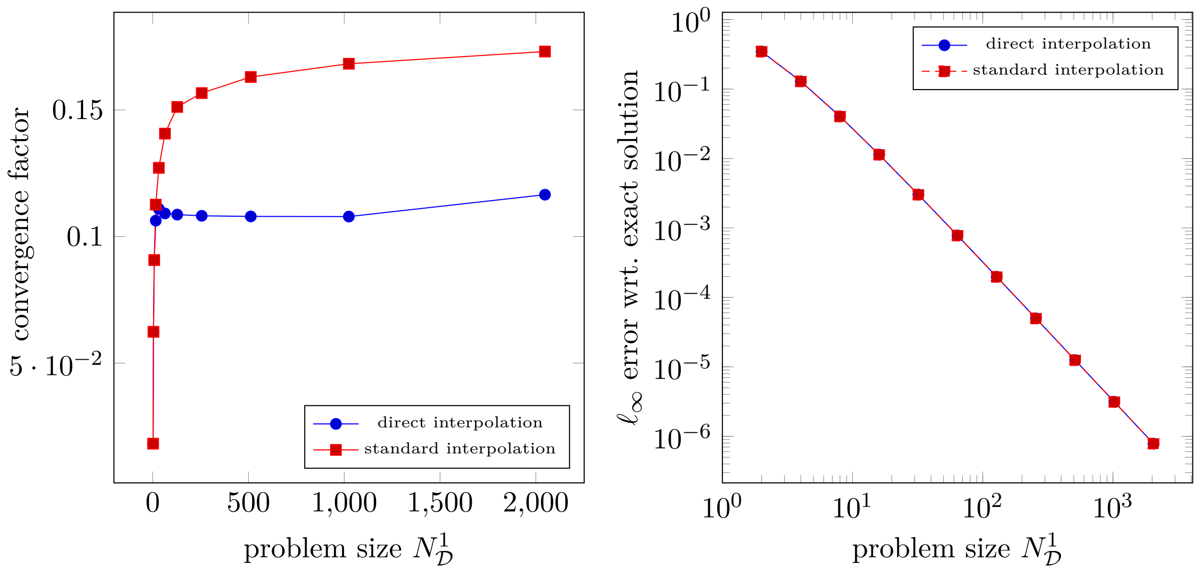


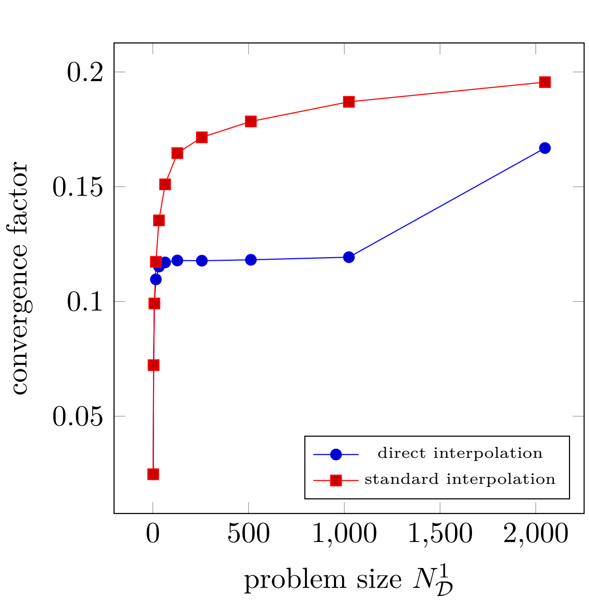
Figure 8.1: Convergence factors (left) and PDE solution error (right) for a 2D Poisson problem with right-hand side given for a known analytic solution, solved with AMG as solver.

Poisson problems with homogeneous Dirichlet boundary, which are discretized by second-order finite differences on a uniform grid. In two dimensions, the standard five point stencil is applied, while in three dimensions the standard seven point finite difference stencil will be used. The right-hand side of the linear system is a zero-component vector. This is done, in order to discuss the same model problems as in the next section. Also, convergence is said to be achieved if the absolute residual reaches a threshold of 10^{-15}.

Figure 8.2 gives results for the two-dimensional test case. The plots on the left-hand side present convergence factors for growing problem sizes with AMG as a pure solver, cf. Algorithm 6. For the smaller problem sizes, direct interpolation again outperforms standard interpolation, however, for growing problem sizes the convergence factor will probably not be asymptotically constant. On the other hand, standard interpolation has a more robust convergence behavior, which is what is expected. The results on the right-hand side of the same figure outline convergence for AMG as preconditioner for a CG method. Although the behavior of both interpolation schemes is identical, a considerably smaller convergence factor, thus a smaller numer of iterations, is achieved.

This is why the test case for a three-dimensional Poisson problem is only discussed for the preconditioned CG case. These results are presented on the left-hand side of Figure 8.3. Here, the strength of standard interpolation becomes evident. Direct interpolation is no longer able to achieve the asymtotically constant convergence factor while standard interpolation achieves the expected rates. Due to the additional amount of memory required on a GPU to set up standard interpolation, it was not possible to perform a test run with 128^3 unknowns.

We conclude this section by a very tough numerical test case which is driven by our application of two-phase flows. In Section 5.1.2 we introduced the three-dimensional Poisson problem for pressure correction in Chorin's projection approach to solve the two-phase Navier-Stokes



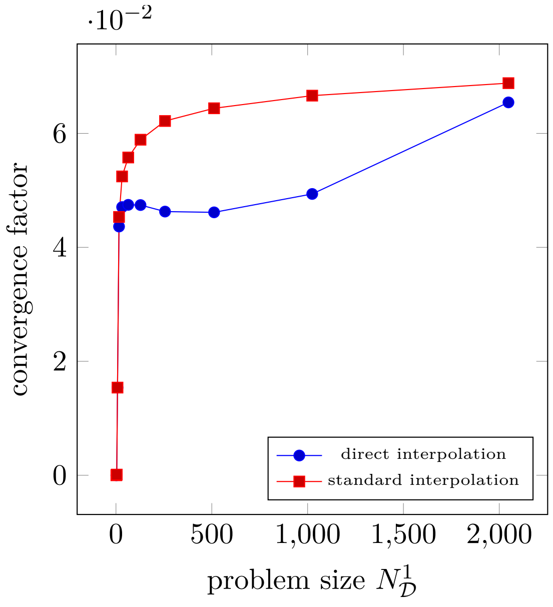
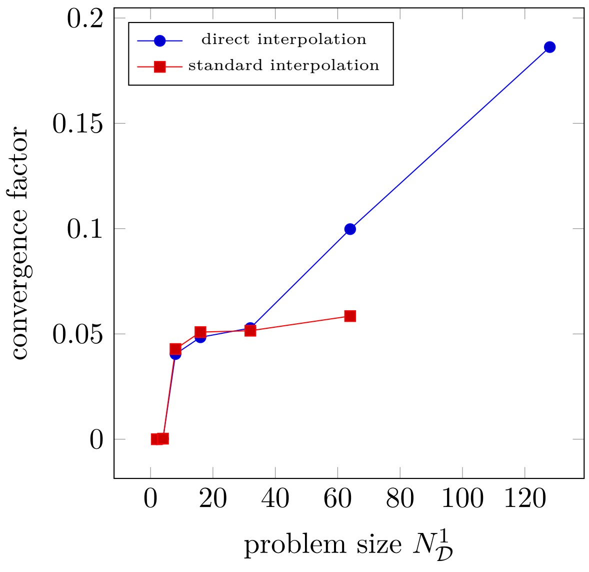


Figure 8.2: Convergence factors for standard 2D Poisson problem solved with AMG as solver (left) and AMG as preconditioner for a CG solver (right).



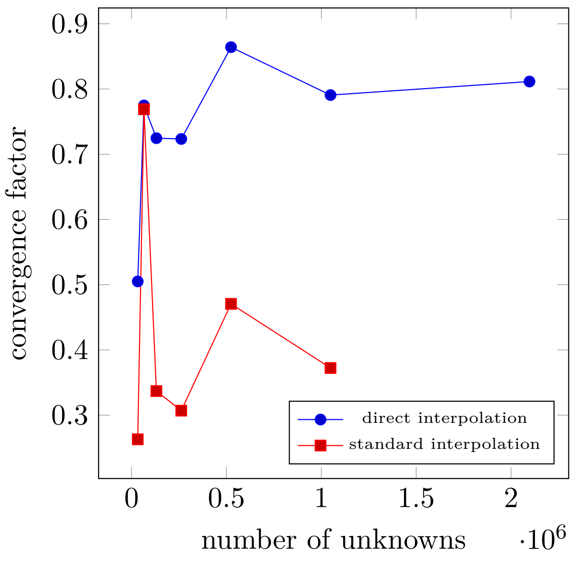


Figure 8.3: Convergence factors for a standard 3D Poisson problem solved with AMG as preconditioner for a CG solver (left) and for a non-constant coefficient Poisson problem from two-phase flow simulation with AMG as solver (right).

equations. The resulting linear system after discretization is badly conditioned in case of a high jump in the density between the two involved fluid phases. This test case has been extracted from a simulation of a large rising gas bubble with a density of $\rho_g = 1.0$ and a viscosity of $\mu_g = 0.1$ in a liquid with density $\rho_l = 1000.0$ and viscosity $\mu_l = 10$ contained in a box of size $1m \times 1m \times 1m$. The bubble is initially of spherical shape with a diameter of $0.5m$ and is centered in the box, which has homogeneous Dirichlet boundary conditions for the velocity field in normal and tangential direction of all boundaries. Standard gravity is applied as volume force. The linear system is finally extracted from the first time step with the right-hand as in the solver. As initial guess, a random vector is used. Like in the first test case, the norm of the initial residual is reduced by a factor of 10^{-12} to achieve convergence.

The results, presented on the right-hand side of Figure 8.3 reflect convergence factors for AMG as solver and discretization grid sizes $32 \times 32 \times 32$, $64 \times 32 \times 32$, $64 \times 64 \times 32$, $64 \times 64 \times 64$, $128 \times 64 \times 64$ and $128 \times 128 \times 64$. Besides of the very large density coefficient jump, this choice of mesh sizes imposes an anisotropic discretization leading to degraded convergence. The achieved convergence factors underline the complexity of the problem. Direct interpolation fails. Some of the problems even did not converge within 100 iterations. In case of standard interpolation, convergence is always achieved, but the rates are still rather bad. One reason for this is for sure the weak smoother. In standard CPU implementations Gauss-Seidel-type smoothers are expected to perform better. Also, we could achive better convergence by increasing the number of smoother sweeps. This is not done here, to keep the results comparable with all other results. As in the previous three dimensional test case, standard interpolation can also not be applied in the 128^3 problem size due to memory limitations.

8.5 Performance results

In this section, some of the previously introduced test cases are used to measure the performance of the implemented method. Even though the hybrid GPU implementation has been constructed starting from a full CPU implementation, the intention here is to compare the hybrid GPU results not with that potentially low-performance pure CPU implementation, but with a standard open-source CPU AMG library, namely BoomerAMG [HY02], which is contained in the solver package Hypre [FY02]. Overall, two CPU AMG configurations will be considered. The first is based on the (at time of writing this thesis) latest available state-of-the-art version of BoomerAMG, which is contained in *hypre-2.9.0b*. The second study tries to give a relative comparison to GAMPACK [ENS12]. Since GAMPACK is not publicly available, we do not compare results directly. However, we contrast performance results of our hybrid GPU implementation with the CPU comparison base presented in performance results [ENS12] for GAMPACK. This comparison base is *hypre-2.8.0b* with specific parameters presented later.

8.5.1 Benchmark setup

The GPU benchmark system consists of a quad-core Intel Xeon E5620 2.40 GHz CPU with 12 GB DDR3 ECC RAM and an Nvidia Tesla M2090 GPU. As CPU system, a dual six-core Intel Xeon X5650 2.667 GHz machine with 24 GB RAM is used. The two six-core CPUs together were almost similar priced as the GPU when they arrived on the market. Therefore it should be balanced to compare the performance of AMG on the one GPU with its performance on these 12 CPU cores. Furthermore, the C/F splitting of the hybrid GPU implementation will run a little bit slower on the GPU benchmark machine due to its slower CPU. On both systems, Ubuntu Linux 12.04 (64 bit) is installed as operating system.

The hybrid GPU code is compiled with `nvcc` from the CUDA 5.5 toolkit and GCC 4.6.3 with optimization parameters `-arch sm_20 -O3`. *hypre 2.9.0b* is compiled by the same GCC compiler with the optimization flag `-O3` and OpenMPI 1.3 to perform a multi-core parallelization. OpenMP is not used. Moreover, the GAMPACK-related comparison uses a purely OpenMP parallelized *hypre-2.8.0b*. Hypre expects to use the Intel compiler in the OpenMP mode, for which version 13.0.1 was applied. Due to some problems with the configuration script, only an optimization with `-O2` is possible in this case.

All timings on CPU and GPU are wall clock times. The GPU AMG implementation is not intended to be an accelerated part of an existing CPU code but a solver within a full GPU code. This is why, in the hybrid GPU case, it is expected, that the system matrix, righ-hand side and initial guess are already availably in GPU memory. However, all transfer times between CPU and GPU memory as well as the full CPU and GPU compute time are included for the hybrid GPU setup/solver test runs.

To perform time measurements with BoomerAMG / *hypre-2.9.0b* on CPU, numerical parameters were choosen different to the GPU implementation, since e.g. standard interpolation with a Jacobi smoother is clearly not the most performant way to apply AMG on CPUs. Instead, the test application `new_ij` delivered with the Hypre framework is used as test base with the same convergence criterion as in the GPU case. All other parameters are left as the default proposed by that application, in the hope to give the best possible parameter set for the CPU benchmark counterpart, at least for Poisson-like problems. BoomerAMG timings are reported as given by `new_ij`. The numerical parameters of *hypre-2.8.0b*, in the GAMPACK-related benchmark, are chosen as close as possible to the parameters proposed in [ENS12]. Thus, standard interpolation and C/F splitting by PMIS are selected as parameters. All other parameter choices (including the smoother) are left to the default of `new_ij`.

8.5.2 Two-dimensional Poisson problem

The performance discussion starts with the standard two-dimensional Poisson problem as used for the convergence results in Figure 8.2 from the last section. In Figure 8.4 on the left-hand side, the runtimes for the hybrid GPU/CPU setup phase are presented. As expected, direct interpolation is faster due to its simpler structure. In the solve phase, cf. Figure 8.4 on the right-hand side, the overall runtime for both direct interpolation and standard interpolation is very similar, even though convergence rates for direct interpolation were slightly better, in this case.

Figure 8.5 gives the performance results for the latest BoomerAMG version. Note that it was not possible to get the test application `new_ij` running on all the 12 CPU cores of the CPU test machine for the MPI-parallelized Hypre version. Here, eight cores, were the maximum. Increasing the number of applied CPU cores improves performance. However, a clear strong scaling is not available. This becomes even more evident for the performance comparison baseline with regard to GAMPACK [ENS12]. When using the older Hypre/BoomerAMG version with different C/F splitting and OpenMP parallelization, cf. Figure 8.6, speed-ups stagnate starting with eight cores.

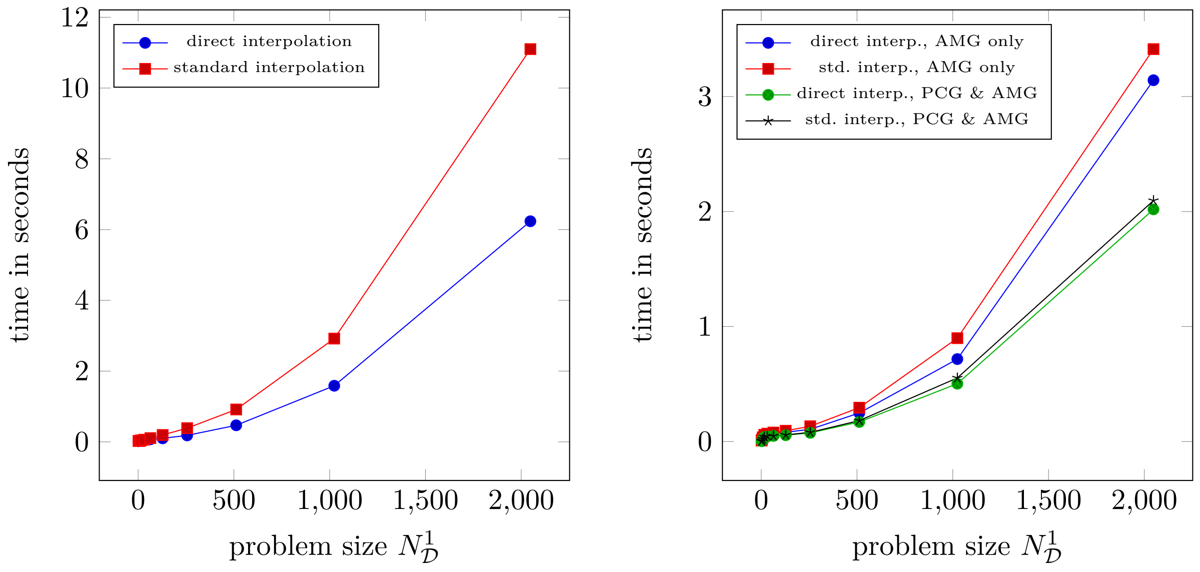


Figure 8.4: Timings of the full AMG setup phase (left) and solve phase (right) for a 2D Poisson problem (including CPU and GPU computations).

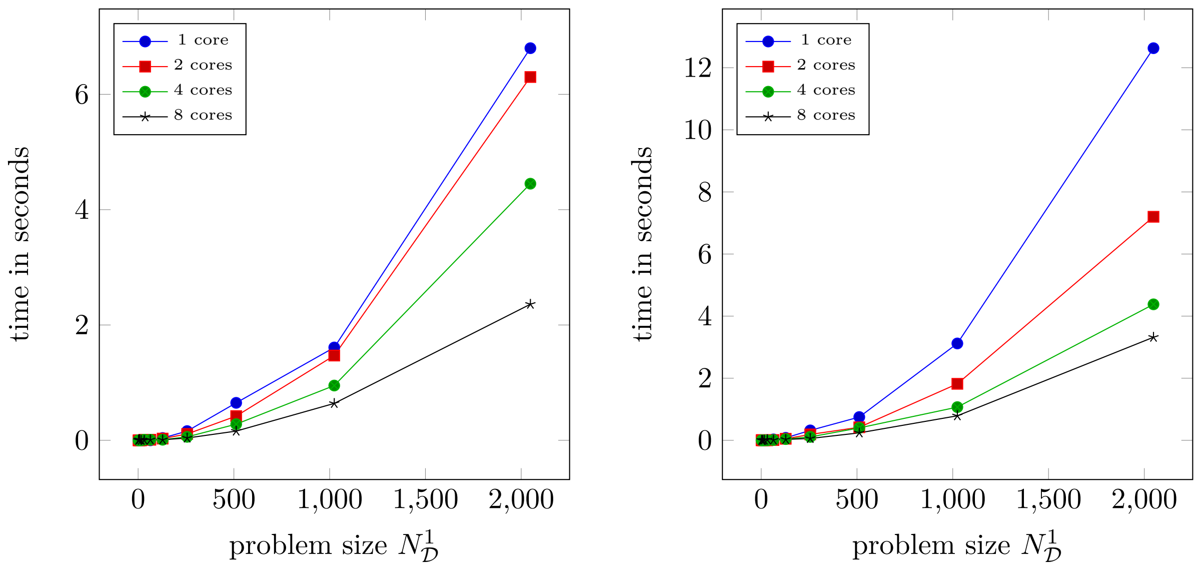
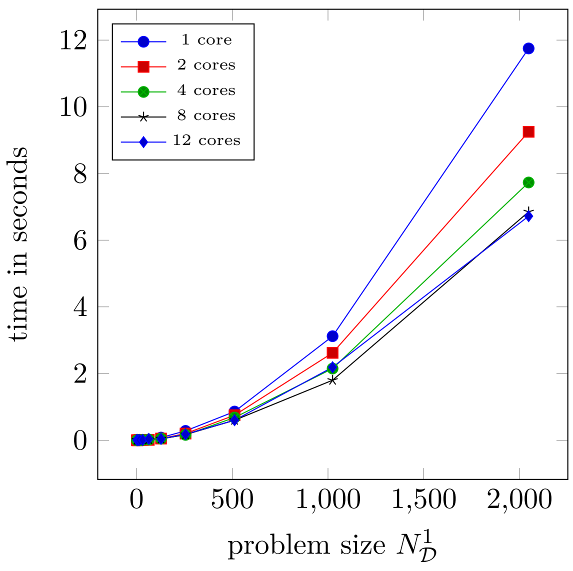


Figure 8.5: Time required in the setup phase (left) and the solve phase with AMG as solver (right) for the 2D Poisson problem computed with the BoomerAMG/Hypre CPU library (at different CPU core counts).



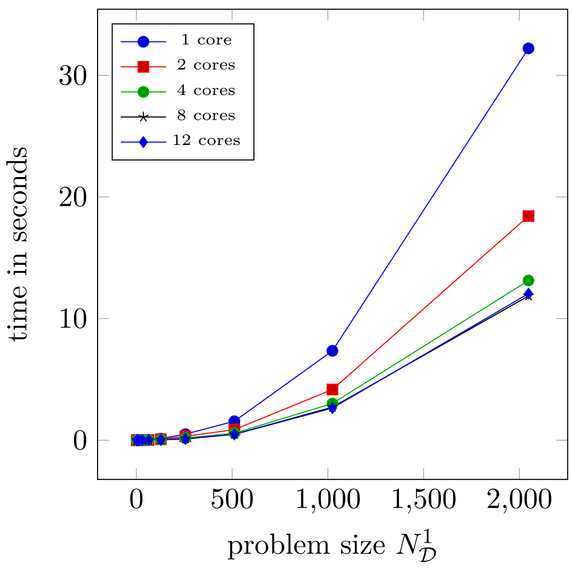


Figure 8.6: Time required in the setup phase (left) and the solve phase with AMG as solver (right) for the 2D Poisson problem computed with the BoomerAMG/Hypre CPU library (at different CPU core counts) with settings similar to those used in [ENS12].

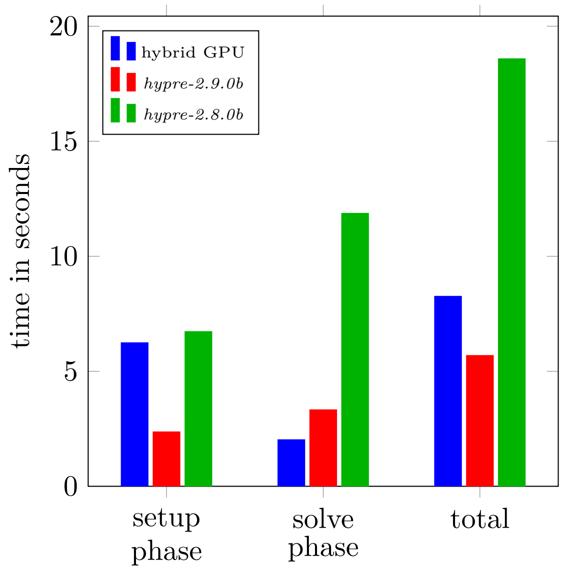
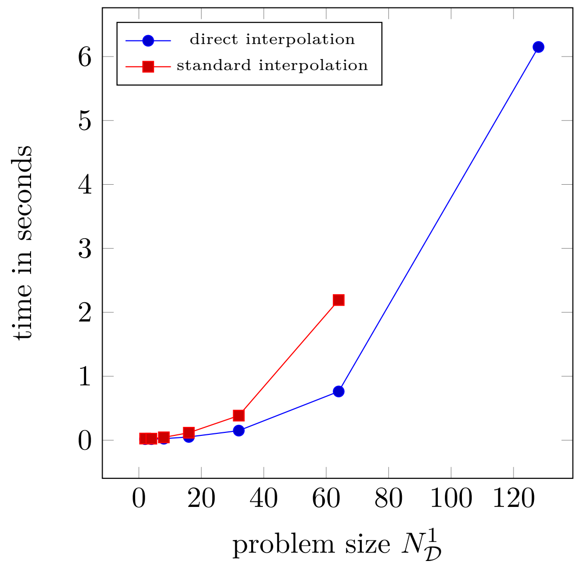


Figure 8.7: Direct runtime comparison for the solution of a two-dimensional Poisson problem with the new hybrid GPU implementation and two different CPU test cases.



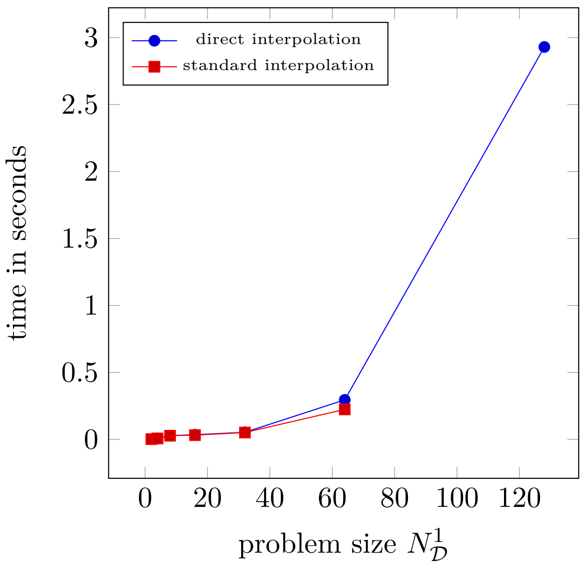


Figure 8.8: Time required in the full setup phase (left) and the solve phase with AMG as preconditioner for a CG solver (right) for the 3D Poisson problem

We finally compare all three test cases for the two-dimensional Poisson problem in Figure 8.7, taking the best available results in each case. For the setup phase, the hybrid GPU approach has similar performance as the GAMPACK-related test case. However, the latest BoomerAMG version is faster. In the solve phase, we observe tht the GPU code is always faster. This becomes rather significant for *hypre-2.8.0b* in the discussed configuration, with more than a factor of three of speedup on almost equally priced hardware. In terms of total solution time, the proposed implementation clearly outperforms *hypre-2.8.0b* and is only a bit slower than the latest BoomerAMG version, comparing one GPU with a 12-core CPU system for the two-dimensional Poisson problem.

8.5.3 Three-dimensional Poisson problem

Next, we discuss performance measurements for the three-dimensional Poisson problem test case, which was earlier analyzed for convergence on the left-hand side of Figure 8.3. Results for the hybrid GPU implementation are outlined in Figure 8.8. The 128^3 test case is only computable with direct interpolation, due to memory limitations. As in the two-dimensional case, the setup phase takes longer for standard interpolation, while the solve phase has a very similar runtime for both interpolation techniques.

Figures 8.9 and 8.10 collect runtimes for the solution by BoomerAMG in *hypre-2.9.0b* and *hypre-2.8.0b*, respectively. Similar parallel scalability is observed as in the two-dimensional case. A comparison of all three test cases is given in Figure 8.11. As before, the lowest runtimes are collected for each method. We here focus on runtimes for 64^3 unknowns. In the setup phase, the hybrid GPU AMG implementation is 60% faster than the state-of-the-art CPU AMG running on a 12-core system. This is a decent result. Furthermore, it is an impressive factor of 2.5 faster than the GAMPACK-related *hypre-2.8.0b* AMG. For the solve phase, we observe a 20%

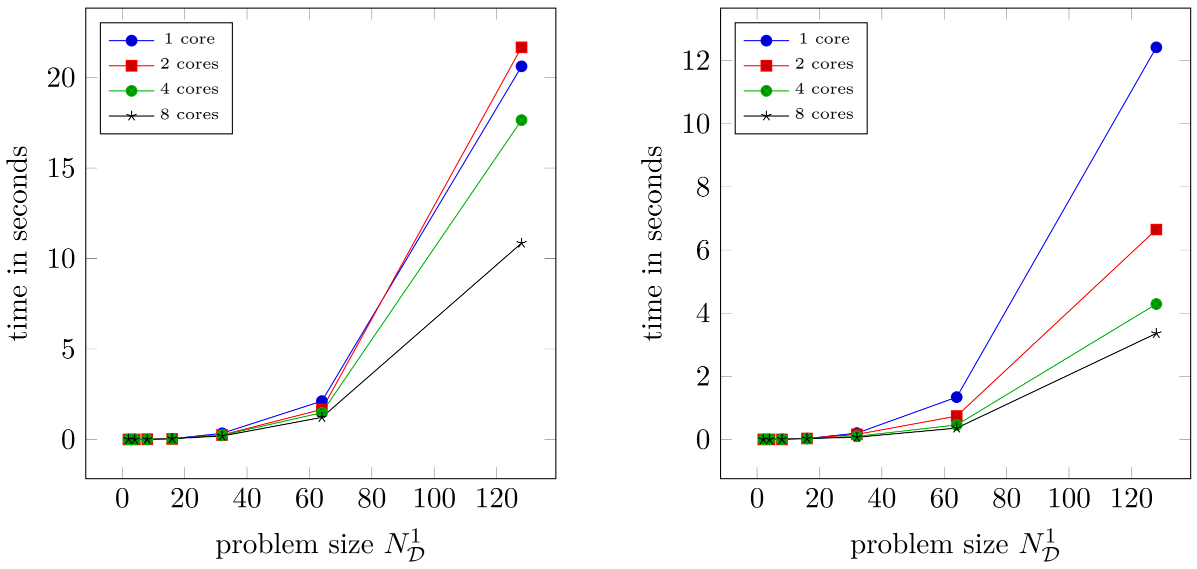


Figure 8.9: Time required in the setup phase (left) and the solve phase with AMG as solver (right) for the 3D Poisson problem computed with the BoomerAMG/Hypre CPU library (at different CPU core counts).

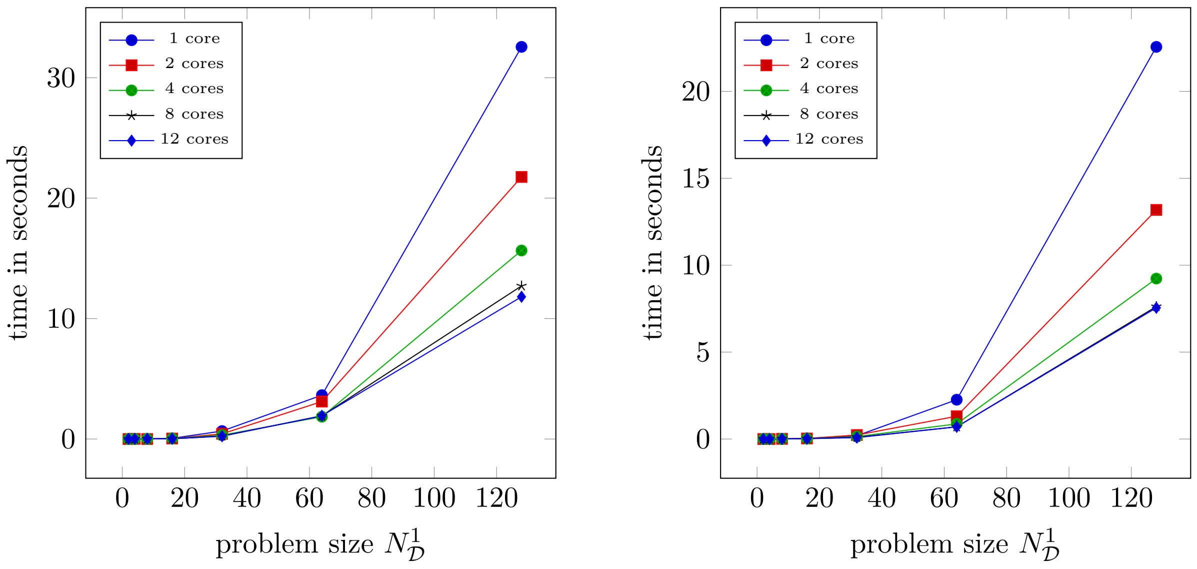


Figure 8.10: Time required in the setup phase (left) and the solve phase with AMG as solver (right) for the 3D Poisson problem computed with the BoomerAMG/Hypre CPU library (at different CPU core counts) with settings similar to those used in [ENS12].

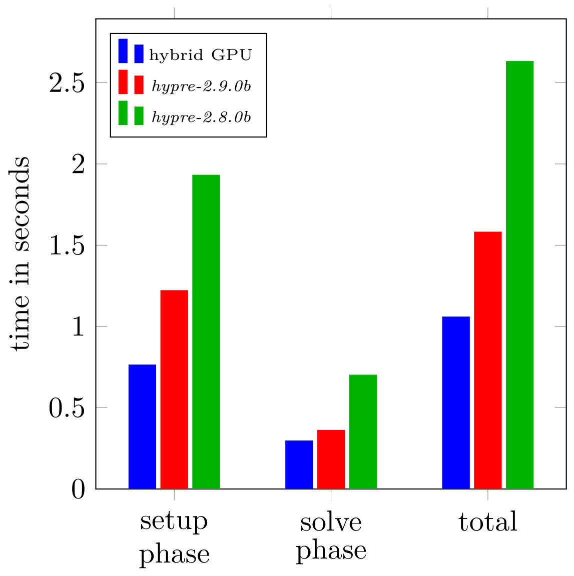


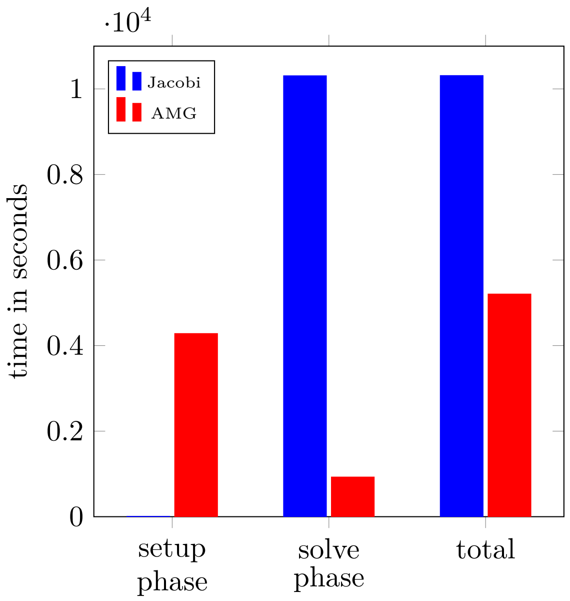
Figure 8.11: The hybrid GPU AMG implementation clearly outperforms both CPU AMG implementations on similar priced hardware, in case of the three-dimensional Poisson problem with a grid of 64^3 points.

performance improvement over *hypre-2.9.0b* and a speedup by a factor of roughly 2.4 over *hypre-2.8.0b*. In total, the full solution process bt the hybrid AMG implementation is about 50% faster than the state-of-the-art AMG and about 2.5 times faster than *hypre-2.8.0b* with PMIS for a three-dimensional Poisson problem. This is a rather impressive result, remembering that these measurements compare almost equally priced hardware.

Summarizing the results presented in this section, it becomes clear that large performance gains for AMG implementations on GPU require a considerable effort in optimization. As already mentioned in Section 8.3, it was not intended overoptimize the constructed hybrid GPU AMG implementation. Instead, a robust AMG implementation, with speedup on GPUs, even with respect to almost equally priced hardware, is constructed. The implemented code is always clearly faster than *hypre-2.8.0* with C/F splitting by PMIS and is in total 50% faster than the state-of-the-art version of BoomerAMG for the three-dimensional Poisson problem.

8.6 Application to random PDE problems

This chapter shall be finished by outlining that the application of the implemented algebraic multigrid method to random PDE problems gives not only an asymptotic complexity improvement, but results in an actual reduction in runtime. To show this, the two elliptic random PDE problems from Section 3.2 are solved again. This time, however, the implemented algebraic multigrid with standard coarsening and the already introduced default parameters is used as preconditioner for the conjugate gradient solver. As described in Section 6.1.1 for the Jacobi-preconditioned case, the iterative solver is stopped at an absolute residual norm of 10^{-15}. All other parameters are also identical to that section. The number of collocation points



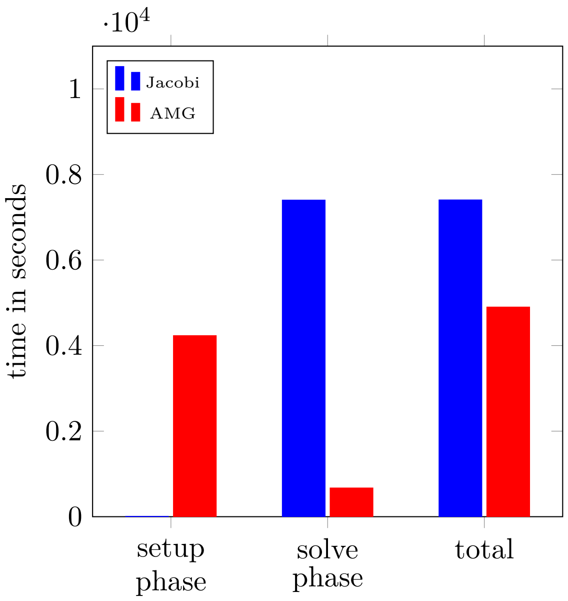


Figure 8.12: Comparison between solution times on a GPU with the Jacobi- and AMG-preconditioned conjugate gradient solver to solve the piecewise constant (left) and the Karhunen-Loève based (right) random coefficient elliptic problems for 2^{12} stochastic realizations.

is $N_\Gamma = 2^{12}$. Furthermore, the Karhunen-Loève based random coefficient elliptic problem is solved for a stochastic dimension of $N_{KL} = 5$.

Figure 8.12 outlines the results of this runtime comparison study, which runs fully on a GPU. On the left-hand side, results for the elliptic problem with piecewise constant coefficients are presented. The given timings represent the accumulated runtimes over all 2^{12} linear problems for the setup phase, the solution phase and the sum of both. Even though the time invested in the setup phase for AMG is quite significant, this pays off in the solve phase. Overall, the AMG-preconditioned CG solver is about a factor of two faster (including setup). This is a rather impressive result which would be even more pronounced for discretizations with more than $N_\mathcal{D} = 512^2$ grid points. On the right-hand side of Figure 8.12, the timings for the five-dimensional Karhunen-Loève based elliptic problem are given. Here, using AMG still gives a decent performance improvement of about 50 percent. Note that recycling the AMG setup phase for many stochastic realizations might even give a much higher performance improvement. However, this shall not be investigated further. Altogether, the presented numerical and runtime results underline that using AMG does not just allow to have a complexity of $O(N_\Gamma N_t N_\mathcal{D})$ or $O(N_\Gamma N_\mathcal{D})$ for the PDE solution part of the stochastic collocation method. It even allows to achieve strong performance improvements.

Part III

Approximation and empirical analysis of fast decaying random PDEs

9 A-posteriori Karhunen-Loève covariance spectrum decay analysis

It is often assumed that random PDEs map a fast decay in the covariance spectrum of the random *input* parameter field $\boldsymbol{a}(\boldsymbol{y}, \boldsymbol{x})$ to a fast decaying spectrum in the covariance $\operatorname{Cov}[\boldsymbol{u}](\boldsymbol{x}, \boldsymbol{x}')$ of the random PDE *solution*. This allows to apply numerical methods such as anisotropic sparse grid stochastic collocation with improved convergence. Even though this relationship might be obvious for simpler PDEs with stochastic coefficients, the covariance spectrum behavior of the output of a non-linear PDE with many random input parameters might be far away from this assumption. On example for such a PDE are the two-phase Navier-Stokes equations. It is therefore of high interest to perform such a spectrum analysis by numerical means. This can be achieved by a Karhunen-Loève decomposition of the solution field $\boldsymbol{u}(\boldsymbol{y}, \boldsymbol{x})$, which will be called *a-posteriori Karhunen-Loève decomposition*, in the following. Note here the difference to the classic application of the Karhunen-Loève expansion in stochastic collocation to approximate a given random *input* parameter field. We thus want to approximate a Karhunen-Loève expansion of a numerically approximated random *solution* field.

In the context of reduced order models we know from e.g. [Loe78, GH14, ST06] that a Karhunen-Loève expansion truncated after the N_{KL}th term delivers a best N_{KL}-term approximation in L^2-sense. Thus, by approximating an a-posteriori Karhunen-Loève expansion, we can also construct a reduced order model with fast or even exponential convergence. This is of high interest for those problems which require a high amount of computational work for a single evaluation, as in the case of the Navier-Stokes equations.

While the theoretical properties of the Karhunen-Loève expansion have been studied in a profound way, the effective evaluation for large-scale three-dimensional PDE problems with potentially millions of unknowns is still a very computationally expensive problem. Here, it is important to know that the a-posteriori Karhunen-Loève expansion requires an approximation of the eigenvalue decomposition of an extremely large dense matrix after discretization. This problem will be tackled by the use of an iterative Lanczos eigenvalue solver, cf. [Saa03], which approximates the most important eigenvalues and eigenvectors in a few iterations. The whole method is implemented to run on an HPC cluster equipped with GPUs to overcome the high computational costs. Note that e.g. [ST06] proposes an alternative efficient Karhunen-Loève approximation approach, using a fast multipole expansion.

This chapter starts by the discretization of the underlying continuous eigenvalue problem of a Fredholm integral equation of second kind by the Nyström method. Then, the Lanczos iterative method is reviewed. After a few remarks on the multi-GPU parallel implementation, numerical results with performance measurements for different random (two-phase) Navier-Stokes problems are given.

9.1 Discretization by the Nyström method

We can formulate the truncated (a-posteriori) Karhunen-Loève decomposition for random PDE solutions as

$$u(\boldsymbol{y},\boldsymbol{x}) \approx \mathbb{E}\left[u\right](\boldsymbol{x}) + \sum_{k=1}^{N_{KL}^u} \sqrt{\lambda_k} Y_k(\boldsymbol{y}) \psi_k(\boldsymbol{x}) \tag{9.1}$$

with

$$Y_k(\boldsymbol{y}) := \frac{1}{\sqrt{\lambda_k}} \int_{\mathcal{D}} \left(u(\boldsymbol{y},\boldsymbol{x}) - \mathbb{E}\left[u\right](\boldsymbol{x})\right) \psi_k(\boldsymbol{x}) d\boldsymbol{x}\,,$$

cf. Section 2.4.1. Note that N_{KL}^u is the truncation parameter for the output which is different to the input truncation N_{KL}. To be concise, time-dependence is omitted here, even though the application problems will have a time component. However, in terms of a snap-shotting technique, time is assumed to be a constant parameter. Moreover, the construction is outlined for scalar-valued solutions $u(\boldsymbol{y},\boldsymbol{x})$ only. From the literature (e.g. [LMK10, ST06]) we know that ψ_k are the eigenfunctions and λ_k the eigenvalues of the operator

$$C_{\mathrm{Cov}[u]} : L^2(\mathcal{D}) \to L^2(\mathcal{D})\,, \quad f \mapsto \int_{\mathcal{D}} \mathrm{Cov}\left[u\right](\cdot,\boldsymbol{x}) f(\boldsymbol{x}) d\boldsymbol{x}\,.$$

These can be found by solving the associated eigenvalue problem of a Fredholm integral equation of second kind with

$$\int_{\mathcal{D}} \mathrm{Cov}\left[u\right](\boldsymbol{x},\boldsymbol{x}') \psi_k(\boldsymbol{x}') d\boldsymbol{x}' = \lambda_k \psi_k(\boldsymbol{x})\,. \tag{9.2}$$

We will now follow the lines of [Hac95] and apply the *Nyström method* to discretize the integral equation. While some literature on the Karhunen-Loève decomposition (e.g. [GH14]) uses Galerkin projection for approximation, we choose the Nyström method since it fits well into our finite difference / finite volume framework with functions discretized by point evaluations.

The basic idea of the Nyström method is to approximate the integral in (9.2) by a quadrature rule

$$Q_{N_{NY}}(f) = \sum_{j=1}^{N_{NY}} w_j f(\boldsymbol{q}_j)\,,$$

with $\boldsymbol{q}_j \in \mathcal{D}$ the quadrature points and $w_j \in \mathbb{R}$ quadrature weights. Using this idea, we derive a semi-discrete formulation of the integral equation (9.2) as

$$\sum_{j=1}^{N_{NY}} w_j \mathrm{Cov}\left[u\right](\boldsymbol{x},\boldsymbol{q}_j) \psi_k(\boldsymbol{q}_j) = \lambda_k \psi_k(\boldsymbol{x}) \qquad \forall \boldsymbol{x} \in \mathcal{D}\,.$$

To derive a fully discrete problem, the $\boldsymbol{x} \in \mathcal{D}$ are also restricted to the same quadrature points $\boldsymbol{q}_j$. We thus get

$$\sum_{j=1}^{N_{NY}} w_j \mathrm{Cov}\left[u\right](\boldsymbol{q}_i,\boldsymbol{q}_j) \psi_k(\boldsymbol{q}_j) = \lambda_k \psi_k(\boldsymbol{q}_i) \qquad i = 1,\dots,N_{NY}\,, \tag{9.3}$$

which is our fully discrete problem. To shorthand the notation, we now set $\psi_{i,k} := \psi_k(\boldsymbol{q}_i)$ and $\beta_{i,j} := w_j \mathrm{Cov}\,[u]\,(\boldsymbol{q}_i, \boldsymbol{q}_j)$ to get

$$\boldsymbol{\psi}_k^{N_{NY}} := \begin{pmatrix} \psi_{1,k} \\ \vdots \\ \psi_{N_{NY},k} \end{pmatrix} \quad \text{and} \quad B := \begin{pmatrix} \beta_{1,1} & \vdots & \beta_{1,N_{NY}} \\ \vdots & \ddots & \vdots \\ \beta_{N_{NY},1} & \vdots & \beta_{N_{NY},N_{NY}} \end{pmatrix},$$

which gives us a short formulation of the discrete equation system (9.3) as

$$(\lambda_k I - B)\,\boldsymbol{\psi}_k^{N_{NY}} = 0\,. \tag{9.4}$$

To evaluate the originally continuous eigenfunctions $\psi_k(\boldsymbol{x})$, $\forall x \in \mathcal{D}$, the *Nyström interpolation*

$$\psi_k(\boldsymbol{x}) = \frac{1}{\lambda_k}\left(\sum_{j=1}^{N_{NY}} w_j \mathrm{Cov}\,[u]\,(\boldsymbol{x}, \boldsymbol{q}_j)\psi_{j,k}\right)$$

can be used. In the following, the quadrature rule $Q_{N_{NY}}$ is restricted to constant-weight rules with weights $w_j = \frac{1}{N_{NY}}$. This choice transforms the short-hand equation (9.4) of the discrete system to

$$\left(\tilde{\lambda}_k I - C\right)\boldsymbol{\psi}_k^{N_{NY}} = 0\,. \tag{9.5}$$

The matrix $C \in \mathbb{R}^{N_{NY}} \times \mathbb{R}^{N_{NY}}$ is the simple structured discrete covariance matrix with entries $c_{i,j} = \mathrm{Cov}\,[u]\,(\boldsymbol{q}_i, \boldsymbol{q}_j)$ evaluated at the quadrature points and $\tilde{\lambda}_k = N_{NY}\lambda_k$. Note here that the introduction of more complex quadrature rules might lead to methods with higher convergence rates. However, this is future work.

In summary, the solution of the eigenvalue problem for the Fredholm integral equation (9.2) is done here by computing the eigenvalues of the discrete covariance matrix with point sampling at quadrature points. The eigenvalues λ_k are eventually given as $\lambda_k = \frac{1}{N_{NY}}\tilde{\lambda}_k$. Related theory for the discussed method is found in [RBDV10]. Note that we will later-on identify the abscissas $\boldsymbol{q}_i$ with all (finite difference) grid-points of the discrete solution of the two-phase Navier-Stokes equations. This leads to a quadratic (dense) matrix C with at least tens to hundreds of thousands of rows.

9.2 Lanczos method for eigenvalue decomposition

Now that we have reduced the integral equation eigenvalue problem (9.2) to the task of finding the eigenvalues and eigenvectors of the discretely sampled symmetric positive definite covariance matrix C, it is necessary to apply an efficient eigenvalue solver.

The *Lanczos* method will be used to approximate the most important eigenvalues and eigenvectors of the covariance matrix. Lanczos' algorithm is a Krylov subspace method which specializes the Arnoldi iterative method to hermitian matrices. It iteratively constructs a tridiagonal matrix with a subset of the eigenvalues of the original matrix. Usually, tridiagonal QR eigenvalue solvers are used to extract all eigenvalues of the reduced matrix in a very efficient way.

Algorithm 8 Lanczos iterative method

```
Require: $C$ symmetric positive definite
1: function LANCZOS($C$, $\ell$)
2:     $\beta_0 = 0$, $\boldsymbol{k}_0 = 0$, $\boldsymbol{b} =$ arbitrary, $\boldsymbol{k_1} = \frac{\boldsymbol{b}}{||\boldsymbol{b}||}$
3:     for $n = 1, 2, \ldots, \ell$ do
4:         $\boldsymbol{v} = C\boldsymbol{k}_n$
5:         $\alpha_n = \boldsymbol{k}_n^T v$
6:         $\boldsymbol{v} = \boldsymbol{v} - \beta_{n-1}\boldsymbol{k}_{n-1} - \alpha_n \boldsymbol{k}_n$
7:         $\beta_n = ||\boldsymbol{v}||$
8:         $\boldsymbol{k}_{n+1} = \boldsymbol{v}/\beta_n$
9:     return $\{\boldsymbol{k}_n\}_n, \{\alpha_n\}_n, \{\beta_n\}_n$
```

It is important to know that extremal eigenvalues of the input matrix C can be identified with very few iterations, since the iterative Lanczos process tends to find extremal eigenvalues first. Together with the potentially fast decay of the covariance matrix spectrum, it is expected to get a very efficient covariance spectrum approximation, cf. [Kie08].

Lanczos' method is well documented in standard textbooks for numerics. We follow e.g. [SB02, Saa03] for a very brief review. The method is described in Algorithm 8. Even though it is allowed to use arbitrary hermitian matrices as input, only the case of symmetric positive definite matrices is considered here. Thus for the symmetric positive definite covariance matrix $C \in \mathbb{R}^{N_{NY}} \times \mathbb{R}^{N_{NY}}$ the Lanczos method constructs after ℓ iterations a tridiagonal symmetric matrix $C_\ell \in \mathbb{R}^\ell \times \mathbb{R}^\ell$ with

$$C_\ell = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{\ell-2} & \alpha_{\ell-1} & \beta_{\ell-1} \\ & & & \beta_{\ell-1} & \alpha_\ell \end{bmatrix}.$$

Here, the total number of iterations ℓ is either a fixed quantity, which corresponds to the number of eigenvalues that shall be approximated, or the iteration process can be stopped for small absolute values β_n in terms of a classical error-based stopping criterion.

During the Lanczos process, orthonormal Krylov vectors $\boldsymbol{k}_1, \ldots, \boldsymbol{k}_\ell \in \mathbb{R}^{N_{NY}}$ are built. It holds

$$C_\ell = K_\ell^T C K_\ell$$

for the matrix $K_\ell \in \mathbb{R}^{N_{NY} \times \ell}$, which is given as the column-wise concatenation of the Krylov vectors

$$K_\ell = \left[\begin{array}{c|c|c|c} \boldsymbol{k}_1 & \boldsymbol{k}_2 & \ldots & \boldsymbol{k}_\ell \end{array}\right].$$

The computational complexity of the Lanczos method for dense input matrices is $O({N_{NY}}^2 \cdot \ell)$. Furthermore, the QR method for tridiagonal systems is used to evaluate the eigenvalue, eigen-

Algorithm 9 Iterative eigenvalue decomposition

```
Require: $C$ hermitian
1: function ITERATIVEEIGENVALUESOLVER($C$, $\ell$)
2:     $(\{\boldsymbol{k}_n\}_{n=1\dots\ell}, \{\alpha_n\}_{n=1\dots\ell}, \{\beta_n\}_{n=1\dots\ell}) \leftarrow$ LANCZOS($C$, $\ell$)
3:     $C_\ell \leftarrow \{\alpha_n\}_{n=1\dots\ell}, \{\beta_n\}_{n=1\dots\ell}$
4:     $(\lambda_n(C_\ell), \boldsymbol{e}_n(C_\ell))_{n=1,\dots,\ell} \leftarrow$ TRIDIAGONALEIGENVALUESOLVER($C_\ell$)
5:     for $n = 1, 2, \dots, \ell$ do
6:         $\boldsymbol{e}_{n_\ell} = K_\ell \boldsymbol{e}_n(C_\ell)$
7:         $\lambda_{n_\ell} = \lambda_n(C_\ell)$
8:     return $\{\boldsymbol{e}_{n_\ell}\}_{n_\ell=1\dots\ell}, \{\lambda_{n_\ell}\}_{n_\ell=1\dots\ell}$
```

vector pairs $\left(\tilde{\lambda}_n(C_\ell), \boldsymbol{e}_n(C_\ell)\right)_{n=1,\dots,\ell}$. It can be implemented with a computational complexity of $O(\ell)$ operations. As the computed eigenvalues $\tilde{\lambda}_n(C_\ell)$ are approximations to a subset of the eigenvalues $\tilde{\lambda}_{n_\ell}(C)$ of matrix C, it is left to approximate the eigenvectors $\boldsymbol{\psi}_{n_\ell}^{N_{NY}}$ of the matrix C by multiplication with the transformation matrix

$$\boldsymbol{\psi}_{n_\ell}^{N_{NY}} \approx K_\ell \boldsymbol{e}_n(C_\ell) \qquad\qquad \forall n = 1, \dots, \ell\,.$$

which requires to store the Krylov vectors $\boldsymbol{k}_n$. The index n_ℓ, $\ell \leq N_{NY}$, shall indicate the mapping of the computed eigenvalue subset from the reduced matrix C_ℓ to the eigenvalues of the matrix C. Altogether we end up with the iterative eigenvalue decomposition method with $O(\ell \cdot N_{\mathcal{D}}{}^2)$ operations which is presented in a slightly more general form in Algorithm 9.

9.3 Multi-GPU parallel implementation

In the multi-GPU implementation, we have three important algorithmic steps. These are the approximation of the covariance matrix C, the derivation of Karhunen-Loève expansion coefficients λ_k and the evaluation of the truncated a-posteriori Karhunen-Loève expansion from (9.1). The implementation extends the existing numerical framework from Chapter 4.

The first step, thus the approximation of the covariance matrix C involves the numerical evaluation of $\mathrm{Cov}\left[\boldsymbol{u}\right](\boldsymbol{x}_i, \boldsymbol{x}_j)$, with $\boldsymbol{x}_i, \boldsymbol{x}_j$ the finite difference discretization points. Here, the dominant computational task is the evaluation of $N_\Gamma{}^2$ quadrature problems associated to the means of the multiplied kernels, $\mathbb{E}\left[k(\cdot, \boldsymbol{y}_j)k(\cdot, \boldsymbol{y}_{j'})\right](\boldsymbol{y}_j, \boldsymbol{y}_{j'})$, cf. Section 4.4. This evaluation is parallelized on multiple GPUs. Results are collected via MPI.

To approximate the Karhunen-Loève expansion coefficients, which is the second step, we have to find the eigenvalues of C. As mentioned before, this is done by a newly implemented multi-GPU parallel Lanczos method with full reorthogonalization. The Lanczos method allows for a straight-forward (parallel) decomposition of the $O(N_{NY}{}^2)$ storage matrix C over many GPUs, since only matrix-vector and vector-vector operations have to be performed. This opens the door for the use of a parallel multi-GPU cluster. Note that it might even be possible to perform the full matrix-vector product without storing it, in terms of an in-place method. The actual implementation is performed by extending the self-implemented multi-GPU parallel

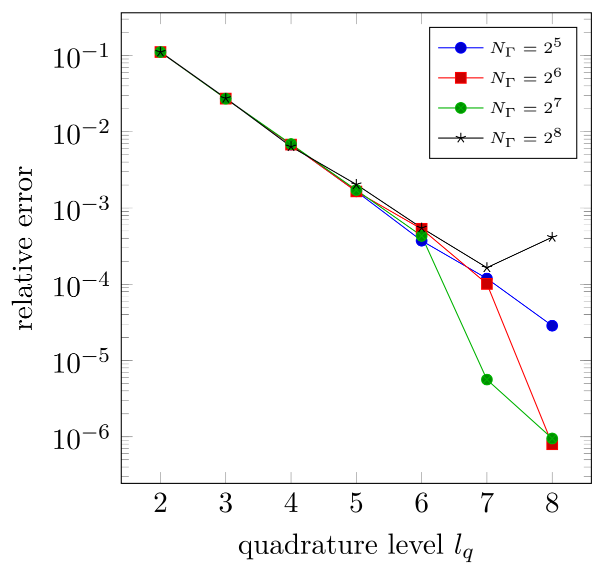


Figure 9.1: Covariance estimate error with Gaussian kernel for explicitly known solution and different numbers of collocation points.

linear algebra library *parla*, cf. Section 4.8 by the Lanczos algorithm. Extracting eigenvalues from the tridiagonal matrix C_ℓ is done on a single GPU with the tridiagonal eigenvalue solver delivered by the CULA library. Eigenvectors are constructed in a multi-GPU fashion.

Eventually, truncated versions of the a-posteriori Karhunen-Loève expansion are evaluated as third step. This is done on GPUs by appropriate kernels. Resulting approximated flow fields are again, cf. Section 4.8.8, written to VTK files.

9.4 Numerical results and performance measurements

In the following, we study convergence and approximation results for different random (two-phase) Navier-Stokes equation problems. If not otherwise stated, the discretization parameters for the Navier-Stokes equations in space and time are chosen identical to Section 6.1 and full tensor-product Clenshaw-Curtis rules are used for quadrature.

9.4.1 Flow over a backward-facing step

Since the flow over a backward-facing step with random inflow velocity, density and viscosity, cf. Section 3.3, is fast to compute, we perform a series of comparative studies for this example.

We start with the error in the evaluation of the covariance field. In order to have an exact reference solution, the first component of the velocity field at $t = 0$, thus $\pi \equiv u_1(\boldsymbol{y}, \boldsymbol{x}, 0)$, is used as quantity of interest. We measure the error in the diagonal entries of the covariance matrix, compared to the analytically known solution. Figures 9.1 and 9.2 display convergence results for different kernels, thus the Gaussian kernel $k_\epsilon, \epsilon = 1.0$ and Wendland kernels of first and third order, in presence of different levels of quadrature and collocation points. In contrast

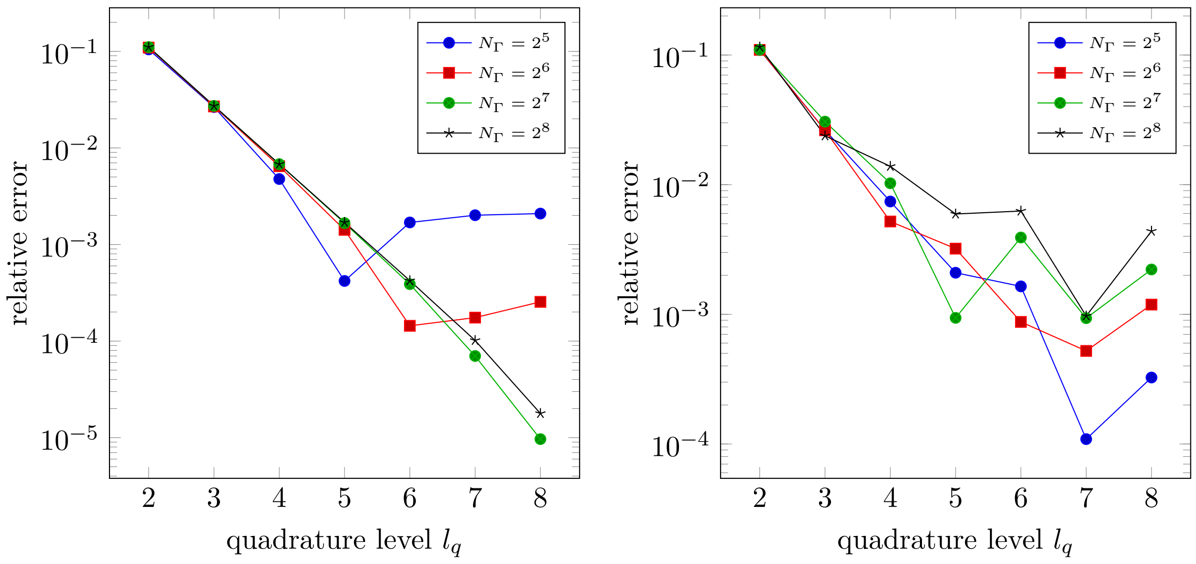


Figure 9.2: Covariance estimate error with Wendland kernel of order 1 (left) and order 3 (right) for explicitly known solution.

to previous convergence studies of this kind, the results present the relative error with respect to a growing number of *quadrature points* instead of collocation points, because the quadrature error is the dominating error for these kinds of approximations.

The Gaussian kernel gives in Figure 9.1 almost the same results, for all collocation levels, up to a quadrature level of $l_q = 6$. This is where the quadrature error starts to be resolved. For collocation point counts $N_\Gamma = 2^5, 2^6, 2^7$ one gets a further decrease in error, as long as one applies a rather strict regularization of $\epsilon_{reg} = 10^{-7}$. However for a collocation level of 8, thus 256 realizations, the error goes up again due to conditioning issues.

On the other hand, in Figure 9.2 on the left-hand side, the robust first-order Wendland kernel $k_{3,1}$ shows the classically expected convergence result with a stagnating error depending on the collocation point level. In fact, the stagnating error level corresponds to the stochastic collocation error. However, the third order Wendland kernel $k_{3,3}$ covariance estimates again suffer from conditioning problems, cf. Figure 9.2 on the right-hand side. Here, the original kernel $k_{3,3}(\boldsymbol{y}, \boldsymbol{y}') := \varphi_{3,3}(\|\boldsymbol{y} - \boldsymbol{y}'\|)$ is replaced by $\varphi_{3,3}(\frac{1}{4}\|\boldsymbol{y} - \boldsymbol{y}'\|)$ and a regularization of $\epsilon_{reg} = 10^{-8}$ is applied.

We now like to discuss approximations to the spectrum of the covariance matrix and the reconstruction error of the Karhunen-Loève series. Our major interest for this test case is the first component of the velocity field for the backward facing step simulation at $t = 10.0s$, thus $\pi \equiv u_1(\boldsymbol{y}, \boldsymbol{x}, 10)$. All remaining tests use Wendland kernels of first order, thus $k_{3,1}$.

Figure 9.3 shows on the left-hand side the decay of the covariance spectrum for a fixed quadrature level and different numbers of collocation points. With growing number of collocation points, a wider range of the actual continuous spectrum is approximated. A similar result holds for the case of a fixed collocation level and growing number of quadrature points as presented on the right-hand side of the same figure. Here, the spectrum is almost identical for

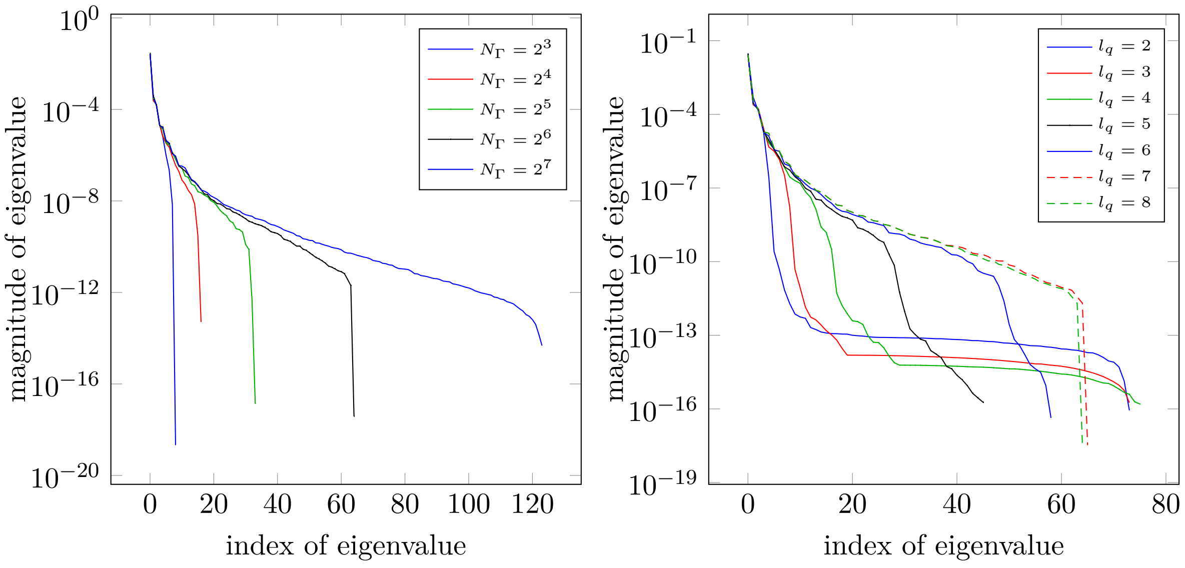


Figure 9.3: Covariance spectrum decay for fixed quadrature level $l_q = 8$ and different collocation levels (left) and the same spectrum decay for a fixed collocation point count $N_\Gamma = 2^6$ and different quadrature levels l_q, both computed with Wendland kernels of first order

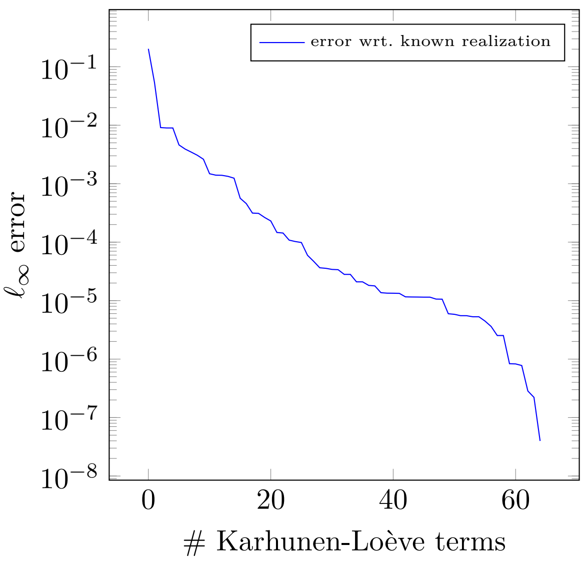


Figure 9.4: Error development of an a-posteriori Karhunen-Loève expansion used for the reconstruction of the flow field u_1 that is computed deterministically as reference solution.

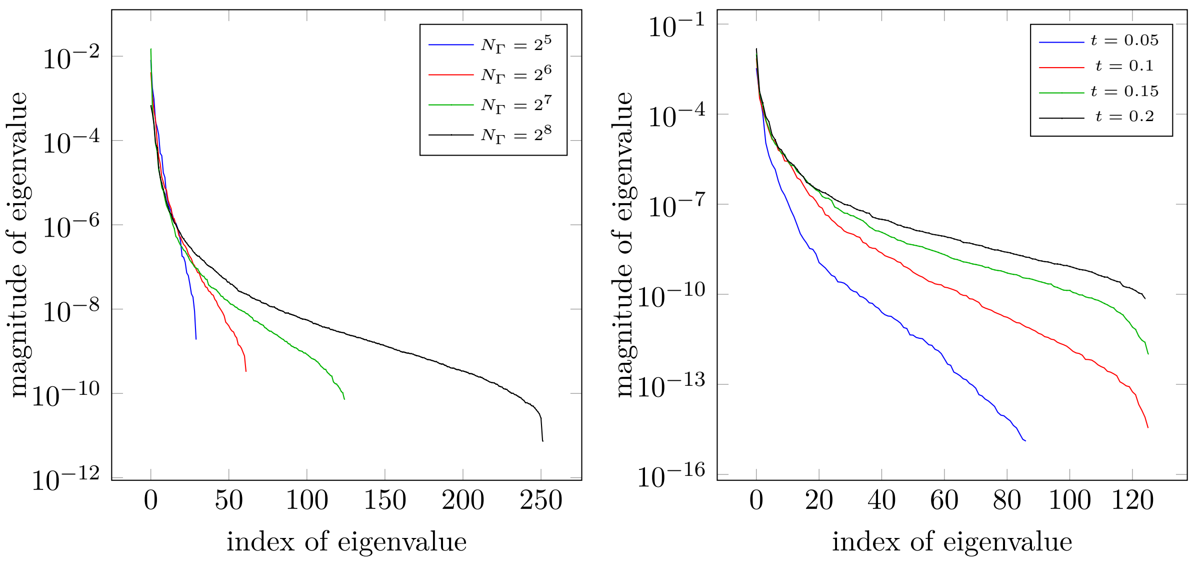


Figure 9.5: Left: Covariance spectrum decay for the two-phase bubble flow problem at $t = 0.2\,s$ with quadrature level $l_q = 6$ and a growing number of collocation points. Right: The spectrum changes over time, here approximated with $N_\Gamma = 2^7$ collocation points at different time steps.

quadrature levels $q = 7$ and $q = 8$, indicating that the collocation error dominates the results starting from these levels.

The Karhunen-Loève expansion is also used to reconstruct a known, deterministically computed stochastic realization. The result of this error analysis is given in Figure 9.4 for the specific case of a Wendland kernel $k_{3,1}$ with $N_\Gamma = 64$ collocation points, quadrature level $l_q = 7$, a regularization of $\epsilon_{reg} = 10^{-13}$ and the quantity of interest $\pi \equiv u_1(\boldsymbol{y}, \boldsymbol{x}, 10)$. As expected, the error decay behaves similar to the covariance spectrum decay. Note that we use here an ℓ_∞ error. It is possible to see that with only 20 terms of the Karhunen-Loève expansion, an error in the range of 10^{-4} is achieve, cf. Figure 9.4. This might be already enough accuracy for many engineering applications.

9.4.2 Two-phase Navier-Stokes problem with random volume forces

After having discussed a series of results based on the first small-scale test case, we now move forward to a two-phase flow example. Let us mention here that evaluating the following results imposes a huge computational effort and was only possible by applying up to 32 GPUs at the same time on a single problem.

The specific application problem is a bubble flow with random volume forces that are modeled by a Karhunen-Loève expansion, cf. Section 3.3, with the quantity of interest $\pi = u_1(\boldsymbol{y}, \boldsymbol{x}, t)$. This application problem is discretized in physical space by $N_\mathcal{D} = 48^3$ grid points and solved over time with adaptive time-stepping, cf. Section 3.3. Random volume forces from (3.24) are modeled with a correlation length of $L_c = 2.0$. The input Karhunen-Loève expansion is truncated with $N_{KL} = 3$. Numerical quadrature in stochastic space uses full tensor product

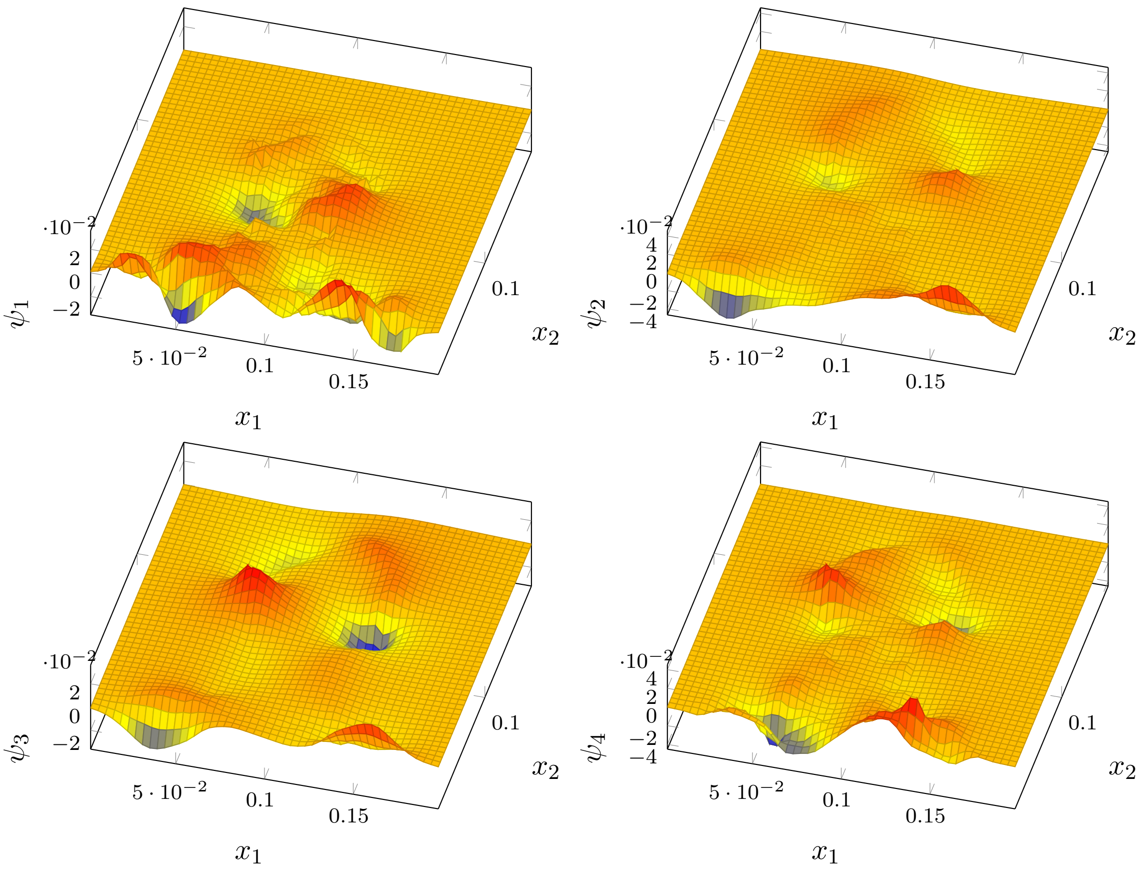


Figure 9.6: The first four eigenvectors $\psi_i(\boldsymbol{x}, t)$, of the output covariance for the two-phase bubble flow problem, here visualized along a slice with $x_3 = 0.1$ and $t = 0.2$.

Clenshaw-Curtis rules with a quadrature level of $l_q = 6$, thus 274625 abscissas. We have to evaluate up to $(2^8)^2 = 65536$ of these quadrature problems. In the RBF stochastic collocation method, we apply the robust first-order Wendland kernels $k_{3,1}$ with regularization $\epsilon_{reg} = 10^{-13}$. Note that we construct a covariance matrix with $(48^3)^2$, thus roughly 12 billion entries, in a multi-GPU fashion. This matrix fills 91.125 GB of distributed GPU memory. It takes only about 4.5 seconds to approximate 128 eigenvalues and eigenvectors (including the tridiagonal QR solve) on 32 GPUs. Some results on parallel scalability of the underlying dense linear algebra library *parla* were already presented in Section 7.4.3.

Figure 9.5 shows on the left-hand side the computed covariance spectrum $\mathrm{Cov}\,[u_1]$ for different numbers of collocation points at $t = 0.2$ seconds. Eigenvalues of absolute value smaller than 10^{-15} are skipped. We observe a strong covariance spectrum decay within the first few eigenvalues. The spectrum levels out depending on the number of collocation points used for the approximation of the covariance. Eigenvectors corresponding to the first four eigenvalues and $N_\Gamma = 2^7$ are displayed in Figure 9.6. A slice with $x_3 = 0.1$ is used to display the eigenvectors ψ_i which are scalar fields in three space dimensions.

To highlight the time-dependence of the covariance spectrum, finally a second covariance spectrum study is given. Figure 9.6 shows on the right-hand side, for $N_\Gamma = 2^7$ collocation points and all other parameters fixed to the previous settings, a changing spectrum over time. Even though there is an initially fast decay for all four time steps, the decay slows down in the upper part of the spectrum for increased time. This might give an indicator for a loss of smoothness in the response surface function of the two-phase Navier-Stokes equations for growing time.

10 Anisotropic RBF kernel-based stochastic collocation

Most random PDE problems have random *input* parameter fields, which can be approximated by Karhunen-Loève expansions. A fast decay in the covariance spectrum of the exact random input parameter field leads to a fast convergence of its Karhunen-Loève expansion. From e.g. [BNT10], we know further that the fast to exponential decay of the input random parameter field does very often map to a similar decay of the covariance spectrum of the output of the random PDE problem. The previous chapter highlighted a numerical study of this input-output relationship. In the literature on e.g. sparse-grid based stochastic collocation methods, problem-specific knowledge is used to construct anisotropic stochastic collocation approximations, cf. [NTW08a], which sometimes allows to break the *curse of dimensionality* for an increasing number of Karhunen-Loève terms, thus stochastic dimensions.

In this chapter, an anisotropic construction for radial basis function kernel-based stochastic collocation is introduced and applied with Gaussian and Matérn kernels. It combines the advantages of purely isotropic kernel-based methods with the improved convergence rates of e.g. anisotropic sparse grid constructions for random PDE problems with a fast decay of the covariance spectrum. Based on previous work on anisotropic RBF interpolation in [CMM07, BDL10, FHW12, GLS13] and motivated by similar constructions in the field of Gaussian process regression, cf. [RW05, Section 5.1], *anisotropic RBF kernels* are introduced by using a weighted norm in the RBF construction with decaying weights either derived by a-priori knowledge or derived based on the first few eigenvalues of the output covariance operator, cf. Chapter 9. This construction leads to a weighted native kernel function space.

Following error estimates for anisotropic RBF interpolation in [BDL10], the new approach is at least expected to feature convergence improvements by constant factors for Gaussian and Matérn kernels in case of optimal weight choice. This leads to clearly improved convergence results in the pre-asymptotic regime, which is of high importance for large-scale applications. Furthermore, for Gaussian kernels, the proposed method is identical to interpolation by weighted tensor-product kernels. Therefore, the proposed anisotropic method complements research results from [FHW12], in which non-constructive dimension-independence results for tensor-products of Gaussian kernels are given. These results suggest to expect dimension-independence of the nominal stochastic dimension in case of a fast decaying output covariance spectrum.

One missing ingredient towards constructive results in [FHW12] is the lack of optimal collocation points, i.e., an optimal design, for anisotropic interpolation for arbitrarily sized point sets. In other words, given an anisotropic RBF kernel, the optimal choice of collocation points in the corresponding anisotropic native spaces does not seem to be obvious, especially when no tensor-product grid shall be used. To overcome this difficulty, a greedy adaptive method, proposed in [DMSW05], is used to construct almost optimal collocation points with respect to the worst-case error. Here, the numerical tool is the *power function* of a reproducing kernel Hilbert space. For appropriately smooth, strictly positive definite radial-symmetric kernel functions, it is possible to give upper bounds to the kernel interpolation error of functions in the respective native space in terms of a product of the power function and the function's native space norm [Sch95]. This estimate motivates to use a greedy method that finds point sets, which minimize the power function, to achieve a small overall error. Note that, in contrast to classical adaptive methods, this approach does not involve any function evaluations, thus deterministic PDE solves. It purely depends on the characteristics of the underlying native space. Combining interpolation by anisotropic kernels with collocation points generated by the greedy adaptive method for the respective native space leads to the proposed anisotropic stochastic collocation method.

This chapter starts by introducing anisotropic RBF kernel functions leading to a weighted native space. An appropriate choice of weights for anisotropic stochastic collocation is discussed. Thereafter, the power function of a reproducing kernel-Hilbert space with the greedy adaptive method are under consideration. By combining both techniques, it will be possible to show clearly improved convergence results for random PDE problems with fast covariance spectrum decay. Numerical results even suggest dimension-independent convergence in some cases. Together with an efficient GPU parallel implementation this leads to an optimal anisotropic kernel method which is not only optimal in terms of convergence rate but which is also fast.

10.1 Anisotropic RBF kernel interpolation

Classical interpolation with RBF kernels uses kernel functions of the form

$$k(\boldsymbol{y}, \boldsymbol{y}') = \varphi(\|\boldsymbol{y} - \boldsymbol{y}'\|_2)\,,$$

for $\boldsymbol{y}, \boldsymbol{y}' \in \Gamma$, cf. Section 4.2. The Eucledian distance metric imposes an isotropic handling of all *directions* in the sampled space $\Gamma \subset \mathbb{R}^{N_{FN}}$. To introduce an anisotropic behavior, [RW05, Section 5.1] and others propose to introduce a generalized norm

$$\|\boldsymbol{y}\|_M := \left(\boldsymbol{y}^\top M \boldsymbol{y}\right)^{1/2}\,,$$

with $M \in \mathbb{R}^{N_{FN} \times N_{FN}}$ a symmetric positive (semi-)definite matrix. This norm would allow to model a non-axis aligned anisotropic behavior. However, we are here only interested in axis aligned anisotropies, thus we can set

$$M := \operatorname{diag}(\boldsymbol{\gamma})^2$$

and $\boldsymbol{\gamma} := (\gamma_1, \ldots, \gamma_{N_{FN}})$ is a weight vector. We are now able to formally introduce anisotropic RBF kernels as

$$k^\gamma(\boldsymbol{y}, \boldsymbol{y}') = \varphi(\|\boldsymbol{y} - \boldsymbol{y}'\|_\gamma), \quad \text{with} \quad \|\boldsymbol{y}\|_\gamma := \left(\boldsymbol{y}^\top \operatorname{diag}(\boldsymbol{\gamma})^2 \boldsymbol{y}\right)^{1/2}\,.$$

In Figure 10.1, as an example, the isotropic and one anisotropic Matérn kernel are compared. The anisotropic kernel interpolation problem looks, given an anisotropic kernel func-

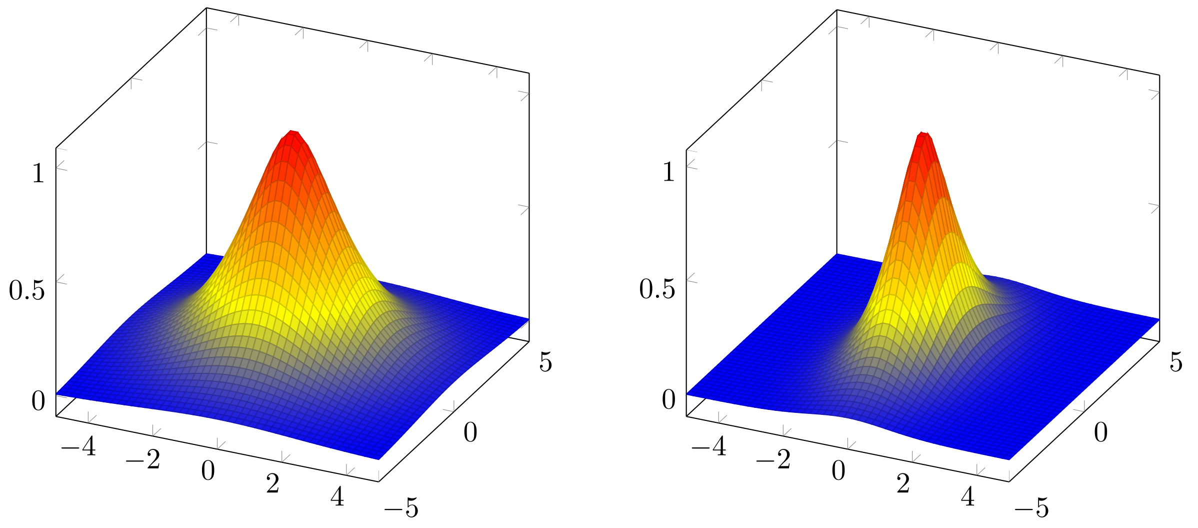


Figure 10.1: By replacing the isotropic Matérn kernel (left) with an anisotropic Matérn kernel (right), it is possible to introduce directional dependence into kernel interpolation problems.

tion $k^\gamma : \Gamma \times \Gamma \to \mathbb{R}$ and function evaluations $\boldsymbol{f} = (f_1, \ldots, f_{N_\Gamma}), f_i = f(\boldsymbol{y}_i)$ at data sites $X_\Gamma := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_{N_\Gamma}\}$, for a set of coefficients $\boldsymbol{\alpha}^\gamma \in \mathbb{R}^{N_{FN}}$ such that

$$\sum_{i=1}^{N_\Gamma} \alpha_i^\gamma k^\gamma(\boldsymbol{y}_j, \boldsymbol{y}_i) = f_j \qquad\qquad \forall j = 1, \ldots, N_\Gamma$$

and we have the interpolant $s^\gamma_{X_\Gamma, f}$ with

$$f(\boldsymbol{y}) \approx s^\gamma_{X_\Gamma, f}(\boldsymbol{y}) = \sum_{i=1}^{N_\Gamma} \alpha_i^\gamma k^\gamma(\boldsymbol{y}, \boldsymbol{y}_i)\,.$$

Obviously the anisotropic Lagrange basis functions for the data sites X_Γ are given as

$$L_i^\gamma(\boldsymbol{y}_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \quad \text{with} \quad L_i^\gamma \in \operatorname{span}\left\{k_\gamma(\cdot, \boldsymbol{y}) | \boldsymbol{y} \in \Gamma\right\}\,.$$

with the native space

$$\mathcal{N}_k^\gamma(\Gamma) := \overline{\operatorname{span}\{k_\gamma(\cdot, \boldsymbol{y}) | \boldsymbol{y} \in \Gamma\}}\,.$$

In fact, the equivalence of such anisotropic native spaces for standard kernel functions to some Sobolev space as well as convergence results are subject to current research, cf. [Fas12]. Recent results include non-constructive tractability results in [FHW12] and error bounds in terms of a growth function in [BDL10].

10.2 Weighting in anisotropic RBF stochastic collocation

Using the basic terminology from the last chapter, we can now formulate the anisotropic RBF stochastic collocation approximation for the full random PDE after finite noise assumption as

$$\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t) \approx (\mathcal{I}^{\gamma}_{N_\Gamma}\boldsymbol{u})(\boldsymbol{y},\boldsymbol{x},t) := \sum_{i=1}^{N_\Gamma} \boldsymbol{u}(\boldsymbol{y}_i,\boldsymbol{x},t) L^{\gamma}_i(\boldsymbol{y})\,.$$

Furthermore, we have $\mathcal{I}^{\gamma}_{N_\Gamma}\boldsymbol{u} \in \mathcal{P}_\gamma(\Gamma) \otimes L^2([0,T];L^2(\mathcal{D}))$ with $\mathcal{P}_\gamma(\Gamma)$ a weighted approximation space. For anisotropic RBF stochastic collocation, we thus have

$$\mathcal{I}^{\gamma}_{N_\Gamma}\boldsymbol{u} \in \mathcal{N}^{\gamma}_k(\Gamma) \otimes L^2([0,T];L^2(\mathcal{D}))\,.$$

In style of [NTW08a], more collocation points shall be spend in those *directions* of the stochastic space Γ, in which convergence is assumed to be slower. These directions are assumed to be *more important*. Note that the norm $\|\cdot\|_\gamma$ has already been defined such that larger values for the weights γ_i correspond to higher importance in the respective direction. That is, the coefficients in the norm are the squares of the weights γ_i.

One possible choice for weights $\boldsymbol{\gamma}$ is to use results from the literature for a given model problem. The Karhunen-Loève expansion-based random-coefficient elliptic problem from Section 3.2.2 is well-studied in [NTW08a], where a-priori estimates result in weight values

$$g(n) = \begin{cases} \frac{1}{2}\log\left(1+\sqrt{\frac{1}{24\sqrt{\pi}L_c}}\right) & \text{for } n=1\,,\\ \frac{1}{2}\left(1+\sqrt{\frac{1}{48\sqrt{\pi}L_c}}\exp\left(\frac{\lfloor\frac{n}{2}\rfloor^2\pi^2{L_c}^2}{8}\right)\right) & \text{for } n>1\,, \end{cases} \tag{10.1}$$

and we set

$$\gamma_i = \frac{1}{g(i)} \qquad\qquad i=1,\dots,N_{KL}\,. \tag{10.2}$$

Another approach is to use a-posteriori information, which becomes especially attractive for coupled engineering problems with little theoretic knowledge on the structure of the response surface, as in the two-phase Navier-Stokes case. In that case, an *a-posteriori Karhunen-Loève expansion*

$$\boldsymbol{u}(\boldsymbol{y},\boldsymbol{x},t) \approx \mathbb{E}\left[\boldsymbol{u}\right](\boldsymbol{x}) + \sum_{k=1}^{N^{\boldsymbol{u}}_{KL}} \sqrt{\lambda_k}Y_k(\boldsymbol{y})\boldsymbol{\psi}_k(\boldsymbol{x},t)$$

can be estimated, as outlined in Chapter 9. The Karhunen-Loève coefficients $\sqrt{\lambda_k}$ in that expansion are good to optimal weights for an anisotropic RBF collocation method, thus we should set

$$\gamma_i = \sqrt{\lambda_i} \qquad\qquad i=1,\dots,N_{KL}\,.$$

Some ideas to make this whole approach efficient, will be outlined in Section 10.5. Note that [NTW08a] propose, as an alternative approach, to estimate the weights γ_i by investigating dimension-wise error decay of the stochastic collocation method. This idea is not considered here. Next, we move over to the optimal sampling for a given weighting $\boldsymbol{\gamma}$ and start with the necessary information on *power functions* to understand the later-on introduced greedy method.

10.3 Power function and approximation estimate

The *power function* is introduced and discussed in e.g. [WS92, Sch95]. According to [Fas11], it is further discussed in stochastics literature as *kriging variance.* In [BDL10], the same results are formulated for anisotropic kernel interpolation. However, no proof is given. To avoid the technical details of conditionally positive definite kernels, the power function and its associated estimates are introduced here for strictly positive definite kernels, following the lines of [Fas07, Section 14.3] adapted to the notation of this thesis. Furthermore we skip the index $\boldsymbol{\gamma}$ in this section, since these results apply to isotropic and anisotropic RBF kernels. We start with definition of the Power function for general kernel native spaces.

Definition 10.1 (Power function [Fas07, Definition 14.1])**.** *Let be $\Gamma \subseteq \mathbb{R}^d$. With $k : \Gamma \times \Gamma \to \mathbb{R}$ a strictly positive definite (anisotropic) kernel function with $k \in C(\Gamma \times \Gamma)$ and $X := \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\} \subseteq \Gamma$ a finite set of distinct collocation points, we can introduce the quadratic form*

$$Q[\boldsymbol{a}](\boldsymbol{y}) = k(\boldsymbol{y}, \boldsymbol{y}) - 2\sum_{j=1}^{N} a_j k(\boldsymbol{y}, \boldsymbol{y}_j) + \sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j k(\boldsymbol{y}_i, \boldsymbol{y}_j)$$

for a given vector $\boldsymbol{a} \in \mathbb{R}^d$. Shall further be $\{L_i\}_{i=1}^{N_\Gamma}$ the Lagrange basis of the corresponding native space $\mathcal{N}_k(\Gamma)$, cf. Section 4.1. Then, the power function is defined as

$$[P_{k,X}(\boldsymbol{y})]^2 := Q[\boldsymbol{L}(\boldsymbol{y})](\boldsymbol{y})$$

with $\boldsymbol{L}(\boldsymbol{y}) := (L_1(\boldsymbol{y}) \ldots L_N(\boldsymbol{y}))^\top$.

In [Fas07, Section 14.3], there are some further remarks on the numerical evaluation of the power function which are collected here in a short Lemma on numerical evaluation. We state

Lemma 10.1 (Numerical evaluation of the power function [Fas07, Section 14.3])**.** *Let Γ, k and X be given as in Definition 10.1. Using the notation of Section 4.3, we can also evaluate the power function $P_{k,X}(\boldsymbol{y})$ as*

$$P_{k,X}(\boldsymbol{y}) = \sqrt{k(\boldsymbol{y}, \boldsymbol{y}) - (A_{k,X,\{\boldsymbol{y}\}})^\top (A_{k,X,X})^{-1} A_{k,X,\{\boldsymbol{y}\}}}\,.$$

This Lemma is derived in [Fas07, Section 14.3]. It will be needed to implement the greedy adaptive algorithm. The main result, on which the adaptive method will be based, is the following theorem on the point-wise interpolation error for functions in a native space.

Theorem 10.1 (Power function interpolation error estimate [Fas07, Theorem 14.2])**.** *With Γ, k, X as before, we consider a function $f \in \mathcal{N}_k(\Gamma)$. Their kernel (Lagrange) interpolant on X shall be $s_{f,X}$. For all $\boldsymbol{y} \in \Gamma$ we have the estimate*

$$|f(\boldsymbol{y}) - s_{f,X}(\boldsymbol{y})| \leq P_{k,X}(\boldsymbol{y}) \|f\|_{\mathcal{N}_k(\Gamma)}\,.$$

The proof is given in [Fas07]. Theorem 10.1 basically states that we can give an upper bound for the interpolation error which decouples in a product of the norm of the function and a term only depending on the kernel and the collocation points.

10.4 Greedy adaptive method for optimal collocation points

Using the mathematical tools from the last section, we now want to construct a greedy method to find almost optimal collocation point sets for approximation in the weighted native space $\mathcal{N}_k^{\gamma}(\Gamma)$. Thus, for a given function $f \in \mathcal{N}_k^{\gamma}(\Gamma)$, we seek for a finite set X' of distinct collocation points, such that

$$\begin{aligned} X' = \operatorname{argmin}_{X \subset \Gamma} \quad & \left\| f - s_{f,X}^{\gamma} \right\|_{L_\infty(\Gamma)} \\ \text{subject to} \quad & |X| \leq N_\Gamma, \quad \boldsymbol{y} \in X \text{ pairwise distinct}\,. \end{aligned} \tag{10.3}$$

Besides of the data-independent greedy method, described later, there are a few data-*dependent* approaches to approximate (10.3) in the isotropic case. One of them is proposed in [SW00]. Here, the authors suggest to use a greedy one-point algorithm to approximate the interpolant (instead of the function itself), by a stepwise minimization of the residual. This however, requires to explicitly evaluate the residual, which is hardly feasible for stochastic collocation problems in which a single function evaluation requires to solve a full time-dependent three-dimensional flow problem. Other approaches with the same problem have been proposed e.g. in [LS09, WH13].

The intention here is thus to rely on the power function estimate from the last section only. Consequently, native space dependent but data-independent collocation points with good approximation properties are constructed. This approach goes back to [DMSW05] and is shortly reviewed (with skipped $\boldsymbol{\gamma}$).

In [DMSW05], the authors start stating that the power function monotonically decreases by increasing the number of distinct collocation points. Without proof, this reads as

Lemma 10.2. *[DMSW05] Let be $\Gamma \subseteq \mathbb{R}^d$ compact satisfying an interior cone condition, cf. Definition 4.12. Shall be $k : \Gamma \to \mathbb{R}$ strictly positive definite and $X', X'' \subseteq \Gamma$ a finite distinct collocation point sets with $X' \subseteq X''$. With the power function from Definition 10.1, we have for all $\boldsymbol{y} \in \Gamma$ that*

$$P_{k,X'}(\boldsymbol{y}) \geq P_{k,X''}(\boldsymbol{y}), \qquad \forall \boldsymbol{y} \in \Gamma\,.$$

We see that adding more collocation points will never increase the power function. Together with Theorem 10.1 this gives a first indicator for convergence of the point-wise error $|f(\boldsymbol{y}) - s_{f,X}(\boldsymbol{y})|$. However, one should keep in mind that adding more points also affects the native space norm and thus might affect the native space norm of the function to be interpolated on the right-hand side of the inequality in Theorem 10.1.

In Algorithm 10, the greedy (data-independent) algorithm from [DMSW05] is summarized. The authors suggest to maximize the power function over a very large discrete subset of the space Γ. They are able to prove a theorem on convergence of this method in terms of $\lim_{j\to\infty} \|P_{k,X_j}\|_{L_\infty(\Gamma)} = 0$. It is given here with adapted notation and without proof as

Algorithm 10 Greedy data-independent collocation point construction [DMSW05]

```
Require: $\Gamma \subseteq \mathbb{R}^d$ compact, sat. interior cone condition, $k : \Gamma \times \Gamma \to \mathbb{R}$ strictly positive definite
1: function GreedyPointGenerator($\Gamma$, $k$)
2:     $X_1 = \{\boldsymbol{y}_1\}$, $\boldsymbol{y}_1 \in \Gamma$ arbitrarily chosen
3:     for $j = 2, 3, \ldots, N$ do
4:         $\boldsymbol{y}_j = \operatorname{argmax}_{\Gamma \setminus X_{j-1}} \|P_{k,X_{j-1}}\|_{L_\infty(\Gamma)}$
5:         $X_j = X_{j-1} \cup \{\boldsymbol{y}_j\}$
6:     return $X_N$
```

Theorem 10.2. *[DMSW05, Theorem 4.3] With Γ as in Lemma 10.2 and $k \in C^2(\Gamma_1 \times \Gamma_1)$ strictly positive definite and defined on a convex and compact domain Γ with $\Gamma_1 \supseteq \Gamma$, Algorithm 10 converges at least like*

$$\|P_{k,X_j}\|_{L_\infty(\Gamma)} \leq C j^{-1/d} ,$$

with $C > 0$ a constant.

In this thesis, in difference to [DMSW05], the maximization shall not be done by maximizing over a large point set, but by numerical optimization, as outlined in the next section.

10.5 Implementation

The greedy adaptive sampling Algorithm 10 is currently implemented in Matlab, since it is a pre-processing step and thus not performance-critical for the overall method. As said before, the original publication [DMSW05] implements step four of Algorithm 10 by a maximization over a large set of sampling points in stochastic space. In difference to this, the proposed implementation uses the *interior-point method* [BV04, Chapter 11] for constraint maximization of the norm of the power function by the `fmincon` command. Constraints are the boundaries of the stochastic domain Γ. The power function for anisotropic kernels is evaluated as proposed in Lemma 10.1. Here, the inverse of the interpolation matrix can be precomputed at the beginning of each iteration. Due to cancellation effects, the squared power function sometimes becomes negative. To achieve a stable evaluation even in those cases, the real part of the power function is considered only. Note that the minimization process always depends on the initial guess of a new collocation point. To reduce this dependence, each iteration of the greedy adaptive algorithm performs n_{rep} optimizations with different (uniformly random sampled) initial guesses and takes the best result.

Before constructing the optimal anisotropic collocation points, it might also be necessary to compute an a-posteriori Karhunen-Loève expansion to get optimal weights $\boldsymbol{\gamma}$. This is done as proposed in Chapter 9. To make this overall approach efficient, the a-posteriori Karhunen-Loève expansion should only be approximated by a few stochastic collocation points and lower PDE space discretization resolution $N_{\mathcal{D}}$ to get a rough estimate for the covariance spectrum decay. Thereafter, many collocation points can be spend on the anisotropic approximation. Note that it is even possible to reuse the previously calculated stochastic realizations in the final anisotropic method by replacing the initial collocation point set X_1 in Algorithm 10 by the isotropic point set used in the a-posteriori Karhunen-Loève analysis.

With given weights and collocation points, the standard RBF stochastic collocation implementation with a different norm in the RBF approximation can be used. Here, the GPU-based code from Section 4.8 is applied.

10.6 Numerical results

In the following, numerical results for the proposed anisotropic RBF collocation method are presented. This starts with a discussion of expected convergence results and examples of optimized anisotropic collocation point sets. Afterwards numerical studies for the random-coefficient elliptic problem from Section 3.2.2 and for the two-phase Navier-Stokes equation test case, with rising bubble and Karhunen-Loève based random volume force from Section 3.3, are presented.

10.6.1 Kernels and convergence expectations

The proposed anisotropic RBF-based stochastic collocation will be applied with Gaussian and Matérn kernels, cf. Section 4.3.2. In [BDL10], error estimates in terms of a growth function are given for anisotropic RBF-based interpolation. An example therein suggests that the application of appropriate anisotropic interpolation improves the upper bound to the error at least by a constant factor over the pure isotropic case. Therefore, it is expected to see an error improvement by anisotropy, at least in the pre-asymptotic regime for Matérn and Gaussian kernels.

Further, non-constructive tractability results in [FHW12] give worst-case error estimates for isotropic and anisotropic interpolation with Gaussians with respect to the corresponding native space. In the case of interpolation or approximation by function values only, a *dimension-independent* convergence rate α in the number of collocation points is given as

$$\alpha := -\max\left(\frac{r(\boldsymbol{\gamma})}{1+\frac{1}{2r(\boldsymbol{\gamma})}}, \frac{1}{4}\right), \quad \text{with} \quad r(\boldsymbol{\gamma}) = \sup\left\{\beta > 0 \,\middle|\, \sum_{i=1}^{\infty} {\gamma_i}^{1/\beta} < \infty\right\}.$$

Here, the γ_i are the decaying anisotropy weights given as before. This says that a given decay in the weights of an anisotropic interpolation problem corresponds to a fixed convergence rate independent of the stochastic dimension N_{KL}.

As a consequence, one might hope to break the curse of dimensionality in case of approximation by anisotropic Gaussian kernels. Nevertheless, it is important to notice that already in the isotropic case interpolation by Gaussians does often result in exponential rates on arbitrarily smooth functions, which might make it impossible to see the dimension-independence in practice. Also, interpolation with Gaussians tends to lead to extremely bad conditioned problems for larger collocation point counts. Therefore, asymptotic convergence rates for Gaussian kernels might be hardly visible. On the other hand, there are no published results on dimension-independence for anisotropic Matérn kernels, at least to the authors knowledge.

Overall it is expected that anisotropic RBF-based stochastic collocation outperforms the

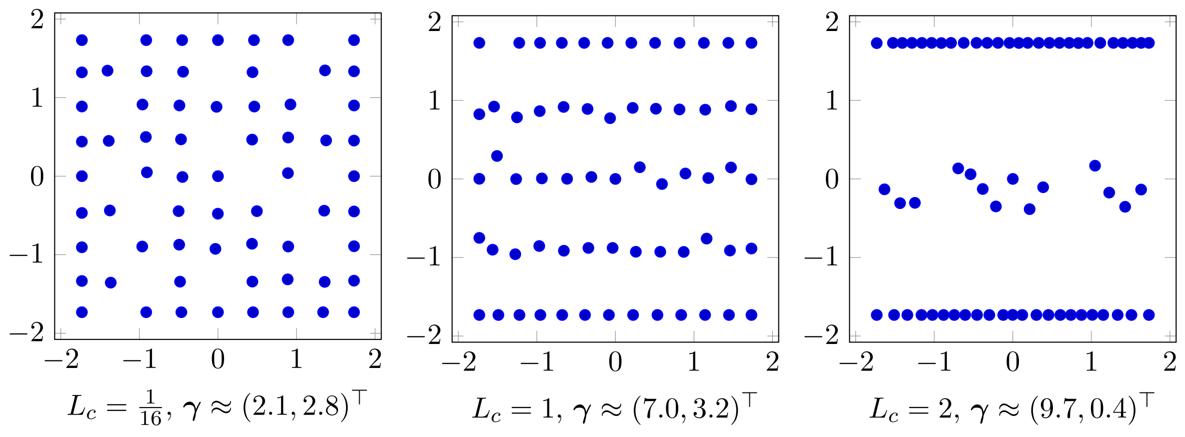


Figure 10.2: The greedy adaptive method constructs almost optimal anisotropic collocation point sets, here, for anisotropic Matérn kernels and the Karhunen-Loève based elliptic problem and different correlation lengths, with weights $\boldsymbol{\gamma}$ following [NTW08a].

isotropic approach at least in the pre-asymptotic regime with some indications towards dimension independence.

10.6.2 Collocation point sets for anisotropic RBF stochastic collocation

To get an example for optimal collocation point sets in anisotropic RBF collocation, a two-dimensional stochastic space $\Gamma = [-\sqrt{3}, \sqrt{3}]^2$ is considered. The weights are computed according to the a-priori weight estimates in (10.1) and (10.2). Figure 10.2 presents collocation point sets for different correlation lengths L_c repeating the optimization process $n_{rep} = 32$ times. Here, a scaled Matérn kernel k_β with $\beta = \frac{d+3}{2} = 2.5$ is used. For a correlation length $L_c = \frac{1}{16}$, an almost regular isotropic grid is constructed. Correlation length $L_c = 1$ corresponds to a weak anisotropy of roughly 2:1. Finally $L_c = 2$ leads to a very anisotropic problem with a point set largely limited to two boundaries. Anisotropic point sets stemming from the Greedy-adaptive method for Gaussian kernels have similar structure. However, the optimization process tends to break down for large collocation point counts and Gaussian kernels due to conditioning issues. Appropriate regularization or preconditioning is subject to future research.

Even though the collocation points in Figure 10.2 seem to have a very regular structure, there is no obvious way to construct them for arbitrary number of points and arbitrary kernel functions, in an a-priori fashion. Therefore, the greedy optimization process is currently the method of choice for construction.

10.6.3 Karhunen-Loève based random coefficient elliptic problem

The first numerical results for mean approximation by anisotropic RBF stochastic collocation are given for the Karhunen-Loève based random-coefficient elliptic problem from Section 3.2.2. The PDE was discretized in space by finite differences and $N_{\mathcal{D}} = 512^2$ grid points and solved

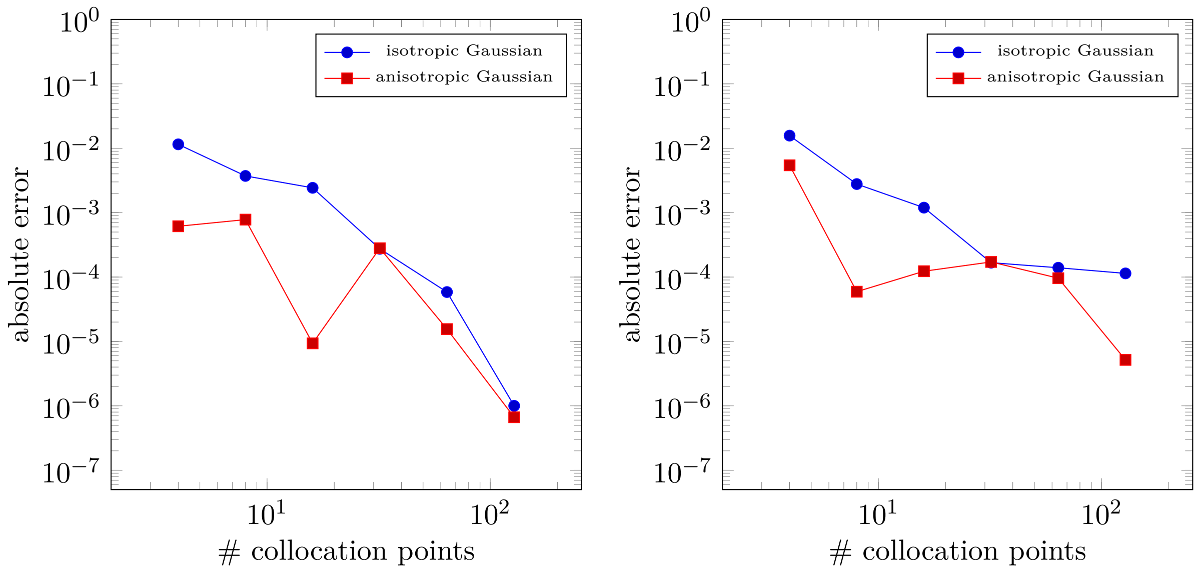


Figure 10.3: Comparison of convergence results for isotropic and anisotropic RBF-based stochastic collocation with *Gaussian* kernels in case of the Karhunen-Loève based elliptic model problem, correlation length $L_c = 2.0$ and stochastic dimensions $N_{KL} = 3$ (left) and $N_{KL} = 5$ (right).

by the algebraic multigrid method from Chapter 8. The objective is to approximate

$$\mathbb{E}\left[\frac{1}{N_{\mathcal{D}}}\sum_{i=1}^{N_{\mathcal{D}}} u(\cdot, \boldsymbol{x}_i)\right] .$$

Errors will be computed by taking the absolute value of the difference between two approximation levels. For problems with stochastic dimension 3 and 5, tensor-product Clenshaw-Curtis quadrature with levels $l_q = 7$ and $l_q = 5$ is used. In higher dimensions, sparse grid quadrature of level 4 is applied, due to limited computational resources. The number of repetitions in the greedy method is $n_{rep} = 32$. Kernel regularization is set to $\epsilon_{reg} = 10^{-12}$.

We start with numerical results for anisotropic stochastic collocation with Gaussian kernels and a correlation length of $L_c = 2$ corresponding to a strong anisotropy in the first stochastic dimension. Weights $\boldsymbol{\gamma}$ are chosen according to equations (10.1) and (10.2). Figure 10.3 gives numerical results for stochastic dimensions $N_{KL} = 3$ on the left-hand side and $N_{KL} = 5$ on the right-hand side with convergence graphs for the isotropic and anisotropic case. The Gaussian kernel k_ϵ is applied with $\epsilon = 0.2$. Anisotropic stochastic collocation clearly outperforms the isotropic approach. Due to conditioning issues and the randomized nature of the point optimization process, the error decay is not as stable for the anisotropic case as for the isotropic case. These additional errors make it difficult to give clear statements on dimension-independence.

In a second study, the anisotropic stochastic collocation method with Gaussian kernels and $\epsilon = 0.1$ is applied to the same random PDE with a smaller correlation length of $L_c = 1.0$ and at least three dominant stochastic dimensions. In Figure 10.4 on the left-hand side, numerical results are given for the classical isotropic stochastic collocation approach and stochastic

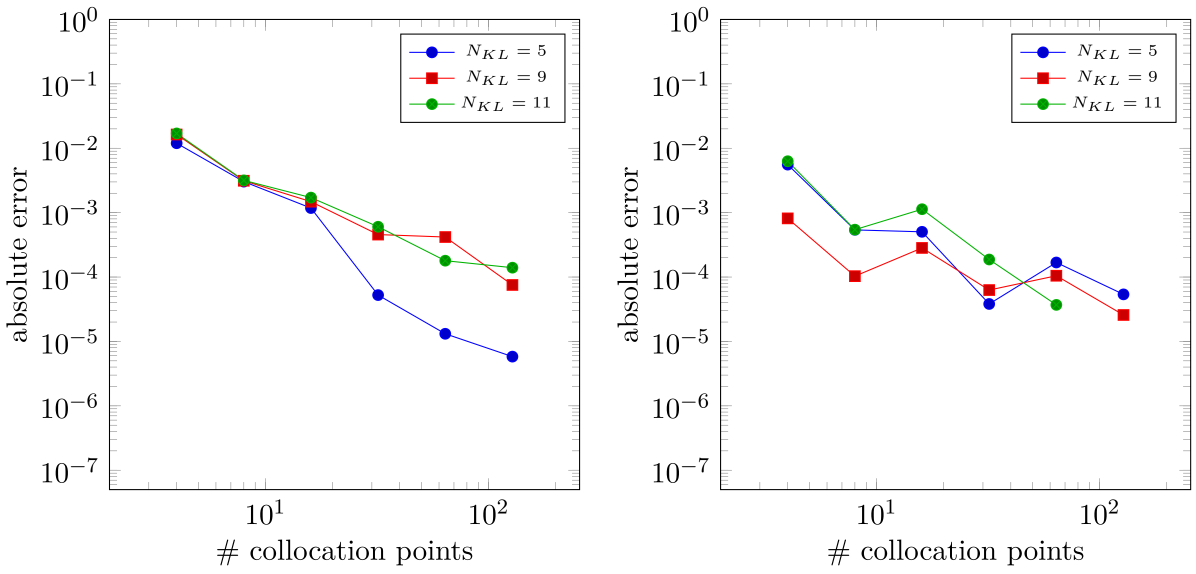


Figure 10.4: The Karhunen-Loève based elliptic random PDE problem is also studied for a smaller correlation length $L_c = 1.0$, *Gaussian* kernels, higher dimensions and with isotropic (left) and anisotropic (right) convergence results.

dimensions $N_{KL} = 5, 9, 11$. A dimension-dependence becomes evident. The right-hand side of the same figure outlines results for anisotropic stochastic collocation and the same stochastic dimensions. The results seem to indicate a more robust behavior towards the stochastic dimension. Overall, for stochastic dimensions $N_{KL} = 9, 11$, the anisotropic method outperforms the classical method. In dimension $N_{KL} = 5$, the isotropic approach has a lower error, maybe due to the fact that the decaying weights for $L_c = 1.0$ are not sufficiently resolved by five stochastic dimensions.

The same model problems are again approximated with (an)isotropic Matérn kernels k_β and $\beta = \frac{N_{KL}+3}{2}$, thus

$$k_{\frac{N_{KL}+3}{2}} = (1 + \sigma\|\boldsymbol{y} - \boldsymbol{y}'\|)\, e^{-\sigma\|\boldsymbol{y}-\boldsymbol{y}'\|}\,. \tag{10.4}$$

Parameter σ allows to introduce an additional scaling. Figure 10.5 presents convergence results for the $L_c = 2.0$ test case with $N_{KL} = 3$ on the left-hand side and $N_{KL} = 5$ on the right hand side. In the presented results, a scaling of $\sigma = \frac{1}{16}$ is used in the anisotropic three-dimensional case and and $\sigma = \frac{1}{20}$ for all other tests. Anisotropy leads to both, higher convergence rates and pre-asymptotic lower errors. In some cases, the error is smaller by impressive several orders of magnitude.

Finally, the study for correlation length $L_c = 1.0$ is repeated for the Matérn kernel case. Figure 10.6 presents the results with convergence graphs for isotropic approximation on the left-hand side and anisotropic approximation on the right-hand side. The scaling parameter is set to $\sigma = \frac{1}{20}$ for the five-dimensional problem while we have $\sigma = \frac{1}{16}$ for $N_{KL} = 9, 11$. Similar to the Gaussian kernel case, the isotropic approximation is dimension-dependent in its convergence properties. On the other hand, the anisotropic approximation is almost identical in error decay for all involved dimensions, giving hints towards a dimension-independent convergence

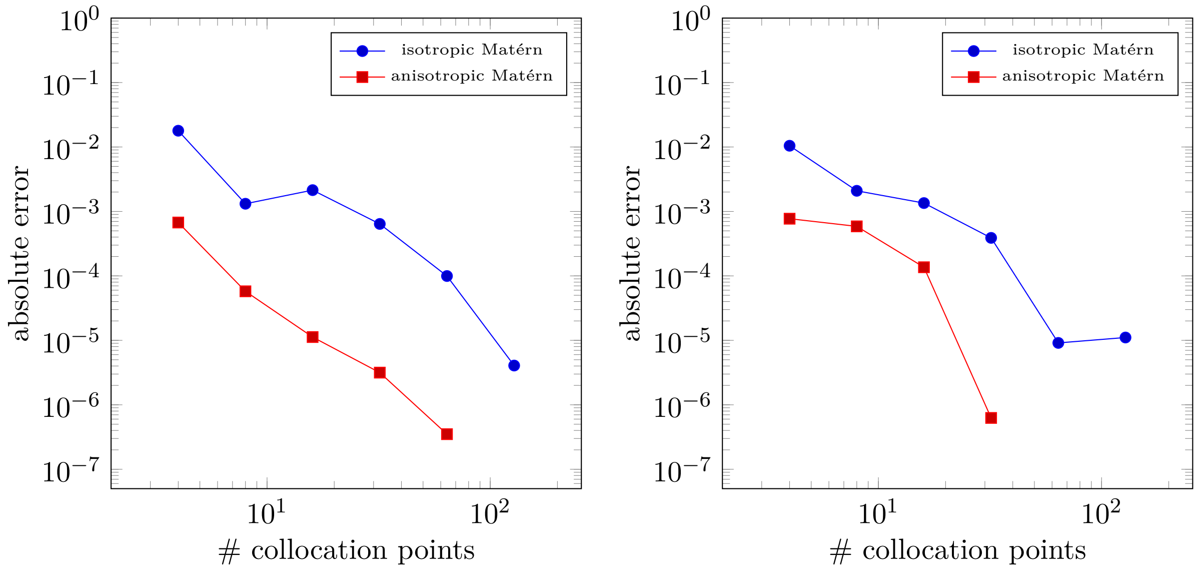


Figure 10.5: Convergence of the mean for the Karhunen-Loève based random coefficient elliptic problem at correlation length $L_c = 2.0$ approximated with isotropic and anisotropic RBF stochastic collocation and stochastic dimensions $N_{KL} = 3$ (left) and $N_{KL} = 5$ (right) in case of *Matérn* kernels.

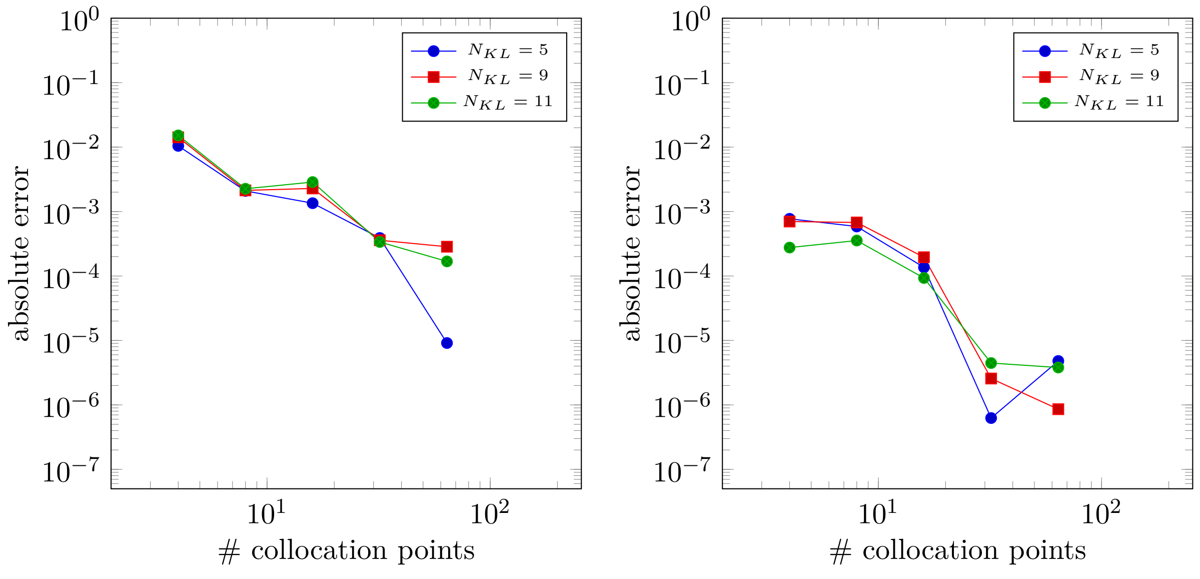


Figure 10.6: The comparison of convergence results for the isotropic (left) and anisotropic (right) *Matérn* kernel gives hints towards dimension-independent convergence for the anisotropically approximated elliptic problem and an appropriate choice of weights $\boldsymbol{\gamma}$, which have to correspond to the given correlation length $L_c = 1.0$.

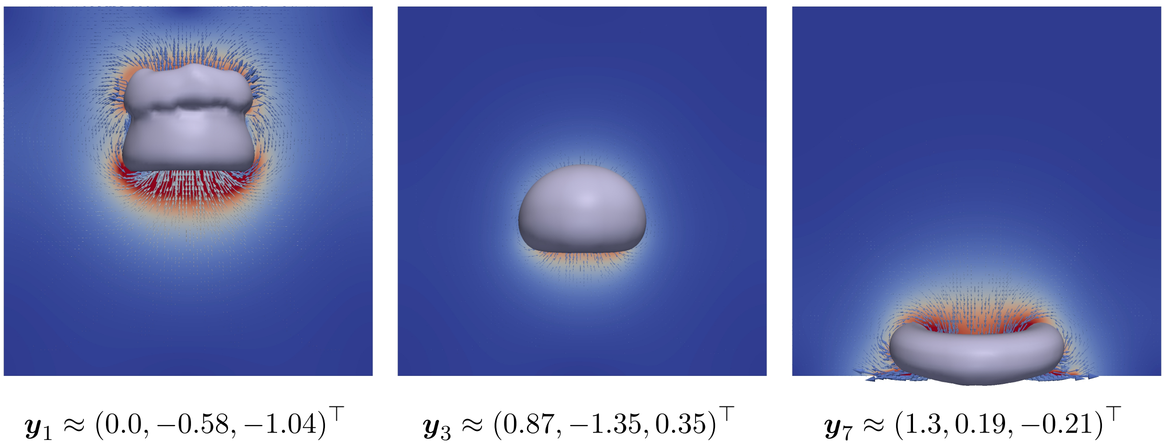

$\boldsymbol{y}_1 \approx (0.0, -0.58, -1.04)^\top$
$\boldsymbol{y}_3 \approx (0.87, -1.35, 0.35)^\top$
$\boldsymbol{y}_7 \approx (1.3, 0.19, -0.21)^\top$

Figure 10.7: Flow field realizations of the bubble flow experiment with volume forces modeled by a truncated Karhunen-Loève expansion, for different parameters $\boldsymbol{y}_i$ at $t = 0.2\,s$.

behavior even though this is not necessarily supported by appropriate theory. Convergence of the anisotropic method levels out for more than 32 collocation points. At the same level, the optimization process for optimal point construction did break down due to conditioning problems. This again suggests, that appropriate regularization or preconditioning might improve the presented results even more. However, this is future work. Nevertheless, anisotropic stochastic collocation gives a clear improvement in the error for the discussed model problem.

10.6.4 Two-phase Navier-Stokes problem with random volume forces

To exemplify the optimality of anisotropic RBF stochastic collocation for a highly complex engineering problem, a convergence study for the two-phase Navier-Stokes problem with a rising bubble is discussed next. Randomness is introduced by a random volume force field $\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{x})$, cf. Section 3.3, which is approximated by a Karhunen-Loève expansion, truncated such that the stochastic dimension is $N_{KL} = 3$. The covariance of the approximated random field has a correlation length of $L_c = 2.0$. This corresponds to a single strongly anisotropic dimension in stochastic space. Figure 10.7 gives three typical flow field realizations for this random Navier-Stokes problem. Here, the free surface is visualized by an iso-surface. Furthermore, a cut-plane through the velocity field displays vector glyphs for the velocities and colors their magnitude.

In difference to the elliptic random PDE problem, there is no theory on the properties of the response surface mapping of this random two-phase Navier-Stokes problem. Therefore, there is little or no knowledge on the covariance structure of the solution of the random PDE problem. This holds even more for the derived quantity of interest $\pi_{center}(t)$ from Section 3.3, thus the bubble center position, for $t = 0.2$, for which the expectation value shall be approximated in the following. Consequently, choosing appropriate weights $\boldsymbol{\gamma}$ is no obvious operation. The aim is to try first the analytically given weights, cf. equations (10.1) and (10.2), from the random elliptic PDE problem. This might be a good choice since the same input covariance structure is used in both problems. Afterwards, as discussed in Section 10.2, almost optimal weighting based on an a-posteriori Karhunen-Loève expansion will be used.

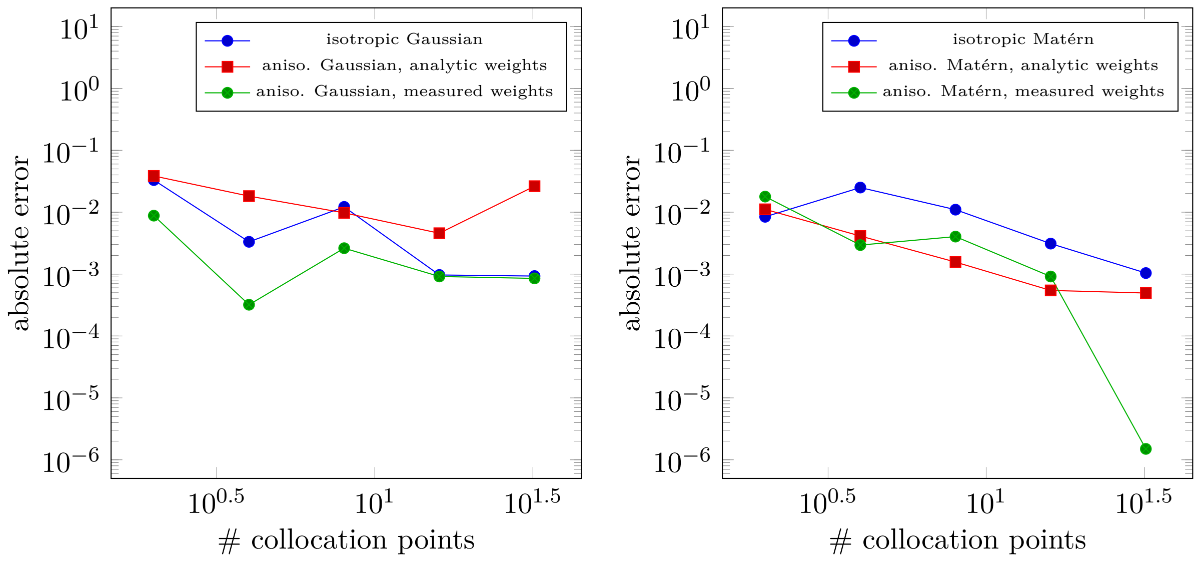


Figure 10.8: Convergence of the mean bubble center at $t = 0.2$ in the rising bubble flow two-phase Navier-Stokes problem with Karhunen-Loève based random volume forces at correlation length $L_c = 2.0$ and $N_{KL} = 3$ stochastic dimensions, approximated with isotropic and anisotropic RBF kernel-based stochastic collocation using Gaussian (left) and Matérn (right) kernels.

To approximate the mean bubble center at $t = 0.2$ seconds, a series of two-phase Navier-Stokes problems has to be solved. These are discretized in space by a uniform grid with $N_{\mathcal{D}} = 100^3$ unknowns. Note that solving the problem for one single collocation point or realization up to a physical time of $t = 0.2$ seconds takes approximately more than two hours of compute time on a GPU, thus this is a rather computationally challenging problem. Consequently, the construction of a true *overkill* solution is not feasible. Therefore, the error will be computed as the norm of the difference between two subsequent solutions. Throughout the numerical experiments, regularization is set to $\epsilon_{reg} = 10^{-12}$ and quadrature is performed with tensor product Clenshaw-Curtis rules at level $l_q = 7$. As for the previous numerical studies, Gaussian and Matérn kernel results are discussed.

Figure 10.8 displays convergence results for isotropic stochastic collocation and anisotropic stochastic collocation using analytically given weights and weights computed by the a-posteriori Karhunen-Loève expansion from Section 9.4.2. Results for Gaussian kernel are given on the left-hand side of Figure 10.8 while results for the Matérn kernel are on the right-hand side of the same figure. The isotropic approximation by the Gaussian kernel k_ϵ is done for scaling parameter $\epsilon = 0.5$. In the anisotropic case with analytic weights, a scaling of $\epsilon = 0.1$ is used, while the measured weights are applied together with a scaling of $\epsilon = 0.5$. Note that scaling parameters are usually chosen such that best possible convergence is achieved. Applying a Gaussian kernel to approximate the random two-phase Navier-Stokes problem leads to serious conditioning problems. Therefore, all results show a rather unstable convergence behavior. The lowest error is achieved with the measured weights, as assumed.

Approximation by Matérn kernels is clearly outperforming the Gaussian case. The scaling parameter in the Matérn kernel function given in (10.4) is set to $\sigma = \frac{1}{20}$ for isotropic approximation and to $\sigma = \frac{1}{12}$ for both anisotropic approximations. Linear regression shows that isotropic approximation has an algebraic rate of 0.9. By applying analytic weights, an improved rate of 1.2 is achieved. However, the best result stems back from the use of measured weights, derived in Chapter 9. With only 16 collocation points an error in the range of 10^{-6} is achieved. Measuring the convergence rate by linear regression, gives a rate of 2.9. This error behavior is a pretty impressive optimal result, underlining the great advantage of the proposed anisotropic method with measured weights.

The above numerical results clearly show that anisotropic RBF-based stochastic collocation almost always outperforms the classical isotropic approach. Some numerical results give hints towards confirming theoretic dimension-independence results for anisotropic Gaussian kernels, though pre-asymptotic behavior and bad conditioning makes this analysis challenging. Appropriate regularization and preconditioning methods are subject to future research.

However, anisotropic stochastic collocation with Matérn kernels is robust, both in the elliptic random PDE test case and the very challenging large-scale random two-phase Navier-Stokes problem. Using a-priori predicted analytic weights for the elliptic and the flow problem gives always improvements in the error. These improvements are significant for the elliptic case. For the Navier-Stokes case, the use of measured weights, which have been approximated by a lower-resolution a-posteriori Karhunen-Loève expansion, leads to optimal approximation results. Overall, the proposed anisotropic approach considerably reduces the error at least in the pre-asymptotic regime, which is of extreme importance for large-scale random PDE problems, and gives hints towards weakening or breaking the curse of dimensionality.

11 Conclusions

Summary

In this thesis, the application of uncertainty quantification to *large-scale problems* has been considered, thus stochastic moments of solutions of random partial differential equations were computed. Large-scale problems require the use of *non-intrusive methods*, since a complete redevelopment of corresponding solvers is out of question. The main contribution of this thesis is therefore the establishment of a numerical framework for the optimal solution of large-scale complex random PDE problems by the newly developed non-intrusive *RBF kernel-based stochastic collocation method*. Apart from elliptic model problems, the target application were the two-phase incompressible Navier-Stokes equations. It could be demonstrated that a previously computationally almost intractable problem can now be solved in a high-order convergent, scaling, parallel and computational optimal fashion within just a few hours of runtime.

High- to exponential-order convergence with low preasymptotic error After discretization in stochastic space, non-intrusive uncertainty quantification methods require to solve for each stochastic sample a single, potentially very computationally challenging PDE. Given a fixed target error tolerance, it is therefore indispensable to keep the number of discretization points in stochastic space as low as possible. The obvious way to overcome this issue is to introduce an approximation method in stochastic space, which has high or even exponential convergence orders with a very small pre-asymptotic error. This has been achieved in this thesis by the introduction of the RBF kernel-based stochastic collocation method. It combines ideas from classical *stochastic collocation* methods based on spectral (sparse) tensor-product approximations with optimal pre-asymptotic convergence from *kriging* and the profound stochastic framework from *Gaussian process regression*. The key idea is to use interpolation in reproducing kernel Hilbert spaces to approximate in stochastic space. Interpolation is done by Lagrange basis functions which are collocated in a set of mesh-free points, sampled from a quasi-Monte Carlo - type Halton sequence. The Lagrange basis is constructed in the so-called *native space of* (strictly positive definite) *radial kernel basis functions*. From theory, we know that function approximation in those native spaces shows some best-approximation properties.

Numerical results were given that underline the excellent properties of the method. For problems with (known) high smoothness in stochastic space, up to exponential convergence has been achieved. In cases of low smoothness in stochastic space, algebraic convergence rates were shown. A small error in the pre-asymptotic regime was almost always present. These results clearly outperformed well-known established methods such as (quasi-)Monte Carlo, (sparse) spectral tensor-product stochastic collocation or Polynomial Chaos expansion, in important application problems. This new result is fundamental for practical applications.

Optimal empirical error balance Due to missing solution theory, a profound analysis of the full stochastic approximation method is currently impossible for problems, which involve the two-phase incompressible Navier-Stokes equations in strong formulation. Therefore, an empirical error balance analysis has been done for these equations, as well as for an elliptic model problem. To this end, suitable discrete approximations error norms in Bochner spaces were introduced and related to continuous norms. It then became possible to numerically measure the partial error terms of the (additive) upper error bound in mean approximation of stationary random PDE problems. These experiments allowed to give a rough, empirical estimate of the asymptotic behavior of the involved errors. It was consequential possible to formulate empirical asymptotic conditions on all discretization parameters, such as quadrature level, number of collocation points or PDE discretization level, to achieve a total mean solution approximation error below a given threshold. Even more, these results allowed to give a computational complexity estimate for the proposed method, which was formulated in terms of the achieved error. This is new in the field.

Fast approximation and optimal scalability Even highly complicated *random* three-dimensional two-phase Navier-Stokes problems could be solved within a few hours. This became possible by parallelizing all relevant numerical methods on GPUs. The key component here was the two-phase incompressible flow solver NaSt3DGPF, which was reimplemented in all core components on GPUs, with a factor of three of speedup over equally priced standard CPU hardware and a reduced power consumption by a factor of two. The solver was further parallelized to scale on a multi-GPU cluster with almost optimal weak scaling efficiency on up to 48 GPUs. Multi-GPU parallel libraries for the iterative solution of dense linear algebra problems were furthermore developed, with applications in kernel interpolation and eigenvalue approximation. Moreover, sampling, quadrature and stochastic collocation were also implemented on GPUs. (Multi-)GPU codes for the corresponding problems were not available before.

Optimal preconditioning Preconditioning techniques were applied to iterative sparse and dense linear solvers, to achieve (almost) problem independent iteration counts. In case of the solution of the dense linear system in interpolation by non-compactly supported kernel functions, preconditioning by so-called local Lagrange bases was used. This special kind of restricted additive Schwarz method is constructed by solving many small dense linear algebra problems. Together with a multi-GPU implementation, a perfectly scaling preconditioner has been implemented which leads to optimal problem-size independent convergence for boundary-free problems and almost optimal convergence in case of the presence of boundaries. More than a factor of five in performance improvement could be shown for model problems. The approach is new on GPUs and for the discussed problem class.

To overcome the complexity bottleneck of solving discretized elliptic equations in model and application problems, a robust hybrid Ruge-Stüben algebraic multigrid method has been implemented for single-GPU applications. It uses a hybrid CPU-based coarse/fine grid point classification and parallelizes all remaining parts of the setup phase and the solve phase on a GPU. Existing state-of-the-art CPU implementations could be outperformed by up to 50 percent in runtime on almost equally priced hardware. Moreover, the time-to-solution for elliptic problems in uncertainty quantification could be halved in contrast to the application of standard GPU-based, preconditioned iterative methods.

Weakening or breaking the curse of dimensionality Finally, the intrinsic dependence on the stochastic dimension, with exponential increase in the number of collocation points for growing dimension, was weakened or even removed for random PDE problems with fast decaying output covariance spectrum. The crucial idea was the introduction of an anisotropic weighting in the underlying radial basis functions. While there is knowledge on optimal weighting of elliptic model problems with random coefficients, theory is not available for the two-phase Navier-Stokes equations. This is why a tool to analyze their output covariance spectrum had to be developed. It approximates the Karhunen-Loève expansion of the solution field of the random PDE, by solving a large-scale dense eigenvalue problem on multi-GPU clusters. Coefficients in the Karhunen-Loève expansion are optimal weights for the proposed weighted approximation method. The newly developed *anisotropic RBF stochastic collocation method* required good to optimal problem-specific sampling in stochastic space. This was made possible by a Greedy optimization process, which minimizes the power function of the corresponding anisotropic kernel function native space. Numerical experiments gave clear indicators towards the weakening or even breaking of the curse of dimensionality in elliptic model problems and two-phase flow problems. A lower pre-asymptotic error over isotropic calculations has been always achieved. This is a fundamental and new result for the given numerical method and the discussed problems.

Overall, the proposed methodology clearly showed that a combined effort of optimal numerical methods and scalable, parallel implementations is necessary to solve large-scale real-world problems in uncertainty quantification. Here, (anisotropic) RBF kernel-based stochastic collocation is a new technique which outperforms existing methods for many important problems.

Outlook

The proposed new framework based on RBF kernel-based stochastic collocation, gives many opportunities for future research. While this thesis clearly targets real-world engineering problems with largely unknown theory, numerical analysis has to be performed for model problems, to get provable convergence and error balance results. Certainly, an important contribution would be to enrich asymptotic results by profound knowledge on the involved constants. These are of crucial importance in real applications. An increased knowledge on error balance, would further allow to design new multi-level schemes which take into account the interplay between PDE discretization error and stochastic discretization error. This could again lead to a decrease in total compute time with high impact on large-scale problems.

An important component of such multi-level schemes would be the proposed anisotropic method in stochastic space. Here future research is indispensable. The given empirical studies towards dimension-independent convergence gave clear hints towards a complex interaction between conditioning in the Greedy sampling method and robustness in the shown results. Here, convergence theory and knowledge on error propagation has to be established. Anisotropic regularization seems to be a valuable approach for future developments. While the proposed anisotropic method is favorable for problems of known structure or for cases in which we can compute empirical estimates on optimal weights, a general-purpose approach should include adaptivity towards the different stochastic dimensions. Developments of cheap a-priori error estimators will be important. This might be closely connected to model reduction.

Even more, local adaptivity in (stochastic) space is a considerable research topic. High impact of such adaptivity is expected for the field of failure probability prediction, thus a different class of problems in uncertainty quantification. Remembering that failure probability prediction is identical to the evaluation of the mean of a characteristic function applied to the random PDE solution, adaptivity will be the method of choice for optimal approximation of the jump in this quantity of interest. As a result, the proposed multi-purpose framework would be able to approximate stochastic moments and failure probabilities. Obviously, the methodology is not limited to elliptic problems or computational fluid dynamics. Therefore, other fields of applications should be addressed, too.

Finally, the applied highly scalable preconditioning for kernel-based interpolation might be one important building block for future Exascale numerical algorithms. After the implementation of appropriate tree or multipole methods, an extremely scalable iterative, thus fault-tolerant, $O(cN_\Gamma \log N_\Gamma)$ complexity technique for mesh-free methods would be available. It would be local, would have a high instruction throughput and could be optimal in terms of problem-size independent convergence. Fields of application would be scalable, mesh-free integration and quadrature as well as the solution of partial differential equations by collocation methods. This could be a new algorithmic design pattern for future scalable, error-resilient and optimal numerical methods.

Bibliography

[ABB$^+$09a] B.M. Adams, L.E. Bauman, W.J. Bohnhoff, K.R. Dalbey, M.S. Ebeida, J.P. Eddy, M.S. Eldred, P.D. Hough, K.T. Hu, J.D. Jakeman, L.P. Swiler, and D.M. Vigil. DAKOTA, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.4 User's manual. Sandia Technical Report SAND2010-2183, Sandia National Lab, December 2009. Updated April 2013.

[ABB$^+$09b] B.M. Adams, L.E. Bauman, W.J. Bohnhoff, K.R. Dalbey, M.S. Ebeida, J.P. Eddy, M.S. Eldred, P.D. Hough, K.T. Hu, J.D. Jakeman, L.P. Swiler, and D.M. Vigil. DAKOTA, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: Version 5.4 Theory manual. Sandia Technical Report SAND2011-9106, Sandia National Lab, December 2009. Updated November 2013.

[ABB$^+$99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, Third edition, 1999.

[AS] M. Anitescu and M.L. Stein. Homepage of the ScalaGAUSS project at the Argonne National Laboratory. `http://press3.mcs.anl.gov/scala-gauss`. last access: 2014/08/24.

[BNT10] I. Babuška, F. Nobile, and R. Tempone. A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM Review*, 52(2):317–355, 2010.

[BTZ04] I. Babuška, R. Tempone, and G. Zouraris. Galerkin finite element approximations of stochastic elliptic partial differential equations. *SIAM Journal on Numerical Analysis*, 42(2):800–825, 2004.

[BAB$^+$13] S. Balay, M.F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W.D. Gropp, D. Kaushik, M.G. Knepley, L. Curfman McInnes, K. Rupp, B.F. Smith, and H. Zhang. PETSc Users manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.

[BBC$^+$94] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the solution of linear systems: Building blocks for iterative methods, 2nd edition*. SIAM, Philadelphia, PA, 1994.

[BSS13] A. Barth, C. Schwab, and J. Sukys. Multilevel Monte Carlo approximations of statistical solutions of the Navier-Stokes equations. SAM report 2013-33, 2013.

[BSZ11] A. Barth, C. Schwab, and N. Zollinger. Multi-level Monte Carlo finite element method for elliptic PDEs with stochastic coefficients. *Numer. Math.*, 119(1):123–161, September 2011.

[BCM99] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11(2-3):253–270, 1999.

[BDL10] R.K. Beatson, O. Davydov, and J. Levesley. Error bounds for anisotropic RBF interpolation. *Journal of Approximation Theory*, 162(3):512–527, 2010. Special Issue: Bommerholz Proceedings Conference on Multivariate Approximation (Bommerholz 2008).

[BL97] R.K. Beatson and W.A. Light. Fast evaluation of radial basis functions: Methods for two-dimensional polyharmonic splines. *IMA Journal of Numerical Analysis*, 17(3):343–372, 1997.

[BLB00] R.K. Beatson, W.A. Light, and S. Billings. Fast solution of the radial basis function interpolation equations: Domain decomposition methods. *SIAM J. Sci. Comput.*, 22(5):1717–1740, May 2000.

[BN92] R.K. Beatson and G.N. Newsam. Fast evaluation of radial basis functions: I. *Computers & Mathematics with Applications*, 24(12):7–19, 1992.

[Beb11] M. Bebendorf. Adaptive cross approximation of multivariate functions. *Constructive Approximation*, 34(2):149–179, 2011.

[BMS14] M. Bebendorf, Y. Maday, and B. Stamm. Comparison of some reduced representation approximations. In A. Quarteroni and G. Rozza, editors, *Reduced Order Methods for Modeling and Computational Reduction*, volume 9 of *MS&A - Modeling, Simulation and Applications*, pages 67–100. Springer International Publishing, 2014.

[BNTT14] J. Beck, F. Nobile, L. Tamellini, and R. Tempone. A quasi-optimal sparse grids procedure for groundwater flows. In M. Azaïez, H. El Fekih, and J.S. Hesthaven, editors, *Spectral and High Order Methods for Partial Differential Equations - ICOSAHOM 2012*, volume 95 of *Lecture Notes in Computational Science and Engineering*, pages 1–16. Springer International Publishing, 2014.

[BDO12] N. Bell, S. Dalton, and L. Olson. Exposing fine-grained parallelism in algebraic multigrid methods. *SIAM Journal on Scientific Computing*, 34(4):C123–C152, 2012.

[BG09] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 18:1–18:11, New York, NY, USA, 2009. ACM.

[BG12] N. Bell and M. Garland. Cusp: Generic parallel algorithms for sparse matrix and graph computations, 2012. Version 0.3.0.

[BS02] W. Benenson and H. Stöcker. *Handbook of Physics.* Prentice Hall, 2002.

[BS13] P. Benner and J. Schneider. Uncertainty quantification for Maxwell's equations using stochastic collocation and model order reduction. Preprint MPIMD/13-19, Max Planck Institute Magdeburg, October 2013.

[BBR13] M. Bernaschi, M. Bisson, and D. Rossetti. Benchmarking of communication techniques for GPUs. *J. Parallel Distrib. Comput.*, 73(2):250–255, February 2013.

[BLMS13] H. Bijl, D. Lucor, S. Mishra, and C. Schwab, editors. *Uncertainty Quantification in Computational Fluid Dynamics*, volume 92 of *Lecture Notes in Computational Science and Engineering.* Springer International Publishing, 2013.

[BV04] S.P. Boyd and L. Vandenberghe. *Convex Optimization.* Berichte über verteilte Messysteme. Cambridge University Press, 2004.

[BKZ92] J.U. Brackbill, D.B. Kothe, and C. Zemach. A continuum method for modeling surface tension. *J. Comput. Phys.*, 100(2):335–354, 1992.

[BCHZ13] J. Brannick, Y. Chen, X. Hu, and L. Zikatanov. Parallel unsmoothed aggregation algebraic multigrid algorithms on GPUs. In O.P. Iliev, S.D. Margenov, P.D. Minev, P.S. Vassilevski, and L.T Zikatanov, editors, *Numerical Solution of Partial Differential Equations: Theory, Algorithms, and Their Applications*, volume 45 of *Springer Proceedings in Mathematics & Statistics*, pages 81–102. Springer New York, 2013.

[BG04] H.-J. Bungartz and M. Griebel. Sparse grids. *Acta Numerica*, 13:147–269, 5 2004.

[BG13] M. Burkow and M. Griebel. A full three dimensional numerical simulation of the sediment transport and the scouring at a rectangular obstacle. 2013. Submitted to Computers & Fluids.

[BNTT11] J. Bäck, F. Nobile, L. Tamellini, and R. Tempone. Stochastic spectral Galerkin and collocation methods for PDEs with random coefficients: A numerical comparison. In J.S. Hesthaven and E.M. Rønquist, editors, *Spectral and High Order Methods for Partial Differential Equations*, volume 76 of *Lecture Notes in Computational Science and Engineering*, pages 43–62. Springer Berlin Heidelberg, 2011.

[Caf98] R.E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. *Acta Numerica*, 7:1–49, 1 1998.

[CS99] X. Cai and M. Sarkis. A restricted additive schwarz preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(2):792–797, 1999.

[CJW$^+$06] J.A. Carlson, A. Jaffe, A. Wiles, Clay Mathematics Institute, and American Mathematical Society. *The Millennium Prize Problems*. American Mathematical Society, 2006.

[CMM07] G. Casciola, L.B. Montefusco, and S. Morigi. The regularizing properties of anisotropic radial basis functions. *Applied Mathematics and Computation*, 190(2):1050–1062, 2007.

[Che13] J. Chen. On the use of discrete Laplace operator for preconditioning kernel matrices. *SIAM Journal on Scientific Computing*, 35(2):A577–A602, 2013.

[CQR14] P. Chen, A. Quarteroni, and G. Rozza. Comparison between reduced basis and stochastic collocation methods for elliptic problems. *Journal of Scientific Computing*, 59(1):187–216, 2014.

[CL91] A.H.-D. Cheng and O.E. Lafe. Boundary element solution for stochastic groundwater flow: Random boundary condition and recharge. *Water Resources Research*, 27(2):231–242, 1991.

[CCGC13] O.A. Chkrebtii, D.A. Campbell, M.A. Girolami, and B. Calderhead. Bayesian uncertainty quantification for differential equations. *ArXiv e-prints*, June 2013.

[Cho67] A.J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 2:12–26, 1967.

[Cho68] A.J. Chorin. Numerical solution of the Navier-Stokes equations. *Mathematics of Computation*, 22(104):745–762, 1968.

[CFY12] I. Cialenco, G.E. Fasshauer, and Q. Ye. Approximation of stochastic partial differential equations by a kernel-based collocation method. *Int. J. Comput. Math.*, 89(18):2543–2561, December 2012.

[Cla08] S. Claus. Numerical simulation of unsteady three-dimensional viscoelastic Oldroyd-B and Phan-Thien Tanner flows. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, July 2008.

[CFH$^+$99] A.J. Cleary, R.D. Falgout, V.E. Henson, J.E. Jones, T.A. Manteuffel, S.F. McCormick, G.N. Miranda, and J.W. Ruge. Robustness and scalability of algebraic multigrid. *SIAM J. Sci. Comput.*, 21(5):1886–1908, December 1999.

[CC60] C.W. Clenshaw and A.R. Curtis. A method for numerical integration on an automatic computer. *Numerische Mathematik*, 2(1):197–205, 1960.

[CGST11] K.A. Cliffe, M.B. Giles, R. Scheichl, and A.L. Teckentrup. Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients. *Computing and Visualization in Science*, 14(1):3–15, 2011.

[CDBC13] P.M. Congedo, C. Duprat, G. Balarac, and C. Corre. Numerical prediction of turbulent flows using Reynolds-averaged Navier–Stokes and large-eddy simulation with uncertain inflow conditions. *International Journal for Numerical Methods in Fluids*, 72(3):341–358, 2013.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[Cor] NVIDIA Corporation. Webpage of AmgX. https://developer.nvidia.com/amgx. last access: 2014/06/08.

[CL11] A. Corrigan and R. Löhner. Porting of FEFLO to multi-GPU clusters. *Proceedings of the 49th AIAA Aerospace Sciences Meeting, Orlando, Florida, USA*, page , 2011.

[CFL67] R. Courant, K. Friedrichs, and H. Lewy. On the partial difference equations of mathematical physics. *IBM J. Res. Dev.*, 11(2):215–234, 1967.

[Cro10] R. Croce. *Numerische Simulation der Interaktion von inkompressiblen Zweiphasenströmungen mit Starrkörpern in drei Raumdimensionen.* Dissertation, Institut für Numerische Simulation, Universität Bonn, Bonn, Germany, 2010.

[CGS04] R. Croce, M. Griebel, and M.A. Schweitzer. A parallel level-set approach for two-phase flow problems with surface tension in three space dimensions. Preprint 157, Sonderforschungsbereich 611, Universität Bonn, 2004.

[CGS09] R. Croce, M. Griebel, and M.A. Schweitzer. Numerical simulation of bubble and droplet-deformation by a level set approach with surface tension in three dimensions. *International Journal for Numerical Methods in Fluids*, 62(9):963–993, 2009.

[CCS$^+$13] R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, and A.G. Gray. MLPACK: A scalable C++ machine learning library. *J. Mach. Learn. Res.*, 14(1):801–805, March 2013.

[DR84] P.J. Davis and P. Rabinowitz. *Methods of numerical integration.* Computer science and applied mathematics. Academic Press, 1984.

[DMSW05] S. De Marchi, R. Schaback, and H. Wendland. Near-optimal data-independent point locations for radial basis function interpolation. *Advances in Computational Mathematics*, 23(3):317–330, 2005.

[Dep13] Department of Transport and Main Roads. *Bridge Scour Manual.* Queensland Government, Australia, March 2013.

[DKS13] J. Dick, F.Y. Kuo, and I.H. Sloan. High-dimensional integration: The quasi-Monte Carlo way. *Acta Numerica*, 22:133–288, 5 2013.

[DGN98] T. Dornseifer, M. Griebel, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics, a Practical Introduction.* SIAM, Philadelphia, 1998.

[DV13] M. D'Elia and A. Veneziani. Uncertainty quantification for data assimilation in a steady incompressible Navier-Stokes problem. *ESAIM: Mathematical Modelling and Numerical Analysis*, 47:1037–1057, 7 2013.

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

[EG13] E.A.S. EL-Ghorab. Reduction of scour around bridge piers using a modified method for vortex reduction. *Alexandria Engineering Journal*, 52(3):467–478, 2013.

[Eld09] M.S. Eldred. Recent advances in non-intrusive polynomial chaos and stochastic collocation methods for uncertainty analysis and design. In *Structures, Structural Dynamics, and Materials Conference*. Sandia National Laboratories, Albuquerque, NM 87185, May 2009.

[EM 13] EM Photonics, Inc. CULA Reference manual, May 2013. Release R17.

[EL12] M. Emans and M. Liebmann. Efficient setup of aggregation AMG for CFD on GPUs. In *PARA*, pages 398–409, 2012.

[EMSU12] O.G. Ernst, A. Mugler, H.-J. Starkloff, and E. Ullmann. On the convergence of generalized polynomial chaos expansions. *ESAIM: Mathematical Modelling and Numerical Analysis*, 46:317–339, 3 2012.

[ES14] O.G. Ernst and B. Sprungk. Stochastic collocation for elliptic PDEs with random data: The lognormal case. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications - Munich 2012*, volume 97 of *Lecture Notes in Computational Science and Engineering*, pages 29–53. Springer International Publishing, 2014.

[ENS12] K.P. Esler, V. Natoli, and A. Samardzic. GAMPACK (GPU accelerated algebraic multigrid package). In *ECMOR XIII - 13th European Conference on the Mathematics of Oil Recovery*, September 2012.

[FY02] R.D. Falgout and U.M. Yang. hypre: A library of high performance preconditioners. In P.M.A. Sloot, A.G. Hoekstra, C.J.K. Tan, and J.J. Dongarra, editors, *Computational Science — ICCS 2002*, volume 2331 of *Lecture Notes in Computer Science*, pages 632–641. Springer Berlin Heidelberg, 2002.

[Far11] R. Farber. *CUDA Application Design and Development*. Applications of GPU computing. Morgan Kaufmann, 2011.

[FHW12] G. Fasshauer, F. Hickernell, and H. Woźniakowski. On dimension-independent rates of convergence for function approximation with Gaussian kernels. *SIAM Journal on Numerical Analysis*, 50(1):247–271, 2012.

[Fas07] G.E. Fasshauer. *Meshfree Approximation Methods with MATLAB*. Interdisciplinary mathematical sciences. World Scientific, 2007.

[Fas11] G.E. Fasshauer. Positive definite kernels: Past, present and future. *Dolomites Research Notes on Approximation*, 4(Special Issue on Kernel Functions and Meshless Methods):21–63, 2011.

[Fas12] G.E. Fasshauer. Green's functions: Taking another look at kernel approximation, radial basis functions, and splines. In M. Neamtu and L. Schumaker, editors, *Approximation Theory XIII: San Antonio 2010*, volume 13 of *Springer Proceedings in Mathematics*, pages 37–63. Springer New York, 2012.

[FY13a] G.E. Fasshauer and Q. Ye. A kernel-based collocation method for elliptic partial differential equations with random coefficients. In J. Dick, F.Y. Kuo, G.W. Peters, and I.H. Sloan, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2012*, volume 65 of *Springer Proceedings in Mathematics & Statistics*, pages 331–347. Springer Berlin Heidelberg, 2013.

[FY13b] G.E. Fasshauer and Q. Ye. Kernel-based collocation methods versus Galerkin finite element methods for approximating elliptic stochastic partial differential equations. In M. Griebel and M.A. Schweitzer, editors, *Meshfree Methods for Partial Differential Equations VI*, volume 89 of *Lecture Notes in Computational Science and Engineering*, pages 155–170. Springer Berlin Heidelberg, 2013.

[FST05] P. Frauenfelder, C. Schwab, and R.A. Todor. Finite elements for elliptic problems with stochastic coefficients. *Computer Methods in Applied Mechanics and Engineering*, 194(2–5):205 – 228, 2005.

[FHN$^+$13] E.J. Fuselier, T. Hangelbroek, F.J. Narcowich, J.D. Ward, and G.B. Wright. Localized bases for kernel spaces on the unit sphere. *SIAM J. Numerical Analysis*, 51(5):2538–2562, 2013.

[GKW$^+$08] B. Ganis, H. Klie, M.F. Wheeler, T. Wildey, I. Yotov, and D. Zhang. Stochastic collocation and mixed finite elements for flow in porous media. *Computer Methods in Applied Mechanics and Engineering*, 197(43–44):3547–3559, 2008. Stochastic Modeling of Multiscale and Multiphysics Problems.

[Gen72] W.M. Gentleman. Implementing Clenshaw-Curtis quadrature, I methodology and experience. *Commun. ACM*, 15(5):337–342, May 1972.

[GLS13] E.H. Georgoulis, J. Levesley, and F. Subhan. Multilevel sparse kernel-based interpolation using conditionally positive definite radial basis functions. In A. Cangiani, R.L. Davidchack, E. Georgoulis, A.N. Gorban, J. Levesley, and M.V. Tretyakov, editors, *Numerical Mathematics and Advanced Applications 2011*, pages 157–164. Springer Berlin Heidelberg, 2013.

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

[GS91] R.G. Ghanem and P.D. Spanos. *Stochastic finite elements: A spectral approach.* Springer-Verlag, New York, 1991.

[GBWT09] D. Göddeke, S.H.M. Buijssen, H. Wobker, and S. Turek. GPU acceleration of an unmodified parallel finite element Navier-Stokes solver. In W.W. Smari and J.P. McIntire, editors, *High Performance Computing & Simulation 2009*, pages 12–21, June 2009.

[Gou09] B. Gough. *GNU Scientific Library Reference Manual - Third Edition.* Network Theory Ltd., 3rd edition, 2009.

[GKN$^+$11] I.G. Graham, F.Y. Kuo, D. Nuyens, R. Scheichl, and I.H. Sloan. Quasi-Monte Carlo methods for elliptic PDEs with random coefficients and applications. *J. Comput. Phys.*, 230(10):3668–3694, 2011.

[GH14] M. Griebel and H. Harbrecht. Approximation of bi-variate functions: Singular value decomposition versus sparse grids. *IMA Journal of Numerical Analysis*, 34(1):28–54, 2014.

[GK14] M. Griebel and M. Klitz. Simulation of droplet impact with dynamic contact angle boundary conditions. In M. Griebel, editor, *Singular Phenomena and Scaling in Mathematical Models*, pages 297–325. Springer International Publishing, 2014.

[GMOS06] M. Griebel, B. Metsch, D. Oeltz, and M.A. Schweitzer. Coarse grid classification: A parallel coarsening scheme for algebraic multigrid methods. *Numerical Linear Algebra with Applications*, 13(2–3):193–214, 2006.

[GR14] M. Griebel and A. Rüttgers. Multiscale simulations of three-dimensional viscoelastic flows in a square-square contraction. *Journal of non-Newtonian Fluid Mechanics*, 205:41–63, 2014.

[GSZ92] M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, North Holland, 1992.

[GZ10] M. Griebel and P. Zaspel. A multi-GPU accelerated solver for the three-dimensional two-phase incompressible Navier-Stokes equations. *Computer Science - Research and Development*, 25(1–2):65–73, May 2010.

[Gri02] M. Grigoriu. *Stochastic Calculus: Applications in Science and Engineering.* Birkhäuser Boston, 2002.

[GR11] S. Gross and A. Reusken. *Numerical methods for two-phase incompressible flows*, volume 40 of *Springer Series in Computational Mathematics.* Springer Berlin / Heidelberg, 2011.

[GD07] N.A. Gumerov and R. Duraiswami. Fast radial basis function interpolation via preconditioned Krylov iteration. *SIAM J. Sci. Comput.*, 29(5):1876–1899, September 2007.

[HLDP10] G. Haase, M. Liebmann, C.C. Douglas, and G. Plank. A parallel algebraic multigrid solver on graphics processing units. In W. Zhang, Z. Chen, C.C. Douglas, and W. Tong, editors, *High Performance Computing and Applications*, volume 5938 of *Lecture Notes in Computer Science*, pages 38–47. Springer Berlin Heidelberg, 2010.

[Hac95] W. Hackbusch. *Integral equations: Theory and numerical treatment*, volume 120 of *International series of numerical mathematics*. Birkhäuser, Basel, 1995.

[Hal60] J.H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2:84–90, 1960. 10.1007/BF01386213.

[HPS12] H. Harbrecht, M. Peters, and R. Schneider. On the low-rank approximation by the pivoted Cholesky decomposition. *Applied Numerical Mathematics*, 62(4):428–440, 2012. Third Chilean Workshop on Numerical Analysis of Partial Differential Equations (WONAPDE 2010).

[HPS14] H. Harbrecht, M. Peters, and M. Siebenmorgen. Effcient approximation of random fields for numerical applications. Preprint 2014-01 Mathematisches Institut Universität Basel, 2014.

[HW65] F.H. Harlow and J.E. Welsh. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8, N. 10:2182–2189, 1965.

[Har07] M. Harris. Optimizing parallel reduction in CUDA. Technical report, NVIDIA Corporation, 2007.

[Hen07] A. Henderson. ParaView guide - A parallel visualization application, 2007. Kitware Inc.

[HY02] V.E. Henson and U.M. Yang. BoomerAMG: A parallel algebraic multigrid solver and preconditioner. *Appl. Numer. Math.*, 41(1):155–177, April 2002.

[HB10] J. Hoberock and N. Bell. Thrust: A parallel template library, 2010. Version 1.7.0.

[JS11] D.A. Jacobsen and I. Senocak. A full-depth amalgamated parallel 3d geometric multigrid solver for GPU clusters. *Proceedings of the 49th AIAA Aerospace Sciences Meeting, Orlando, Florida, USA*, page , 2011.

[JTS10] D.A. Jacobsen, J.C. Thibault, and I. Senocak. An MPI-CUDA implementation for massively parallel incompressible flow computations on multi-GPU clusters. *Proceedings of the 48th AIAA Aerospace Sciences Meeting and Exhibit, Orlando, Florida, USA*, page , 2010.

[JNX13] J.D. Jakeman, A Narayan, and D. Xiu. Minimal multi-element stochastic collocation for uncertainty quantification of discontinuous functions. *Journal of Computational Physics*, 242(0):790–808, 2013.

[JR13] J.D. Jakeman and S.G. Roberts. Local and dimension adaptive stochastic collocation for uncertainty quantification. In J. Garcke and M. Griebel, editors, *Sparse Grids and Applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 181–203. Springer Berlin Heidelberg, 2013.

[Jak08] H.A. Jakobsen. *Chemical Reactor Modeling: Multiphase Reactive Flows.* Springer, 2008.

[KFL00] M. Kang, F. Fedkiw, and X.-D. Liu. A boundary condition capturing method for multiphase incompressible flow. *J. Sci. Comput.*, 15(3):323–360, 2000.

[KBU05] N. Kantarci, F. Borak, and K.O. Ulgen. Bubble column reactors. *Process Biochemistry*, 40(7):2263–2283, 2005.

[Kel09] J. Kelly. GPU-accelerated simulation of two-phase incompressible fluid flow using a level-set method for interface capturing. *ASME Conference Proceedings*, 2009(43826):2221–2228, 2009.

[Kie08] A. Kielmann. Lanczos-Verfahren bei exponentiell abfallenden Eigenwerten. Master's thesis, Christian-Albrechts-Universität zu Kiel, Institut für Informatik, Dezember 2008.

[KH10] D.B. Kirk and W.-M.W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[KL06] O.M. Knio and O.P. Le Maître. Uncertainty propagation in CFD using polynomial chaos decomposition. *Fluid Dynamics Research*, 38(9):616–640, 2006. Recent Topics in Computational Fluid Dynamics.

[KHRVed] D.P. Kouri, M. Heinkenschloos, D. Ridzal, and B. G. Van Bloemen Waanders. A trust-region algorithm with adaptive stochastic collocation for PDE optimization under uncertainty. *SISC*, 09/2012 2014, submitted.

[KF12] J. Kraus and M. Förster. Facing the multicore-challenge II. chapter Efficient AMG on Heterogeneous Systems, pages 133–146. Springer-Verlag, Berlin, Heidelberg, 2012.

[KFBS13] J. Kraus, M. Förster, T. Brandes, and T. Soddemann. Using LAMA for efficient AMG on hybrid clusters. *Computer Science - Research and Development*, 28(2-3):211–220, 2013.

[Kri51] D.G. Krige. A statistical approach to some basic mine valuation problems on the Witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, December 1951.

[KCLL11] S.-H. Kuo, P.-H. Chiu, R.-K. Lin, and Y.-T. Lin. GPU implementation for solving incompressible two-phase flows. *World Academy of Science, Engineering and Technology*, pages 878–886, 2011.

[Lav67] M.M. Lavrent'ev. *Some Improperly Posed Problems of Mathematical Physics.* Springer tracts in natural philosophy. Springer, 1967.

[LMK10] O.P. Le Maître and O.M. Knio. *Spectral Methods for Uncertainty Quantification: With Applications to Computational Fluid Dynamics.* Scientific Computation. Springer, Dordrecht, 2010.

[LK04] L. Ling and E.J. Kansa. Preconditioning for radial basis functions with domain decomposition methods. *Mathematical and Computer Modelling*, 40(13):1413 – 1427, 2004.

[LS09] L. Ling and R. Schaback. An improved subspace selection algorithm for meshless collocation methods. *International Journal for Numerical Methods in Engineering*, 80(13):1623–1639, 2009.

[LOC94] X.-D. Liu, S. Osher, and T. Chan. Weighted essentially non-oscillatory schemes. *Journal of Computational Physics*, 115(1):200–212, 1994.

[Llo82] S. Lloyd. Least squares quantization in PCM. *Information Theory, IEEE Transactions on*, 28(2):129–137, Mar 1982.

[Loe78] M. Loeve. *Probability Theory II.* Graduate texts in mathematics. Springer-Verlag, 1978.

[LB08] G.J.A. Loeven and H. Bijl. Airfoil analysis with uncertain geometry using the probabilistic collocation method. In *Proceedings of the 49th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Schaumburg, IL, USA, April 2008.

[LWB07] G.J.A. Loeven, J.A.S. Witteveen, and H. Bijl. A probabilistic radial basis function approach for uncertainty quantification. In J. Benek and J. Hirsch, C. Luckring, editors, *Proceedings of the NATO RTO-MP-AVT-147 Computational Uncertainty in Military Vehicle design symposium*, pages 1–12, 2007.

[Luk12] D. Lukarski. *Parallel Sparse Linear Algebra for Multi-core and Many-core Platforms.* PhD thesis, Karlsruher Instituts für Technologie (KIT), 2012.

[Luk14] D. Lukarski. PRALUTION user manual, May 2014. Version 0.7.0.

[Mar03] G. Marsaglia. Xorshift RNGs. *Journal of Statistical Software*, 8(14):1–6, 7 2003.

[MX09] Y. Marzouk and D. Xiu. A stochastic collocation approach to Bayesian inference in inverse problems. *Communications in Computational Physics*, 6(4):826–847, 2009.

[MHZ05] L. Mathelin, M.Y. Hussaini, and T.A. Zang. Stochastic approaches to uncertainty quantification in CFD simulations. *Numerical Algorithms*, 38(1-3):209–236, 2005.

[Mat63] G. Matheron. Principles of geostatistics. *Economic Geology*, 58(8):1246–1266, 1963.

[Mer] D. Merrill. Homepage of the CUB library. `http://nvlabs.github.io/cub`. last access: 2014/10/02.

[MGG12] D. Merrill, M. Garland, and A. Grimshaw. Scalable GPU graph traversal. In *Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '12, pages 117–128, New York, NY, USA, 2012. ACM.

[Mes94] Message Passing Forum. MPI: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[Mic09] P. Micikevicius. 3d finite difference computation on GPUs using CUDA. In *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 79–84, New York, NY, USA, 2009. ACM.

[MC94] W.J. Morokoff and R.E. Caflisch. Quasi-random sequences and their discrepancies. *SIAM J. Sci. Comput*, 15:1251–1279, 1994.

[MGM$^+$11] A. Munshi, B. Gaster, T.G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, first edition, July 2011.

[MWB$^+$11] A. Murarasu, J. Weidendorfer, G. Buse, D. Butnaru, and D. Pflüger. Compact data structure and scalable algorithms for the sparse grid technique. In *Proceedings of the 16th ACM Symposium on Principles and Practice of Parallel Programming*, PPoPP '11, pages 25–34, New York, NY, USA, 2011. ACM.

[NJ14] A. Narayan and J. Jakeman. Adaptive Leja sparse grid constructions for stochastic collocation and high-dimensional approximation. *ArXiv e-prints*, April 2014.

[Nie87] H. Niederreiter. Point sets and sequences with small discrepancy. *Monatshefte für Mathematik*, 104(4):273–337, 1987.

[Nie92] H. Niederreiter. *Random Number Generation and quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[NTW08a] F. Nobile, R. Tempone, and C. Webster. An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 46(5):2411–2442, 2008.

[NTW08b] F. Nobile, R. Tempone, and C. Webster. A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM J. Numer. Anal.*, 46(5):2309–2345, May 2008.

[NVI10] NVIDIA Corporation. NVIDIA CUDA Toolkit, 2010. Version 3.2.

[NVI14a] NVIDIA Corporation. NVIDIA CUDA C Best Practice Guide, 2014. Version 6.0.

[NVI14b] NVIDIA Corporation. NVIDIA CUDA C Programming Guide, 2014. Version 6.0.

[OS88] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, 79(1):12–49, 1988.

[OLG$^+$07] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, March 2007.

[PLW$^+$07] G. Plank, M. Liebmann, R. Weber dos Santos, E.J. Vigmond, and G. Haase. Algebraic multigrid preconditioner for the cardiac bidomain model. *IEEE Trans. Biomed. Engineering*, 54(4):585–596, 2007.

[PWB$^+$12] S. Potluri, H. Wang, D. Bureddy, A.K. Singh, C. Rosales, and D.K. Panda. Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication. In *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, pages 1848–1857, May 2012.

[PS12] C. Powell and D. Silvester. Preconditioning steady-state Navier–Stokes equations with random data. *SIAM Journal on Scientific Computing*, 34(5):A2482–A2506, 2012.

[RW05] C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005.

[RZ09] C. Rieger and B. Zwicknagl. Deterministic error analysis of support vector regression and related regularized kernel methods. *J. Mach. Learn. Res.*, 10:2115–2132, December 2009.

[Roa97] P.J. Roache. Quantification of uncertainty in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 29(1):123–160, 1997.

[RBDV10] L. Rosasco, M. Belkin, and E. De Vito. On learning with integral operators. *J. Mach. Learn. Res.*, 11:905–934, March 2010.

[RS] J.W. Ruge and K. Stüben. *4. Algebraic Multigrid*, chapter 4, pages 73–130.

[Ruz04] M. Ruzicka. *Nichtlineare Funktionalanalysis: Eine Einführung.* Springer-Lehrbuch Masterclass. Springer Berlin Heidelberg, 2004.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edition, 2003.

[SK10] J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming.* Addison-Wesley Professional, 1st edition, 2010.

[SM11] S. Sankaran and A.L. Marsden. A stochastic collocation method for uncertainty quantification and propagation in cardiovascular simulations. *J Biomech Eng*, 133(3):031001, Mar 2011.

[Sch95] R. Schaback. Comparison of radial basis function interpolants. In *In Multivariate Approximation. From CAGD to Wavelets*, pages 293–305. World Scientific, 1995.

[Sch13] R. Schaback. Greedy sparse linear approximations of functionals from nodal data. *Numerical Algorithms*, pages 1–17, 2013.

[SW00] R. Schaback and H. Wendland. Adaptive greedy techniques for approximate solution of large RBF systems. *Numerical Algorithms*, 24(3):239–254, 2000.

[Sch11] M. Schick. *Uncertainty Quantification for Stochastic Dynamical Systems: Spectral Methods using Generalized Polynomial Chaos*. PhD thesis, Engineering Mathematics and Computing Lab (EMCL) – Karlsruhe Institute of Technology (KIT), 2011.

[SHL14] M. Schick, V. Heuveline, and O.P. Le Maître. A Newton–Galerkin method for fluid flow exhibiting uncertain periodic dynamics. *SIAM/ASA Journal on Uncertainty Quantification*, 2(1):153–173, 2014.

[SL14] B. Schieche and J. Lang. Adjoint error estimation for stochastic collocation methods. In J. Garcke and D. Pflüger, editors, *Sparse Grids and Applications - Munich 2012*, volume 97 of *Lecture Notes in Computational Science and Engineering*, pages 271–293. Springer International Publishing, 2014.

[SAH00] W.J. Schroeder, L.S. Avila, and W. Hoffman. Visualizing with VTK: A tutorial. *IEEE Comput. Graph. Appl.*, 20(5):20–27, September 2000.

[SG11] C. Schwab and C.J. Gittelson. Sparse tensor discretizations of high-dimensional parametric and stochastic PDEs. *Acta Numerica*, 20:291–467, 5 2011.

[ST06] C. Schwab and R.A. Todor. Karhunen-Loève approximation of random fields by generalized fast multipole methods. *J. Comput. Phys.*, 217(1):100–122, September 2006.

[Sch90] H.A. Schwarz. *Gesammelte Mathematische Abhandlungen*, volume 2, pages 133–143. Springer Verlag, Berlin, Deutschland / Heidelberg, Deutschland / London, UK / etc., 1890.

[Sem12] A. Semechko. Uniform sampling of a sphere. Matlab File Exchange `http://www.mathworks.com/matlabcentral/fileexchange`, June 2012. last access: 2014/12/14.

[SB96] M.-Z. Shao and N. Badler. Spherical sampling by Archimedes' theorem. Technical Report 184, University of Pensylvania, Department of Computer and Information Science, 1996.

[SBG96] B.F. Smith, P.E. Bjørstad, and W.D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, NY, USA, 1996.

[Smo63] S. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics, Doklady*, 4:240–243, 1963.

[Sob67] I.M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[Son89] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52, 1989.

[Sør12] H.H.B. Sørensen. High-performance matrix-vector multiplication on the GPU. In *Proceedings of the 2011 International Conference on Parallel Processing*, Euro-Par'11, pages 377–386, Berlin, Heidelberg, 2012. Springer-Verlag.

[Ste99] M.L. Stein. *Interpolation of Spatial Data: Some Theory for Kriging.* Springer Series in Statistics. Springer New York, 1999.

[SCM12] M.L. Stein, J. Chen, and A. Mihai. Difference filter preconditioning for large covariance matrices. *SIAM J. Matrix Analysis Applications*, 33(1):52–72, 2012.

[Sto04] J. Stoer. *Numerische Methematik 1.* Springer-Verlag, 9th edition, 2004.

[SB02] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Texts in Applied Mathematics. Springer, 2002.

[Str06] E. Strohmaier. Top500 Supercomputer. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[STCE06] J. Strybny, C. Thorenz, R. Croce, and M. Engel. A parallel 3d free surface Navier-Stokes solver for high performance computing at the German Waterways Administration. In *The 7th Int. Conf. on Hydroscience and Engineering (ICHE-2006)*, Philadelphia, USA, September 2006.

[Sud08] B. Sudret. Global sensitivity analysis using polynomial chaos expansions. *Reliability Engineering & System Safety*, 93(7):964–979, 2008.

[SB13] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets - uncertainty quantification test problems. `http://www.sfu.ca/~ssurjano/uq.html`, 2013. last access: 2014/06/18.

[SF99] M. Sussman and E. Fatemi. An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow. *SIAM J. Sci. Comput.*, 20:1165–1191, February 1999.

[SSO94] M. Sussman, P. Smereka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114:146–159, 1994.

[SSH$^+$07] M. Sussman, K.M. Smith, M.Y. Hussaini, M. Ohta, and R. Zhi-Wei. A sharp interface method for incompressible two-phase flows. *Journal of Computational Physics*, 221(2):469–505, 2007.

[TLN14] L. Tamellini, O. Le Maître, and A. Nouy. Model reduction based on proper generalized decomposition for the stochastic steady incompressible Navier–Stokes equations. *SIAM Journal on Scientific Computing*, 36(3):A1089–A1117, 2014.

[Tec10] Mellanox Techologies. NVIDIA GPUDirect Technology - Accelerating GPU-based systems. Whitepaper, May 2010.

[TJWG14] A.L. Teckentrup, P. Jantsch, C.G. Webster, and M. Gunzburger. A multilevel stochastic collocation method for partial differential equations with random input data. *ArXiv e-prints*, April 2014.

[TSLV11] K.E. Teigen, P. Song, J. Lowengrub, and A. Voigt. A diffuse-interface method for two-phase flows with soluble surfactants. *J. Comput. Phys.*, 230(2):375–393, January 2011.

[TS09] J.C. Thibault and I. Senocak. CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows. In *Proceedings of the 47th AIAA Aerospace Sciences Meeting*, Orlando, Florida, USA, 2009.

[TA77] A.N. Tikhonov and V.I.A. Arsenin. *Solutions of ill-posed problems.* Scripta series in mathematics. Winston, 1977.

[TDB10] S. Tomov, J. Dongarra, and M. Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5–6):232–240, 2010.

[Top11] Top500 list of supercomputers, June 2011.

[TB97] L.N. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM: Society for Industrial and Applied Mathematics, 1997.

[TS01] U. Trottenberg and A. Schuller. *Multigrid.* Academic Press, Inc., Orlando, FL, USA, 2001.

[TPME11] R.S. Tuminaro, E.T. Phipps, C.W. Miller, and H.C. Elman. Assessment of collocation and Galerkin approaches to linear diffusion equations with random data. *International Journal for Uncertainty Quantification*, 1(1):19–33, 2011.

[vK04] W.C.M. van Beers and J.P.C. Kleijnen. Kriging interpolation in simulation: A survey. In *Proceedings of the Winter Simulation Conference 2004*, volume 1, pages 113–121, Dec 2004.

[van14] H.-W. van Wyk. Multilevel sparse grid methods for elliptic partial differential equations with random coefficients. *ArXiv e-prints*, April 2014.

[VCG$^+$08] B. Verleye, R. Croce, M. Griebel, M. Klitz, S.V. Lomov, G. Morren, H. Sol, I. Verpoest, and D. Roose. Permeability of textile reinforcements: Simulation, influence of shear, validation. *Composites Science and Technology*, 68(13):2804–2810, October 2008.

[von05] G. von Winckel. Fast Clenshaw-Curtis quadrature. The Mathworks Central File Exchange, Feb. 2005.

[Vra12] Vratis Ltd. SpeedIT 2.3 reference manual, November 2012.

[WHCX13] L. Wang, X. Hu, J. Cohen, and J. Xu. A parallel auxiliary grid algebraic multigrid method for graphic processing units. *SIAM Journal on Scientific Computing*, 35(3):C263–C283, 2013.

[Wan11] Q. Wang. *Uncertainty Quantification for Unsteady Fluid Flow Using Adjoint-based Approaches*. Proquest, Umi Dissertation Publishing, 2011.

[WA11] X. Wang and T. Aoki. Multi-GPU performance of incompressible flow computation by lattice Boltzmann method on GPU cluster. *Parallel Computing*, 37(9):521–535, 2011. Emerging Programming Paradigms for Large-Scale Scientific Computing.

[Wen95] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *Advances in Computational Mathematics*, 4(1):389–396, 1995.

[Wen04] H. Wendland. *Scattered Data Approximation*. Cambridge University Press, 2004.

[Wen06] H. Wendland. Computational aspects of radial basis function approximation. In K. Jetter, M.D. Buhmann, W. Haussmann, R. Schaback, and J. Stöckler, editors, *Topics in Multivariate Approximation and Interpolation*, volume 12 of *Studies in Computational Mathematics*, pages 231–256. Elsevier, 2006.

[WH13] D. Wirtz and B. Haasdonk. A vectorial kernel orthogonal greedy algorithm. In *Proceedings of the DWCAA12*, volume 6, pages 83–100, 2013.

[WS92] Z.-m. Wu and R. Schaback. Local error estimates for radial basis function interpolation of scattered data. *IMA J. Numer. Anal*, 13:13–27, 1992.

[Xiu09] D. Xiu. Fast numerical methods for stochastic computations: A review. *Communications in Computational Physics*, 5(2-4):242–272, 2009.

[Xiu10] D. Xiu. *Numerical Methods for Stochastic Computations: A Spectral Method Approach*. Princeton University Press, Princeton, NJ, 2010.

[XK02] D. Xiu and G. Karniadakis. The Wiener–Askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, 2002.

[Yan06] U.M. Yang. Parallel algebraic multigrid methods – High performance preconditioners. In A. Bruaset and A. Tveito, editors, *Numerical Solution of Partial Differential Equations on Parallel Computers*, volume 51 of *Lecture Notes in Computational Science and Engineering*, pages 209–236. Springer Berlin Heidelberg, 2006.

[Yin06] L. Ying. Short note: A kernel independent fast multipole algorithm for radial basis functions. *J. Comput. Phys.*, 213(2):451–457, April 2006.

[YBK09] R. Yokota, L.A. Barba, and M.G. Knepley. PetRBF–A parallel O(N) algorithm for radial basis function interpolation. *CoRR*, abs/0909.5413, 2009.

[ZG11] P. Zaspel and M. Griebel. Photorealistic visualization and fluid animation: Coupling of Maya with a two-phase Navier-Stokes fluid solver. *Computing and Visualization in Science*, 14(8):371–383, 2011.

[ZG13] P. Zaspel and M. Griebel. Solving incompressible two-phase flows on multi-GPU clusters. *Computers & Fluids*, 80(0):356–364, 2013.

[ZLY$^+$13] G. Zhang, D. Lu, M. Ye, M. Gunzburger, and C. Webster. An adaptive sparse-grid high-order stochastic collocation method for Bayesian inference in groundwater reactive transport modeling. *Water Resources Research*, 49(10):6871–6892, 2013.