Multi-GPU Parallel Numerical Methods for Uncertainty Quantification in Computational Fluid Dynamics

Michael Griebel  Christian Rieger  Peter Zaspel

Institute for Numerical Simulation
Rheinische Friedrich-Wilhelms-Universität Bonn

GPU Technology Conference 2013, San José, CA
March 18-21, 2013
Application: Flows through hydraulic constructions

Real-world sluice system was successfully modeled and simulated

Joint project with German Federal Waterways Engineering and Research Institute

computationally challenging by its own
Two-phase Navier-Stokes equations

PDE system

\[ \rho(\phi)(\partial_t \mathbf{u} + (\mathbf{u} \cdot \nabla)\mathbf{u}) = \nabla \cdot (\mu(\phi) S) - \nabla p \]
\[ -\sigma \kappa(\phi) \delta(\phi) \nabla \phi + \rho(\phi) g \]
\[ \nabla \cdot \mathbf{u} = 0 \]
\[ \partial_t \phi + \mathbf{u} \cdot \nabla \phi = 0 \]

- \( \mathbf{u} \) fluid velocity
- \( \mu \) dynamic viscosity
- \( \kappa \) local curvature of fluid surface
- \( \rho \) density
- \( \mu \) dynamic viscosity
- \( \sigma \) surface tension
- \( \delta \) Dirac delta-functional
- \( \phi \) level set function
- \( \nabla \) gradient
- \( g \) volume forces
- \( S := \nabla \mathbf{u} + \{\nabla \mathbf{u}\}^T \)

\[ \rho(\phi) := \rho_2 + (\rho_1 - \rho_2) H(\phi) \]
\[ \mu(\phi) := \mu_2 + (\mu_1 - \mu_2) H(\phi) \]
\[ H(\phi) := \begin{cases} 
0 & \text{if } \phi < 0 \\
\frac{1}{2} & \text{if } \phi = 0 \\
1 & \text{if } \phi > 0 
\end{cases} \]
Uncertainty Quantification (UQ) in CFD

Current standard
CFD simulations for fixed and known input parameters

Problem
- nature phenomena: input data not known exactly
- engineering: constructions / measurements always subject to perturbations

Solution
numerical modeling and analysis of uncertainties ⇒ Uncertainty Quantification

Major industrial relevance
High-level idea of standard UQ methods

Algorithmic idea:

1. sampling of stochastic input parameters according to some distribution
2. computation of hundreds or thousands of stochastic realizations (high-resolution simulations)
3. extraction of averaged data (expectation value), variance information, ... as post-processing step

**non-intrusive method** $\Rightarrow$ all available solvers usable without reprogramming

Challenge:

- computation of realizations highly parallel but extremely time-intensive
- handling of enormous data quantities in post-processing step

optimal setting for multi-GPU clusters in Exascale
Stochastic formulation of the Navier-Stokes equations

Find stochastic functions $u : \overline{D} \times \mathbb{R}^+ \times \Omega \rightarrow \mathbb{R}^3$, $p : \overline{D} \times \mathbb{R}^+ \times \Omega \rightarrow \mathbb{R}$ such that $P$-almost surely:

$$\frac{\partial}{\partial t} u(x, t, \omega) + u(x, t, \omega) \cdot \nabla u(x, t, \omega) + \frac{1}{\rho(\omega)} (\nabla p(x, t, \omega) - \mu(\omega) \Delta u(x, t, \omega)) = f(x, \omega)$$
on $D$

$$\nabla \cdot u(x, t, \omega) = 0$$ on $D$

$$u(x, t, \omega) = g(x, \omega)$$ on $\partial D$

- $x \in D \subset \mathbb{R}^3$ physical domain
- $(\Omega, \mathcal{F}, P)$ complete probability space
- $\Omega$ set of outcomes, $\mathcal{F} \subset 2^\Omega$ $\sigma$-algebra of events, $P : \mathcal{F} \rightarrow [0, 1]$ probability measure
- $\rho, \mu : \Omega \rightarrow \mathbb{R}$, $f, g : \Omega \rightarrow \mathbb{R}^3$ stochastic functions
Formulation for general setting
Find stochastic functions $u : \Omega \times \overline{D} \times [0, T] \rightarrow S$, $\overline{D} \subset \mathbb{R}^d$, $S \subset \mathbb{R}^s$, such that $P$-almost surely:

$$
\mathcal{L}(a(\omega, x, t))u(\omega, x, t) = f(\omega, x, t) \quad \text{in } \overline{D} \times [0, T] \\
+ BC
$$

- $\mathcal{L}$ general operator depending on parameters modeled by random field $a$
- $a, f : \Omega \times \overline{D} \times [0, T] \rightarrow \mathbb{R}^s$ stochastic function
- $x \in D \subset \mathbb{R}^3$ physical domain
- $(\Omega, \mathcal{F}, P)$ complete probability space
- $\Omega$ set of outcomes, $\mathcal{F} \subset 2^\Omega$ $\sigma$-algebra of events, $P : \mathcal{F} \rightarrow [0, 1]$ probability measure
Outline

1. Introduction
2. Multi-GPU parallel solver for the two-phase Navier-Stokes equations
3. RBF kernel methods for non-intrusive stochastic collocation
4. Fast approximation of Karhunen-Loève decompositions for large-scale problems
5. Numerical results
6. Summary
NaSt3DGPF - A 3D two-phase Navier-Stokes solver

3D grid-based fluid solver for the two-phase incompressible Navier-Stokes equations for simulation of two interacting fluids (e.g. air, water)

- FD solver based on Chorin’s pressure projection approach
- level-set for interface description with continuum surface force method
- high-order space discretizations: e.g. WENO 5th,
- time discretizations: Runge-Kutta 3rd, Adams-Bashforth
- Jacobi-preconditioned conjugate gradient solver for Poisson equation
- complex geometries with different boundary conditions
- multi-GPU MPI parallelization by domain decomposition

GPU implementation

General
- CUDA as GPU programming framework, double precision
- Linearization of 3D data fields

Multi-GPU parallelization

What remains on CPU?
- Configuration file parser, I/O, parallel communication

Overlapping comm. and computation
- Matrix-vector product on inner cells
- Exchange boundary data
- Matrix-vector product on boundary cells
- Results
General performance

<table>
<thead>
<tr>
<th>Simulation Grid Resolution</th>
<th>Speed-up on one GPU</th>
<th>Power consumption in kWh</th>
</tr>
</thead>
<tbody>
<tr>
<td>64³</td>
<td>1.61, 1.63</td>
<td>0.21, 0.12, 0.09, 0.08</td>
</tr>
<tr>
<td>128³</td>
<td>2.24, 2.57</td>
<td>0.21, 0.12, 0.09, 0.08</td>
</tr>
<tr>
<td>256 × 128²</td>
<td>2.26, 2.86</td>
<td>0.21, 0.12, 0.09, 0.08</td>
</tr>
<tr>
<td>256² × 128</td>
<td>3.01, 3.26</td>
<td>0.21, 0.12, 0.09, 0.08</td>
</tr>
</tbody>
</table>

- GT200 GPU vs. 6-core Xeon CPU
- GF100 GPU w. ECC vs. dual 6-core Xeon CPU
- GF100 GPU w/o ECC vs. dual 6-core Xeon CPU
Multi-GPU performance (Fermi GPUs)

Strong scaling / speed-up relative to one Fermi GPU

Weak scaling efficiency (in percent) relative to one GPU

strong scaling / speed-up

weak scaling / scale-up
Further improvement: Algebraic multigrid (AMG) method

Motivation

- mesh width dependence Jacobi-preconditioned CG solver for Poisson equation
- need optimal $O(1)$ convergence rate and linear complexity

Classical Ruge-Stüben AMG

- purely sequential coarsening / fine grid classification
- hard to parallelize, very few results for GPUs
Current implementation of AMG

Features

- implementation on top of CUSP
- standard V-cycle
- (damped) Jacobi smoothers / some more fancy experimental smoothers
- direct / standard / Jacobi interpolation (w. / w/o truncation)
- coarse grid solver: CULA (GPU Lapack)
- C/F classification CPU-based / some experimental GPU versions

Preliminary results (3D Poisson, CPU C/F class., 1 CPU core vs. 1 GPU)

- parallel solve phase 5.7x (mainly based on standard CUSP implementation)
- parallelized setup phase 3.9x
Outline

1 Introduction
2 Multi-GPU parallel solver for the two-phase Navier-Stokes equations
3 **RBF kernel methods for non-intrusive stochastic collocation**
4 Fast approximation of Karhunen-Loève decompositions for large-scale problems
5 Numerical results
6 Summary
Stochastic collocation using a Lagrange basis

Finite set of collocation points

\[ X_\Gamma := \{y_1, \ldots, y_{N_\Gamma}\} \subset \Gamma. \]

Solution of deterministic problems

\[ \mathcal{L}(a(y_i, x, t))u(y_i, x, t) = f(y_i, x, t) \quad \text{in } \bar{D} \times [0, T], \quad \forall i = 1 \ldots N_\Gamma. \]

Full solution \( u(y, x, y) \) approximated by interpolation with Lagrange basis

\[ u_{N_\Gamma}(y, x, t) \approx \sum_{i=1}^{N_\Gamma} u(y_i, x, t)L_i(y) \]

\( \{L_i\}_{i=1}^{N_\Gamma}, \quad L_i : \Gamma \rightarrow \mathbb{R}, \quad \text{with } L_i(y_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \)

Lagrange basis constructed using **radial basis functions**

\[ L_i(y) \in \text{span } \{k(||y - y_j||), j = 1, \ldots, N_\Gamma\} \]
Given: collocation points in stochastic space: \( X_\Gamma := \{ y_1, \ldots, y_{N_\Gamma} \} \subset \Gamma \)

Target: Lagrange basis \( \{ L_i(y) \}_{i=1}^{N_\Gamma} \) in space \( \mathcal{P}(\Gamma) := \text{span}\{ K(\cdot, y_i)|y_i \in X_\Gamma \} \)

\[ L_i(y) \in \mathcal{P}(\Gamma), \quad L_i(y_j) = \delta_{i,j}, \quad k(y_i, y_j) = k_0(\| y_i - y_j \|) \]

Evaluation of Lagrange basis at point \( q \in \Gamma \rightarrow L(q) \)

\[
A = \begin{pmatrix}
  k(y_1, y_1) & \cdots & k(y_1, y_{N_\Gamma}) \\
  \vdots & \ddots & \vdots \\
  k(y_{N_\Gamma}, y_1) & \cdots & k(y_{N_\Gamma}, y_{N_\Gamma})
\end{pmatrix}, \quad
L(q) = \begin{pmatrix}
  L_1(q) \\
  \vdots \\
  L_{N_\Gamma}(q)
\end{pmatrix}, \quad
R(q) = \begin{pmatrix}
  k(q, y_1) \\
  \vdots \\
  k(q, y_{N_\Gamma})
\end{pmatrix}
\]

Solution of a linear system for each \( q \)

\[
A L(q) = R(q)
\]
Numerical methods to solve $A L(q) = R(q)$

Current method of choice

- dense LU decomposition applied to many right-hand sides
- complexity with $N_e$ number of evaluations: $O(N_\Gamma^3 + N_\Gamma^2 N_e)$
- $N_\Gamma$ pretty small $\rightarrow$ number of simulations
- $N_e$ quite large $\rightarrow$ number of quadrature points in evaluation of $E[L_i]$

Outlook: More sophisticated methods?

- iterative linear solvers (many right-hand sides?)
- multi-grid preconditioner $\Rightarrow O(N_\Gamma^2)$
- fast multipole for matrix-vector product $\Rightarrow O(N_\Gamma \log N_\Gamma)$ $\leftarrow$ ExaFMM?
- truncation of kernel matrix / compactly supported kernels $\Rightarrow O(N_\Gamma)$
Collocation points / Quadrature

Choice of collocation points

Monte Carlo / Quasi Monte Carlo (Halton sequence) / Sparse grids (point locations)

Stochastical quantities of interest in Lagrange basis (e.g. expectation value)

\[
\mathbb{E}[u^{(h)}(x, t)] = \int_{\Omega_M} u^{(h)}(x, t, y) \rho_M(y) dy \approx \sum_{i=1}^{N(\Omega)} u^{(h)}(x, t, y_i) \mathbb{E}[L_i]
\]

Quadrature for expectation values of Lagrange basis ⇒ need \( \mathbb{E}[L_i] \) and \( \mathbb{E}[L_i L_j] \)

\[
\mathbb{E}[L_i] = \int_{\Omega_M} L_i(y) \rho_M(y) dy, \quad \mathbb{E}[L_i L_j] = \int_{\Omega_M} L_i(y)L_j(y) \rho_M(y) dy
\]

Quadrature methods

- Monte Carlo
- Quasi Monte Carlo: using Sobol sequence
GPU-parallel implementation of non-intrusive UQ method

Framework
- configuration files for independent fluid solver runs generated by Python scripts
- simulations performed with full multi-GPU code on 36 Nvidia Tesla M2090 cluster at our institute
- stochastics as post-processing on GPU using VTK data files produced by fluid solver

GPU programming
- Nvidia CUDA 5.0
- Thrust (transforms, reductions, . . .)
- CULA Dense R14 (LU factorization)
- CUBLAS (dot product with stride)
- GPU kernels (matrix setup, . . .)
Convergence results for QMC quadrature and QMC colloc. pts.

\[ \varphi_{\epsilon}(r) = e^{-(\epsilon r)^2} \]

\[ \varphi_{3,0}(r) = ((1 - r)^2)_+ \]

\[ \varphi_{3,1}(r) = ((1 - r)^4)_+(4r + 1) \]

\[ \varphi_{3,2}(r) = ((1-r)^6)_+(35r^2+18r+3) \]
Outline

1. Introduction
2. Multi-GPU parallel solver for the two-phase Navier-Stokes equations
3. RBF kernel methods for non-intrusive stochastic collocation
4. Fast approximation of Karhunen-Loève decompositions for large-scale problems
5. Numerical results
6. Summary
Karhunen-Loève expansion

\[ u(\omega, x) = \mathbb{E}[u](x) + \sum_{k=1}^{N_{KL}} \sqrt{\lambda_k} Y_k(\omega) \psi_k(x) \]

\[ Y_k(\omega) := \frac{1}{\sqrt{\lambda_k}} \int_{D} (u(\omega, x) - \mathbb{E}[u](x)) \psi_k(x) dx, \quad \mathbb{E}[Y_k(\omega)] = 0, \quad \mathbb{E}[Y_k Y_{k'}] = \delta_{k,k'} \]

**How to compute Karhunen-Loève expansion?**

\((\psi_k, \lambda_k)\) given as eigenpairs of operator

\[ C_{\text{Cov}[u]} : L^2(D) \to L^2(D), \quad f \mapsto \int_{D} \text{Cov}[u](\cdot, x) f(x) dx. \]

Possible to compute by Nyström discretization as eigenvalues / eigenfunctions of

\[ C \in \mathbb{R}^{N_D \times N_D}, \quad c_{ij} := \text{Cov}[u](x_i, x_j) \]
Lanczos iterative method for Eigenvalue decomposition of \( C \)

**Challenge**
- choice of sampling points: full set of PDE discretization points
  - \( \rightarrow \) huge matrix \( C, C \in \mathbb{R}^{\text{gridpoints} \times \text{gridpoints}} \)
- efficient & easy parallelizable method necessary

**Lanczos iterative method**
- Krylov subspace method for hermitian matrices
- iterative construction of tridiagonal matrix with subset of original eigenvalues
- Eigenvalue computation:
  1. compute matrix \( C_\ell \) by Lanczos algorithm
  2. extract eigenpairs \((\lambda_n(C_\ell), e_n(C_\ell))_{n=1,...,\ell}\) by tridiagonal QR method
  3. transform results \( e_{n_\ell} = K_\ell e_n(C_\ell), \lambda_{n_\ell} = \lambda_n(C_\ell) \)

**Computational complexity**
- Lanczos for dense input matrices: \( O(N_{NY}^2 \cdot \ell) \)
- QR method for tridiagonal systems: \( O(\ell) \)
Multi-GPU parallel implementation

Covariance matrix setup

implemented in multi-GPU parallel way

Large-scale eigenvalue decomposition bei multi-GPU library parla

- self-built library for multi-GPU (dense) linear algebra
- operations: matrix-vector, scalar product, ...
- local matrix-vector, scalar products: cuBLAS
- parallelization by matrix-row decomposition & MPI
- algorithms: Lanczos iterative method with full reorthogonalization
- tridiagonal QR method: CULADense (single-GPU)
- usable without any GPU knowledge

Current limitation

low scalability, however currently only memory limited
Numerical results

- **Covariance spectrum approximation**
  - Index of the eigenmode vs. size of the eigenmode
  - Graph showing data for different numbers of realizations (8, 16, 32, 64, 128, 256, 512) and Lanczos
  - Relative error norms $L^\infty$ and $L^1$

- **Karhunen-Loève reconstruction error**
Outline

1 Introduction

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

3 RBF kernel methods for non-intrusive stochastic collocation

4 Fast approximation of Karhunen-Loève decompositions for large-scale problems

5 Numerical results

6 Summary
Visual results: Uncertainty in reattachment length.
Uncertainty quantification for vortex reattachment length

distribution

\[ u_{inflow} \sim \mathcal{U}(0.1, 1.0) \]
\[ \mu \sim \mathcal{U}(0.001, 1.0) \]
\[ \rho \sim \mathcal{U}(500, 1000) \]

reattachment length

\[ l_{reattach} = x_{end} - x_{step} \]

expected value

\[ \sim 3.39m \text{ at } t = 1.0s \]
Full 3D two-phase result: Mixing in bubble reactors
Fluid flow as large scale application in uncertainty quantification

(Multi-)GPU parallelization necessary for usability of real-world UQ

Optimal RBF kernel-based numerical methods with fast convergence

Thank you!
Backup Slides
# Benchmarking platforms

## CPU Hardware
- Dual-6-core Intel Xeon E5650 CPU 2.67 GHz
- 24 GB DDR3-RAM

## GPU Cluster (8 GT200 GPUs)
- 2 workstations with
  - 4-core Intel Core i7-920 CPU 2.66 GHz
  - 12 GB DDR3-RAM
  - NVIDIA Tesla S1070 (4 GPUs)
- Ubuntu Linux 10.04 64 bit
- GCC 4.4.3 compiler, CUDA 3.2 SDK, OpenMPI 1.4.1

## GPU Hardware (GF100 Fermi)
- 4-core Intel Xeon E5620 CPU 2.40 GHz
- 6 GB DDR3-RAM
- NVIDIA Tesla C2050 GPU

## GPU Cluster (48 GF100 GPUs) @ RWTH Aachen University, Germany
- 24 double-GPU systems with
  - 2 Intel Xeon X5650 CPUs @ 2.66 GHz
  - 2 NVIDIA Quadro 6000 GPUs
  - QDR InfiniBand
- Scientific Linux 6.1, GCC 4.4.5, OpenMPI 1.5.3, CUDA 4.0.17
Performance measurements for GPUs

Perfectly fair CPU-GPU benchmarks are very hard!

1 GPU vs. 1 CPU core
- + good GPU results
- – CPU speed unclear
- – – not realistic wrt. price

1 GPU vs. 1 CPU socket
- + better price realism
- – # of cores per socket?
- – speed per CPU core?

Performance per dollar
- ++ best price realism
- – price per node / CPU?
- – prices subject to changes

Performance per Watt
- ++ Green IT
- + power ≡ costs
- – high influence of config.
Core technique for two-phase flows: Level-set method

Level-set method

Representation of a free surface \( \Gamma_t \) by a signed distance function \( \phi \in \mathbb{R}^3 \times \mathbb{R} \rightarrow \mathbb{R} \):

\[
\Gamma_t = \{ x \mid \phi(x, t) = 0 \}
\]

\[|\nabla \phi| = 1\]

Fluid phase distinction by sign of level set function:

\[
\phi(x, t) > 0 \quad \text{for } x \in \Omega_1
\]

\[
\phi(x, t) \leq 0 \quad \text{for } x \in \Omega_2
\]
Solver algorithm based on pressure projection

For $t = 1, 2, \ldots$ do:

1. set boundary conditions for $u^n$
2. compute intermediate velocity field $u^*$:

$$\frac{u^* - u^n}{\delta t} = -(u^n \cdot \nabla)u^n + g + \frac{1}{\rho(\phi^n)} \nabla \cdot (\mu(\phi^n)S^n) - \frac{1}{\rho(\phi^n)} \sigma_k(\phi^n) \delta(\phi^n) \nabla \phi^n$$

3. apply boundary conditions and transport level-set function:

$$\phi^* = \phi^n + \delta t (u^n \cdot \nabla \phi^n)$$

4. reinitialize level-set function by solving

$$\partial_\tau d + \text{sign}(\phi^*)(|\nabla d| - 1) = 0, \quad d^0 = \phi^*$$

5. solve the pressure Poisson equation with $\phi^{n+1} = d$:

$$\nabla \cdot \left( \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1} \right) = \nabla \cdot u^*$$

6. apply velocity correction:

$$u^{n+1} = u^* - \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1}$$

This is now done on multiple GPUs.
Solver algorithm based on pressure projection

For \( t = 1, 2, \ldots \) do:

1. set boundary conditions for \( u^n \)
2. compute intermediate velocity field \( u^* \):
   \[
   \frac{u^* - u^n}{\delta t} = -(u^n \cdot \nabla)u^n + \rho(u^n) \nabla \phi^n \nabla \cdot \rho(u^n)(\nabla(S_n) - \rho(u^n) \Phi_n) - 1 \rho(u^n) \Phi_n S_n
   \]
3. apply boundary conditions
4. reinitialize level-set function by solving
   \[
   \partial_t \delta + \text{sign}(\phi^*)(|\nabla \delta| - 1) = 0, \quad \delta_0 = \phi^*
   \]
5. solve the pressure Poisson equation with \( \phi^{n+1} = d \):
   \[
   \nabla \cdot \left( \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1} \right) = \nabla \cdot u^*
   \]
6. apply velocity correction: \( u^{n+1} = u^* - \frac{\delta t}{\rho(\phi^{n+1})} \nabla p^{n+1} \)

This is now done on multiple GPUs.
Benchmarking problem: air bubble rising in water

<table>
<thead>
<tr>
<th>Properties</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>domain size:</td>
<td>20 cm $\times$ 20 cm $\times$ 20 cm</td>
</tr>
<tr>
<td>liquid phase:</td>
<td>water at 20$^\circ$C</td>
</tr>
<tr>
<td>gas phase:</td>
<td>air at 20$^\circ$C</td>
</tr>
<tr>
<td>surface tension:</td>
<td>standard</td>
</tr>
<tr>
<td>volume forces:</td>
<td>standard gravity</td>
</tr>
<tr>
<td>initial air bubble radius:</td>
<td>3 cm</td>
</tr>
<tr>
<td>initial center position of bubble:</td>
<td>(10 cm, 6 cm, 10 cm)</td>
</tr>
</tbody>
</table>
Finite noise assumption

Construction

Infinite dimensional stochastic variable $\omega \Rightarrow$ finite representation (Karhunen-Loève )

$$\{ Y_n(\omega) \}_{n=1}^{N_{FN}}, \quad Y_n : \Omega \rightarrow \mathbb{R}$$

Finite noise approximation

$$a(\omega, x, t) \approx a(Y_1(\omega), \ldots, Y_{N_{FN}}(\omega), x, t)$$
$$f(\omega, x, t) \approx f(Y_1(\omega), \ldots, Y_{N_{FN}}(\omega), x, t) \quad \text{on } \Omega \times D.$$ 

Joint probability density $\rho : \Gamma \rightarrow \mathbb{R}^+$, $\Gamma := \prod_{n=1}^{N_{FN}} \Gamma_n$, $\Gamma_n \equiv Y_n(\Omega)$, $\rho \in L^\infty(\Gamma)$

New deterministic problem formulation

Find function $u : \Gamma \times \tilde{D} \times [0, T] \rightarrow \mathcal{S}$, such that

$$\mathcal{L}(a(y, x, t))u(y, x, t) = f(y, x, t) \quad \text{in } \Gamma \times \tilde{D} \times [0, T]$$
Reproducing kernel Hilbert spaces

**Definition (Reproducing kernels)**

Let $\mathcal{F}$ be a Hilbert space of functions $f : \Omega \to \mathbb{R}$. A function $k : \Omega \times \Omega \to \mathbb{R}$ is called reproducing kernel for $\mathcal{F}$ if

1. $k(\cdot, y) \in \mathcal{F}$ for all $y \in \Omega$,
2. $f(y) = (f, k(\cdot, y))_{\mathcal{F}}$ for all $f \in \mathcal{F}$ and all $y \in \Omega$.

**Definition (Positive (semi-)definite kernels)**

$$
\sum_{j=1}^{N} \sum_{k=1}^{N} \alpha_j \alpha_k k(y_j, y_k) \geq 0 \quad (> 0 \Rightarrow \text{positive definite}).
$$

**Theorem**

Suppose that $\mathcal{F}$ is a reproducing kernel Hilbert function space with reproducing kernel $k : \Omega \times \Omega \to \mathbb{R}$. Then $k$ is positive semi-definite. Moreover, $k$ is positive definite if and only if the point evaluation functionals are linearly independent in $\mathcal{F}^*$. 
Radial kernels

Definition

Let $\mathcal{T}$ be a group of transformations $T : \Omega \to \Omega$. $\mathcal{F}$ is invariant under the group $\mathcal{T}$ if

1. $f \circ T \in \mathcal{F}$ for all $f \in \mathcal{F}$ and $T \in \mathcal{T}$,

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

Theorem

Suppose that the RKHS is invariant under the transformations of $\mathcal{T}$, then the reproducing kernel $k$ satisfies for all $x, y \in \Omega$ and all $T \in \mathcal{T}$:

$$k(Tx, Ty) = k(x, y).$$

Construction of radial kernels

$\Omega = \mathbb{R}^d$, $\mathcal{T}$ set of translations and orthogonal transformations

$A \in \mathbb{R}^{d \times d}$ are orthogonal transformations s.th. $A\xi = ||\xi||e_1$, $\xi = x - y$

$$k(x, y) = k(Ax, Ay) = k(0, x - y) =: k_0(x - y) = k_0(||x - y||e_1) =: k_1(||x - y||).$$
Native spaces

Theorem

If \( k : \Omega \times \Omega \to \mathbb{R} \) is a symmetric positive definite kernel then \((\cdot, \cdot)_k\) with
\[
\left( \sum_{j=1}^{N} \alpha_j k(\cdot, x_j), \sum_{k=1}^{M} \beta_k k(\cdot, y_k) \right)_k := \sum_{j=1}^{N} \sum_{k=1}^{M} \alpha_j \beta_k k(x_j, y_k)
\]
defines an inner product on the pre-Hilbert space \( F_k(\Omega) \) with reproducing kernel \( k \), given by
\[
F_k(\Omega) := \text{span} \{ k(\cdot, y) : y \in \Omega \}.
\]

Definition (Native Hilbert function space)

The native Hilbert function space corresponding to the symmetric positive definite kernel \( k : \Omega \times \Omega \to \mathbb{R} \) is defined by
\[
\mathcal{N}_k(\Omega) := R(\mathcal{F}_k(\Omega)), \quad R : \mathcal{F}_k(\Omega) \to C(\Omega), \quad R(f)(x) := (f, k(\cdot, x))_k.
\]

Theorem

Suppose that \( k : \Omega \times \Omega \to \mathbb{R} \) is a symmetric positive definite kernel. Then its associated native space \( \mathcal{N}_k(\Omega) \) is a Hilbert function space with reproducing kernel \( k \).
Error estimates in native spaces

Requirement

\[ f \in \mathcal{N}_{k_\epsilon}(\Omega), \quad \Omega \text{ cube in } \mathbb{R}^s \]

Definition (Fill distance)

\[ h_{X,\Omega} := \sup_{y \in \Omega} \min_{y_j \in X} ||y - y_i||_2 \]

Theorem (Gaussian kernel \( k_\epsilon(y_i, y_j) = e^{-\epsilon^2 ||y_i - y_j||^2} \))

\[ ||f - s_{X,f}||_{L_\infty(\Omega)} \leq e^{\frac{-c \log h_{X,\Omega}}{h_{X,\Omega}}} ||f||_{\mathcal{N}_{k_\epsilon}(\Omega)} \]

Theorem (Wendland kernels)

\[ |f(x) - s_{X,f}(x)| \leq Ch_{X,\Omega}^{k + \frac{1}{2}} ||f||_{\mathcal{N}_k(\Omega)} \quad \forall x \in \Omega \]

s_{X,f} function \( f \) interpolated by Lagrange interpolation with collocation points \( X \)
Discretization by the Nyström method (1)

\[ \int_{D} \text{Cov}[u](x, x') \psi_k(x')dx' = \lambda_k \psi_k(x). \]

**Basic idea:** Integral discretization by quadrature rule

\[ Q_{NY}(f) = \sum_{j=1}^{N_{NY}} w_j f(q_j) \quad q_j \in D \text{ q. points, } w_j \in \mathbb{R} \text{ q. weights} \]

**Semi-discrete formulation of integral equation**

\[ \sum_{j=1}^{N_{NY}} w_j \text{Cov}[u](x, q_j) \psi_k(q_j) = \lambda_k \psi_k(x) \quad \forall x \in D \]

**Fully discrete problem** (by also restricting \( x \in D \) to \( q_j \))

\[ \sum_{j=1}^{N_{NY}} w_j \text{Cov}[u](q_i, q_j) \psi_k(q_j) = \lambda_k \psi_k(q_i) \quad i = 1, \ldots, N_{NY} \]
Shorthand notation of discrete problem

\[
\psi^\text{NY}_k := \begin{pmatrix} \psi_{1,k} \\ \vdots \\ \psi_{\text{NY},k} \end{pmatrix} \quad \text{and} \quad B := \begin{pmatrix} \beta_{1,1} & \cdots & \beta_{1,\text{NY}} \\ \vdots & \ddots & \vdots \\ \beta_{\text{NY},1} & \cdots & \beta_{\text{NY},\text{NY}} \end{pmatrix},
\]

\[
\psi_{i,k} := \psi_k(q_i), \quad \beta_{i,j} := w_j \text{Cov}[u](q_i, q_j)
\]

\[(\lambda_k I - B) \psi^\text{NY}_k = 0.\]

Evaluation of continuous eigenfunctions \( \psi_k(x) \) (Nyström interpolation)

\[
\psi_k(x) = \frac{1}{\lambda_k} \left( \sum_{j=1}^{\text{NY}} w_j \text{Cov}[u](x, q_j) \psi_{j,k} \right)
\]
Discretization by the Nyström method: Specializing quadrature

Choice of quadrature

Monte-Carlo or Quasi Monte-Carlo \( \Rightarrow \) quadrature weights \( w_j = \frac{1}{N_{Ny}} \)

Final problem

\[
\left( \tilde{\lambda}_k I - C \right) \psi_{k}^{N_{NY}} = 0
\]

- \( C \in \mathbb{R}^{N_{NY} \times N_{NY}} \) discrete covariance matrix, \( \tilde{\lambda}_k = N_{NY} \lambda_k \)
- entries \( c_{i,j} = \text{Cov}[u](q_i, q_j) \) evaluated at the quadrature points

Summary of solution method for Fredholm integral equation EV problem

- computing EVs of discrete covariance matrix with point sampling at quad. points
- eigenvalues \( \lambda_k \) given by \( \lambda_k = \frac{1}{N_{NY}} \tilde{\lambda}_k \)
Convergence analysis

**Convergence of Nyström method eigenvalue problems**

Under some assumptions:

\[ |\lambda - \lambda_n| \leq Ch^p, \quad ||f_n|| \leq Ch^p \]

**Convergence of Lanczos method**

very fast (exponential?) convergence of extremal eigenvalues

**Remark**

spectrum of covariance matrix usually expected to have exponential decay

⇒ very fast convergence
Convergence results for MC quadrature and QMC collo. pts.

\[ \varphi_\epsilon(r) = e^{-(\epsilon r)^2} \]

\[ \varphi_{3,0}(r) = ((1-r)^2)_+ \]

\[ \varphi_{3,1}(r) = ((1-r)^4)_+ (4r+1) \]

\[ \varphi_{3,2}(r) = ((1-r)^6)_+ (35r^2 + 18r + 3) \]
Convergence results for QMC quadrature and sparse grid collocation points.

\[ \varphi_\epsilon(r) = e^{-(\epsilon r)^2} \]
Lanczos at work (1)

Result after $\ell$ iterations

\[
C_\ell = \begin{bmatrix}
\alpha_1 & \beta_1 \\
\beta_1 & \alpha_2 & \beta_2 \\
& \ddots & \ddots & \ddots \\
& & \beta_{\ell-2} & \alpha_{\ell-1} & \beta_{\ell-1} \\
& & & \beta_{\ell-1} & \alpha_\ell
\end{bmatrix},
\]

\[
K_\ell = \begin{bmatrix}
k_1 & k_2 & \ldots & k_\ell
\end{bmatrix}, \text{ } k_i \text{ orthonormal}
\]

\[
C_\ell = K_\ell^T C K_\ell
\]

Lanczos algorithm

Require: $C$ s.p.d.

1: \textbf{function LANCZOS}(C, $\ell$)

2: \begin{align*}
\beta_0 &= 0, \quad k_0 = 0, \quad b = \text{arbitrary}, \\
k_1 &= \frac{b}{||b||}
\end{align*}

3: \textbf{for } n = 1, 2, \ldots, \ell \textbf{ do}

4: \quad v = C k_n

5: \quad \alpha_n = k_n^T v

6: \quad v = v - \beta_{n-1} k_{n-1} - \alpha_n k_n

7: \quad \beta_n = ||v||

8: \quad k_{n+1} = v / \beta_n

9: \textbf{return } \{k_n\}_n, \{\alpha_n\}_n, \{\beta_n\}_n
Stochastic quantities of interest

**Expectation value**

\[
\mathbb{E}[u](x, t) = \int_{\Omega_M} u(x, t, y) \rho_M(y) \, dy
\]

**Variance**

\[
\text{Var}[u](x, t, y) = \mathbb{E}[(u(x, t, \cdot) - \mathbb{E}[u](x, t))^2]
\]

**Covariance**

\[
\text{Cov}[u](x, x', t) = \mathbb{E}[u(x, t, \cdot)u(x', t, \cdot)] - \mathbb{E}[u](x, t) \cdot \mathbb{E}[u](x', t)
\]
Test case: Flow over a backward facing step

- domain size: $20m \times 4m \times 2m$, grid resolution: 50x40x3
- storage for 512 realizations over 100 time steps: 30GB
- stochastic parameters:
  
  $u_{inflow} \sim \mathcal{U}(0.1, 1.0), \quad \mu \sim \mathcal{U}(0.001, 1.0) \quad \text{and} \quad \rho \sim \mathcal{U}(500, 1000)$.

- weighted norm in kernels:
  
  $$\| (u_{inflow}, \mu, \rho)^T \| := \sqrt{u_{inflow}^2 + \mu^2 + 10^{-6} \rho^2}$$
Convergence of expectation value of full velocity field at \( t = 1.0s \).
trying out more combinations of kernels, collocation points and quadrature
adaptive choice of collocation points?
optimal-convergence (multi-grid?) solver for interpolation problem?
multi-GPU implementation of basic UQ method (more quadrature points)?
automatic job creation framework for combination technique
numerical tests with combination technique
scalable multi-GPU implementation of Lanczos method
nice application for Karhunen-Loève → model reduction / optimal kernels
error analysis?