Adaptive Sparse Grids for UQ with SG++

SIAM UQ, MS “Software for UQ”

Dirk Pflüger, Fabian Franzelin

Simulation of Large Systems, IPVS/SimTech, University of Stuttgart

April 6, 2016
Demonstrator: CO$_2$ benchmark problem [Class et al., 2009]

- CO$_2$ injected into deep aquifer,
- spreads within aquifer,
- reaches a leaky well and
- rises up to a shallower aquifer
Demonstrator: CO$_2$ benchmark problem [Class et al., 2009]

- CO$_2$ injected into deep aquifer,
- spreads within aquifer,
- reaches a leaky well and
- rises up to a shallower aquifer

- properties of subsurface unknown
- assumption: similar to others,
- ... from which we have data
- solver as black box
Uncertainty Quantification Pipeline

\[ \mathcal{D} = \{ \xi^{(k)} \}_{k=1}^{n} \rightarrow \hat{f}(\xi) \approx f(\xi) \rightarrow \tilde{\xi} = (\xi_1, \ldots, \xi_d) \in \Omega \]

\[ u(\vec{x}, t; \tilde{\xi}) \rightarrow \mathbb{E}_f(u), \nabla f(u), \tilde{f}(u) \]

Data Density Estimation Uncertain Parameters

Physical Model Forward Propagation Analysis
Uncertainty Quantification Pipeline

\[ \mathcal{D} = \{ \xi^{(k)} \}_{k=1}^{n} \]

\[ \hat{f}(\xi) \approx f(\xi) \]

\[ \tilde{\xi} = (\xi_1, \ldots, \xi_d) \in \Omega \]

\[ u(\bar{x}, t; \tilde{\xi}) \]

\[ \mathbb{E}_f(u), \, \nabla f(u), \, \tilde{f}(u) \]

Data  \rightarrow  Density Estimation  \rightarrow  Uncertain Parameters

Physical Model  \rightarrow  Forward Propagation  \rightarrow  Analysis
Overview

1. Sparse Grids
2. SG++
3. Walk through the UQ pipeline
Sparse Grids in a Nutshell

- Mitigate curse of dimensionality! $O(2^l d) \Rightarrow O(2^l l^{d-1})$
- Main ideas:
  - hierarchical, incremental, piecewise polynomial spaces

\[ \begin{align*}
\phi_{1,1} \\
\phi_{2,1} \quad \phi_{2,3} \\
\phi_{3,1} \quad \phi_{3,3} \quad \phi_{3,5} \quad \phi_{3,7}
\end{align*} \]

\[ \begin{align*}
l = 1 \\
l = 2 \\
l = 3
\end{align*} \]
Sparse Grids in a Nutshell

- Mitigate curse of dimensionality! $O((2^l)^d) \Rightarrow O(2^l l^{d-1})$
- Main ideas:
  - hierarchical, incremental, piecewise polynomial spaces
  - $d$-dim: optimal truncation of tensor product spaces ($H_{mix}^2$)

\[ \phi_{1,1}, \phi_{2,1}, \phi_{2,3}, \phi_{3,1}, \phi_{3,3}, \phi_{3,5}, \phi_{3,7} \]

\[ x_{1,1}, x_{2,1}, x_{2,3}, x_{3,1}, x_{3,3}, x_{3,5}, x_{3,7} \]

$l=1$

$l=2$

$l=3$

$l_1=1$

$l_1=3$

$l_2=1$

$l_2=2$

$l_2=3$

$l_1$

$l_2$
Sparse Grids in a Nutshell

- Mitigate curse of dimensionality! $O((2^l)^d) \Rightarrow O(2^d l^{d-1})$
- Main ideas:
  - hierarchical, incremental, piecewise polynomial spaces
  - $d$-dim: optimal truncation of tensor product spaces ($\mathcal{H}_{mix}^2$)
  - spatially adaptive: adapt to local features
- note: not combination technique!

\[
\begin{align*}
l_1 &= 1 \\
l_1 &= 2 \\
l_1 &= 3 \\
l_2 &= 1 \\
l_2 &= 2 \\
l_2 &= 3 \\
l_2 &= 4 \\
l_2 &= 5
\end{align*}
\]
General framework for (spatially adaptive) sparse grids
- Open source
- Active development
- Extensible (write/contribute your own module)
Scope: High-Dimensional Problems

Financial mathematics
Plasma physics
Automotive engineering
Data Mining
Hydraulic engineering
Uncertainty Quantification
Computational Steering
Mechanics
Model reduction
Clustering
PDEs
Interpolation
High dimensionalities
Regression
Dimensionality reduction
Optimization
Manifold learning
Sensitivity analysis
Quadrature
Density estimation
Regression
Density estimation
SG++

- Written in C++11
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
  - multi-core
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
  - multi-core
  - hybrid nodes (even multi-vendor support)
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
  - multi-core
  - hybrid nodes (even multi-vendor support)
  - up to >30000 cores
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
  - multi-core
  - hybrid nodes (even multi-vendor support)
  - up to >30000 cores
- Performance portability via auto-tuning and OpenCL + JIT
SG++

- Written in C++11
- Usability: interfaces to Python, Matlab, Java
- Multi-platform support: Linux/Unix, Windows, OSX
- Efficient implementations
  - multi-core
  - hybrid nodes (even multi-vendor support)
  - up to >30000 cores
- Performance portability via auto-tuning and OpenCL + JIT
- Factory pattern selects best available implementation for given task
SG++

- Designed for flexibility and modularity
SG++

- Designed for flexibility and modularity
  - current modules:
    - base, pde, solver, datadriven, quadrature, optimization

Vast range of functionality
large range of basis functions
SG++

- Designed for flexibility and modularity
  - current modules: base, pde, solver, datadriven, quadrature, optimization
- Vast range of functionality
SG++

- Designed for flexibility and modularity
  - current modules: base, pde, solver, datadriven, quadrature, optimization
- Vast range of functionality
  - large range of basis functions

Plenty of refinement criteria: interpolation, quadrature, optimization, data-driven tasks (UQ, regression, classification, density estimation, ...)
**SG++**

- Designed for flexibility and modularity
  - current modules:
    - base, pde, solver, datadriven, quadrature, optimization
- Vast range of functionality
  - large range of basis functions
  
- plenty of refinement criteria
SG++

- Designed for flexibility and modularity
  - current modules: base, pde, solver, datadriven, quadrature, optimization

- Vast range of functionality
  - large range of basis functions

- plenty of refinement criteria
  - interpolation, quadrature, optimization, data-driven tasks (UQ, regression, classification, density estimation, . . .)
Density Estimation

Why data-driven

- Start from ground truth
- Expert knowledge can be subjective
  [Oladyshkin 2012; Franzelin, P 2016]
- Use analytical representation to adapt in forward propagation
- No further numerical approximation necessary
Density Estimation

Why data-driven
- Start from ground truth
- Expert knowledge can be subjective
  [Oladyshkin 2012; Franzelin, P 2016]
- Use analytical representation to adapt in forward propagation
- No further numerical approximation necessary

How
- Start from data
  \[ S^k = \{ \vec{x}_1, \ldots, \vec{x}_m \} \subset \mathbb{R}^d \]
- Penalized L2 minimization
  \[ f = \arg\min_{u \in V} \int_{\Omega} (u(\vec{x}) - f_{S^k}(\vec{x}))^2 d\vec{x} + \lambda \|Su\|_{L^2}^2 \]
  with initial guess \( f_{S^k} := \frac{1}{m} \sum_{i=1}^{m} \delta_{\vec{x}_i} \)
- Ritz-Galerkin
- Solve iteratively in \( O(kN + \log(N)m) \); offline-online splitting possible
Density Estimation

Results

![Graphs showing results of density estimation.](image)
Density Estimation

Results

- Comparison of different DE methods
- Cross entropy (bootstrapping): SGDE best

<table>
<thead>
<tr>
<th># training</th>
<th># test</th>
<th>analytic (L)</th>
<th>libagf (L)</th>
<th>dtrees (L)</th>
<th>SGDE (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td>363</td>
<td>50</td>
<td>0.7279</td>
<td>0.00278</td>
<td>-0.1314</td>
<td>-0.3042</td>
</tr>
</tbody>
</table>

**Table:** Mean cross entropy (L) for different density estimation methods, 10 runs
Density Estimation: SG++

Example in Matlab

```matlab
% load library
sgpp.LoadJSGPPLib.loadJSGPPLib();

% import packages
import sgpp.RegularGridConfiguration
[...] import sgpp.DataVector

% load data set
data = csvread('exp_2d.csv');
[...]
```
Density Estimation: SG++

Example in Matlab

```matlab
% configure
gridConfig = RegularGridConfiguration();
gridConfig.setDim_(numDims);
gridConfig.setLevel_(3);
gridConfig.setType_(GridType.LinearBoundary);

adaptConfig = AdaptivityConfiguration();
adaptConfig.setNoPoints_(3);
adaptConfig.setNumRefinements_(5);

solverConfig = SLESolverConfiguration();
solverConfig.setType_(SLESolverType.CG);
solverConfig.setMaxIterations_(100);
solverConfig.setEps_(1e−10);
solverConfig.setThreshold_(1e−10);
```
Density Estimation: SG++

Example in Matlab

```matlab
regularizationConfig = RegularizationConfiguration();
regularizationConfig.setRegType_(RegularizationType.Laplace);

cvConfig =
    CrossvalidationForRegularizationConfiguration();
cvConfig.setEnable_(true);
cvConfig.setKfold_(5);
cvConfig.setLambdaStart_(1e-1);
cvConfig.setLambdaEnd_(1e-10);
cvConfig.setLambdaSteps_(5);
cvConfig.setLogScale_(true);
cvConfig.setShuffle_(true);
cvConfig.setSeed_(1234567);
cvConfig.setSilent_(false);
```
Density Estimation: SG++

Example in Matlab

```matlab
% learn
learner = LearnerSGDE(gridConfig, adaptConfig, solverConfig, regularizationConfig, cvConfig);
learner.initialize(samples);

% plot / use / ...
```
Density Estimation: SG++

Fast track with Python

```python
# load module
[...]  
# load data set
[...]  

# estimate sparse grid density
sgde.byConfig(samples,
              config={
                  "grid_level": 6,
                  "grid_type": "Linear",
                  "refinement_numSteps": 5,
                  "regularization_type": "Laplace",
                  "crossValidation_enable": True,
                  "crossValidation_kfold": 5},
              bounds=[[0, 1], [0, 1]])
```
Transformation: Rosenblatt

Why

- Correlated densities
- Decorrelate to fit tensor-product structure of sparse grid space
- Generate additional samples (fast inverse)
Transformation: Rosenblatt

Why
- Correlated densities
- Decorrelate to fit tensor-product structure of sparse grid space
- Generate additional samples (fast inverse)

How
- Compute \( R : (\xi_1, \ldots, \xi_d) \in \Omega \mapsto (\zeta_1, \ldots, \zeta_d) \in \mathcal{U} \)

\[
\begin{align*}
\zeta_1 &= F_1(\xi_1) \\
\vdots \\
\zeta_d &= F_{d|d-1,\ldots,1}(\xi_{d|d-1,\ldots,\xi_1})
\end{align*}
\]

- Sparse grid representation provides: conditionalization by marginalization, 1D quadrature w/o extra numerical approximation

See talk by Franzelin in MS 111, Thu, 4:40pm
**Transformation: Rosenblatt: SG++**

**Python**

```python
builder = ParameterBuilder()
up = builder.defineUncertainParameters()
up.new().isCalled('x_0').withUniformDistribution(0, 1)
up.new().isCalled('x_1', 'x_2').withSGDEDistribution(sgde).withRosenblattTransformation()

params = builder.andGetResult()
```
**Forward Propagation**

**Why sparse grid collocation**
- Solver as black box
- Direct sparse grid space: local support of basis functions + spatial adaptive refinement
  ⇒ Gibbs and Runge
- Robust and flexible
Forward Propagation

Why sparse grid collocation

- Solver as black box
- Direct sparse grid space: local support of basis functions + spatial adaptive refinement
  ⇒ Gibbs and Runge
- Robust and flexible

Example:

true model

sparse grid function

PCE model
Forward Propagation

How
- Suitable basis functions
- Adaptive refinement
  - Standard criterion for free
  - Problem-adapted criteria

Example:
Forward Propagation

How

- Suitable basis functions
- Adaptive refinement
  - Standard criterion for free
  - Problem-adapted criteria

Example:

\[
||u(x) - y|| \leq 10^{-1} \quad 10^{-2} \quad 10^{-3} \quad 10^{-4} \quad 10^{-5} \quad 10^{-6} \quad 10^{-7}
\]

\[
\text{collocation nodes}
\]

Different methods and their performance:
- MC (average)
- QMC
- (l) full grid
- (l) regular sg
- (p) regular sg
- (p) expectation
- (p) variance

Slope analysis:
- \(N^{-1/2}\)
- \(N^{+1/2} \log(N)^{d}\)
Forward Propagation: SG++

```python
# input modeling
builder = ParameterBuilderInterface()
up = builder.defineUncertainParameters()
up.new().isCalled('x_0').withUniformDistribution(0, 1)
up.new().isCalled('x_1, x_2').withSGDEDistribution(sgde)
    .withRosenblattTransformation()
params = builder.andGetResult()

# define UQ setting
builder = ASGCUQManagerBuilder()
builder.withParameters(params)
    .withTypesOfKnowledge([KnowledgeTypes.SIMPLE,
                            KnowledgeTypes.SQUARED])
    .useInterpolation()
builder.defineUQSetting().withSimulation(g)
```
Forward Propagation: SG++

`samplerSpec = builder.defineSampler()

samplerSpec.withGrid().withLevel(4)
    .withPolynomialBase(2)
    .withBorder(BorderTypes.TRAPEZOIDBOUNDARY)

samplerSpec.withRefinement()
    .withAdaptThreshold(1e−10)
    .withAdaptPoints(5)
    .withBalancing()
    .refineMostPromisingNodes()
        .withSquaredSurplusRanking()
        .createAllChildrenOnRefinement()
        .refineInnerNodes()
    .withStopPolicy().withAdaptiveIterationLimit(0)

# do the work
uqManager = builder.andGetResult()`
Analysis

**How**: Let \( \hat{f} \approx f \) be SGDE, \( g_I \approx u_M \) SG surrogate, then computation of moments without numerical quadrature
Analysis

How: Let \( \hat{f} \approx f \) be SGDE, \( g_I \approx u_M \) SG surrogate, then computation of moments without numerical quadrature

\[
\mathbb{E}_{\hat{f}_K}(g_I) = \int_{\Omega} g_I(\vec{\theta}) \hat{f}_K(\vec{\theta}) d\vec{\theta}
\]

\[
= \int_{\Omega} \sum_{(\vec{l}, \vec{i}) \in \mathcal{I}} v_{\vec{l}, \vec{i}} \psi_{\vec{l}, \vec{i}}^{(p)}(\vec{\theta}) \sum_{(\vec{k}, \vec{j}) \in \mathcal{K}} w_{\vec{k}, \vec{j}} \varphi_{\vec{k}, \vec{j}}^{(q)}(\vec{\theta}) d\vec{\theta}
\]

\[
= \sum_{(\vec{l}, \vec{i}) \in \mathcal{I}} v_{\vec{l}, \vec{i}} \sum_{(\vec{k}, \vec{j}) \in \mathcal{K}} w_{\vec{k}, \vec{j}} \int_{\Omega} \psi_{\vec{l}, \vec{i}}^{(p)}(\vec{\theta}) \varphi_{\vec{k}, \vec{j}}^{(q)}(\vec{\theta}) d\vec{\theta}
\]

\[
= \sum_{(\vec{l}, \vec{i}) \in \mathcal{I}} v_{\vec{l}, \vec{i}} \sum_{(\vec{k}, \vec{j}) \in \mathcal{K}} w_{\vec{k}, \vec{j}} \int_{\Omega_1} \psi_{\vec{l}, \vec{i}}^{(p)}(\vec{\theta}) \varphi_{\vec{k}, \vec{j}}^{(q)}(\vec{\theta}) d\vec{\theta}_1 \ldots \int_{\Omega_d} \psi_{\vec{l}, \vec{i}}^{(p)}(\vec{\theta}) \varphi_{\vec{k}, \vec{j}}^{(q)}(\vec{\theta}) d\vec{\theta}_d
\]

\[
= \vec{v}^T A \vec{w}
\]

- direct: \( O(NM) \), using unidir. scheme: \( O(N + M) \)
- Variance, marginals correspondingly
Analysis

Python

# build analysis
analysis = ASGCAnalysisBuilder()
    .withUQManager(uqManager)
    .withAnalyticEstimationStrategy()
    .andGetResult()
analysis.computeMoments()['data']

# anova decomposition
anova = analysis.getAnovaDecomposition(nk=len(params))

# main effects
me = anova.getSobolIndices()
te = anova.getTotalEffects()
[...]
Whole UQ pipeline

Results:

- Close subjective gap
- Expectation: SG second; variance: SG best
Summary

CO2 benchmark problem

- Starting from data: SGDE
- Transformations
- Forward propagation using adaptive sparse grids
- Analysis
Summary

CO2 benchmark problem
- Starting from data: SGDE
- Transformations
- Forward propagation using adaptive sparse grids
- Analysis

SG++
- Demonstration of SG++ for UQ
- Flexible toolkit
- Largest set of functionality for spatially adaptive sparse grids
- Easy to extend
Thanks to:

...and all others!

Thank you for your interest!

