UQTk

A C++/Python Toolkit for Uncertainty Quantification

Bert Debusschere, Khachik Sargsyan, Cosmin Safta, Kenny Chowdhary

bjdebus@sandia.gov
Sandia National Laboratories,
Livermore, CA, USA

Wed April 6, 2016 – SIAM UQ 16

QUANTIFICATION OF UNCERTAINTY IN EXTREME SCALE COMPUTATIONS
QUEST

Acknowledgements

Cosmin Safta Khachik Sargsyan Kenny Chowdhary Habib Najm	Sandia National Laboratories Livermore, CA, USA
Omar Knio	Duke University, Raleigh, NC, USA
Roger Ghanem	University of Southern California Los Angeles, CA, USA
Olivier Le Maître	LIMSI-CNRS, Orsay, France
and many others ...	

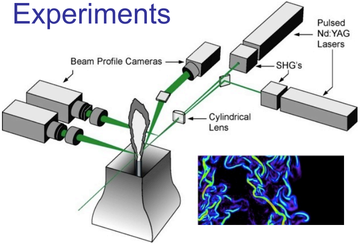
This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Scientific Discovery through Advanced Computing (SciDAC) and Applied Mathematics Research (AMR) programs.

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

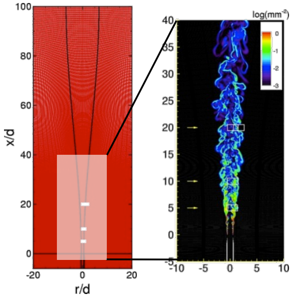
Outline

UQ is about enabling predictive simulations

Experiments



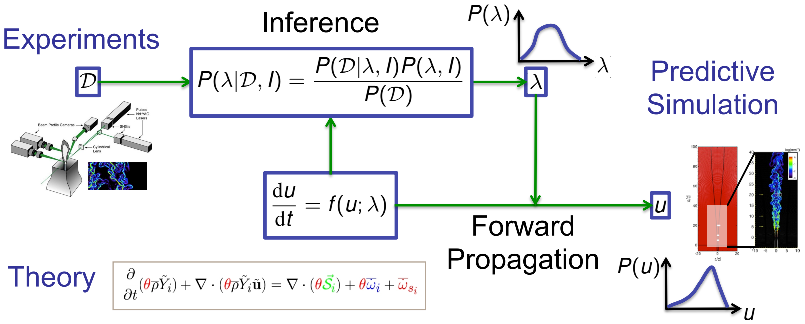
Predictive Simulation



$$\frac{\partial}{\partial t}(\theta\bar{\rho}\tilde{Y}_i) + \nabla\cdot(\theta\bar{\rho}\tilde{Y}_i\tilde{\mathbf{u}}) = \nabla\cdot(\theta\vec{S}_i) + \theta\bar{\dot{\omega}}_i + \bar{\dot{\omega}}_{s_i}$$

Theory

UQ methods extract information from all sources to enable predictive simulation



- UQ not just about propagating uncertainties
- The term UQ covers a wide range of methods

UQTk provides tools to build a general UQ workflow

- Tools for
  - Representation of random variables and stochastic processes
  - Forward uncertainty propagation
  - Inverse problems
  - Sensitivity analysis
  - Bayesian Compressive Sensing
  - Gaussian Processes
- Tools can be used stand-alone or combined into a general workflow

We want UQTk to be straightforward to download, install and use

- Target usage:
  - Rapid prototyping of UQ workflows
  - Algorithmic research in UQ
  - Tutorials / educational
- Released under the GNU Lesser General Public License
  - `http://www.sandia.gov/UQToolkit/`
  - Current version 2.1
  - Version 3.0 coming (very) soon
- No massive third party libaries to download, install, and configure

UQTk is used in a variety of applications

- Direct collaborations
  - US DOE SciDAC QUEST UQ institute `http://www.quest-scidac.org`
  - Variety of US DOE SciDAC partnership projects
  - Part of US DOE BER ACME climate model analysis tools
  - Always welcome new applications / collaborations
- Downloads from `http://www.sandia.gov/UQToolkit`
  - $\approx$ 600 total downloads
  - $\approx$ 425 downloads of version 2.x
  - Mostly academic and laboratory research groups
- Mailing lists
  - `uqtk-announce@software.sandia.gov`
  - `uqtk-users@software.sandia.gov`
  - Join at `http://www.sandia.gov/UQToolkit`

We rely on Polynomial Chaos expansions (PCEs) to represent uncertainty

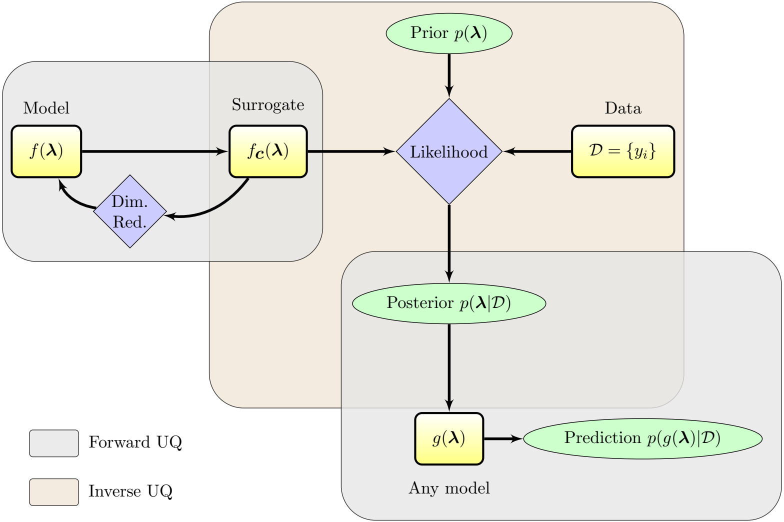
- Standard PC Basis types supported:
  - Gauss – Hermite
  - Uniform – Legendre
  - Gamma – Laguerre
  - Beta – Jacobi
- Also support for custom orthogonal polynomials
  - Defined by user-provided three-term recurrence formula
- Both intrusive and non-intrusive PC tools provided
  - Primarily Galerkin projection methods
  - Some regression approaches offered through Bayesian Compressed Sensing module
  - See also Debusschere, *et al.* 2004; Sargsyan, *et al.* 2014

UQTk uses a combination of C++ and Python

- Main libraries in C++
  - `PCBasis` and `PCSet` classes: PC tools (intrusive and non-intrusive)
  - `Quad` class: quadrature rules (full tensor and sparse tensor product rules)
  - `MCMC, Gproc,...`
- Functionality available via
  - Direct linking of C++ code
  - Standalone apps
  - Python interface based on Swig (UQTk version 3.0)
- Download as tar file and configure with `CMake`
- Examples of common workflows provided

Outline

An example UQTk workflow (UQTk version 3.0)



An example UQTk workflow (UQTk version 3.0)

- Consider uncertain parameter vector $\boldsymbol{\lambda}$
- Propagate uncertainty in $\boldsymbol{\lambda}$ through model $g(\boldsymbol{\lambda})$
- First calibrate $\boldsymbol{\lambda}$ using data on function $f(\boldsymbol{\lambda})$
- Consider the following calibration functions $f(\boldsymbol{\lambda})$:
  - Gaussian: $f^G(\boldsymbol{\lambda}) = \exp\left(-\sum_{i=1}^{d} a_i^2 \lambda_i^2\right)$
  - Exponential: $f^E(\boldsymbol{\lambda}) = \exp\left(\sum_{i=1}^{d} a_i \lambda_i\right)$
  - 5 dimensional $\boldsymbol{\lambda}$: $\boldsymbol{a} = (0.4, 0.3, 0.2, 0.1, 0.05)$
- Forward models $g(\boldsymbol{\lambda})$:
  - Gaussian: $g_1(\boldsymbol{\lambda}) = f^G(\boldsymbol{\lambda})$
  - Exponential: $g_2(\boldsymbol{\lambda}) = f^E(\boldsymbol{\lambda})$
  - Summation: $g_3(\boldsymbol{\lambda}) = \sum_{i=1}^{d} \lambda_i$
- For more details, see “Handbook of Uncertainty Quantification", Springer, 2016

Surrogate models provide computationally cheap approximations for full forward model

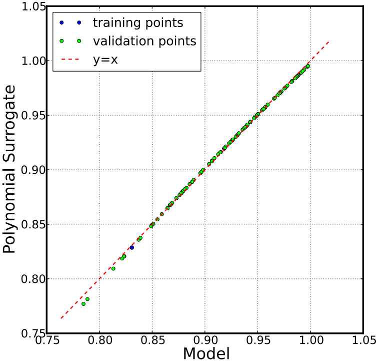
- Used instead of full forward model in computationally demanding operations such as optimization and calibration
- Use PCE surrogate model
- Same approach as forward UQ
  - Legendre-Uniform PCEs
  - Use uniform distributions over range of input parameters
  - Galerkin projection with Gauss quadrature
  - 3^{rd} order PCE using $4^5 = 1024$ quadrature points
  - 111 random validation samples to assess surrogate accuracy

Sample UQTk commands (using stand-alone apps)

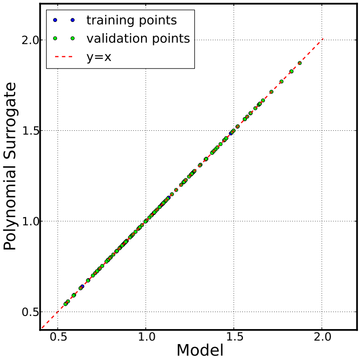
- Generate quadrature points:
  `generate_quad -d 5 -g LU -x full -p 4`
- Generate random samples for validation:
  `pce_rv -w PCvar -d 5 -n 111 -p 5 -x LU`
- Evaluate model at quadrature and validation points
- Perform Galerkin projection:
  `pce_resp -d 5 -x LEG -o 3 -e`
- Evaluate output PCE at validation points to compute error:
  `pce_eval -x PC -s LU -o 3 -f <INPC>`

Surrogate Construction

Gaussian



Exponential

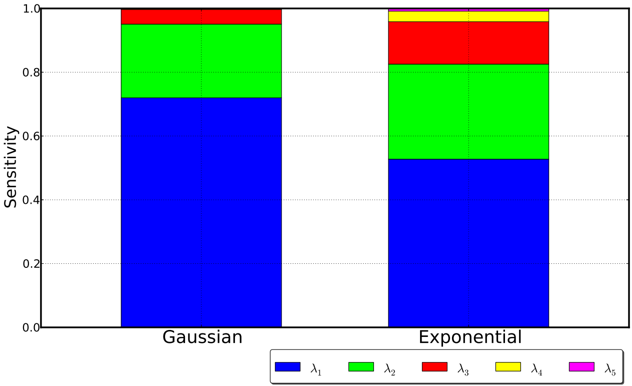


- 3^{rd} order PC surrogate accurate up to 0.1% relative error for both Gaussian and Exponential function

Sensitivity analysis enables dimensionality reduction

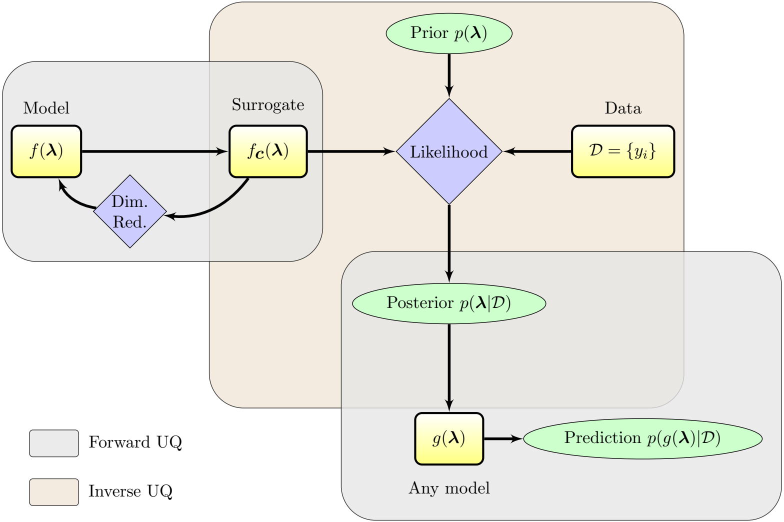
- UQTk computes main, joint, and total sensitivity coefficients
  - S_i: Fraction of variance due to λ_i only
  - S_{ij}: Fraction of variance due to both λ_i and λ_j
  - S_i^T: Fraction of variance due to λ_i by itself and in combination with any other λ_j
- Can be computed analytically from PC response surface
- `pce_sens -m mindex.dat \`
  `-f PCcoeff_quad.dat -x LU`

Components that explain most of the variance are retained



- More than 80% of variance attributed to first two components
- Include only λ_1 and λ_2 in calibration

Bayesian calibration using surrogate models



Bayesian Parameter Inference

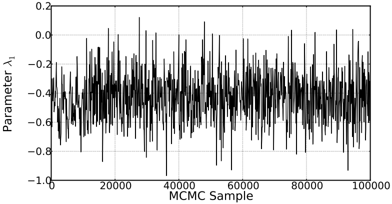
$$\overbrace{p(\boldsymbol{\lambda}|\mathcal{D})}^{\text{Posterior}} \propto \overbrace{p(\mathcal{D}|\boldsymbol{\lambda})}^{\text{Likelihood}}\,\overbrace{p(\boldsymbol{\lambda})}^{\text{Prior}}$$

$$\mathcal{L}_{\mathcal{D}}(\boldsymbol{\lambda}) = p(\mathcal{D}|\boldsymbol{\lambda}) \propto \prod_{j=1}^{R} \exp\left(-\frac{(y_j^G - f^G(\boldsymbol{\lambda}))^2}{2\sigma^2}\right) \exp\left(-\frac{(y_j^E - f^E(\boldsymbol{\lambda}))^2}{2\sigma^2}\right)$$

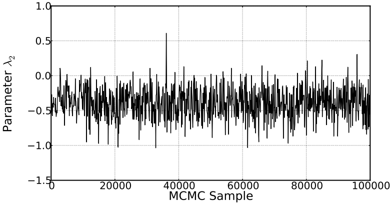
- Generate R random noisy realizations of $f^G(\boldsymbol{\lambda})$ and $f^E(\boldsymbol{\lambda})$ as data $\mathcal{D}$
- Assume Normally distributed priors $N(0, 0.3)$ on λ_1 and λ_2
- Infer λ_1 and λ_2 against data on both $f^G(\boldsymbol{\lambda})$ and $f^E(\boldsymbol{\lambda})$
- ```
  model_inf -x xfile.dat -y yfile.dat -f pc \
    -l classical -m 100000 -e 0.01 -i normal
  ```
  ```

Markov Chain Monte Carlo generates a set of posterior samples

λ_1

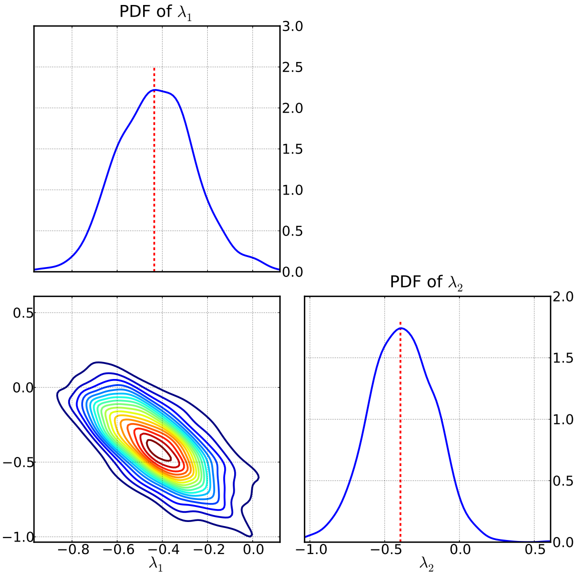


λ_2

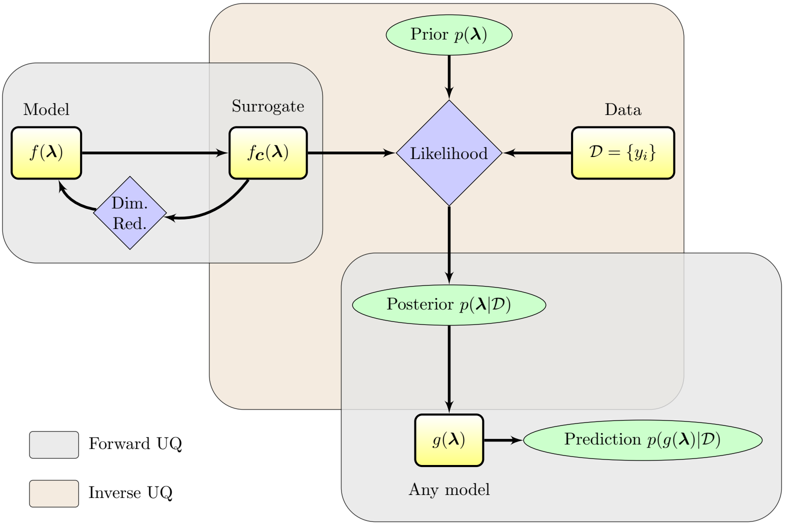


- The chains for both parameters show good mixing

Marginal and Joint Posterior Densities



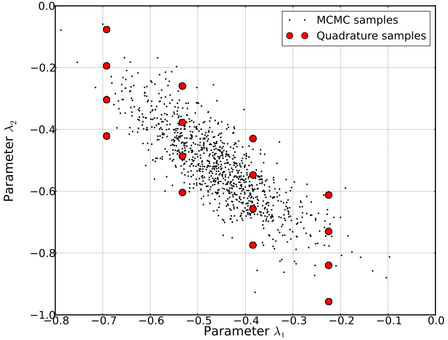
Forward propagation with calibrated parameters



The Rosenblatt transformation maps the posterior samples to standard Gaussian random variables

- Posterior distributions represent the parameter uncertainties
- Set of MCMC samples characterizes these posteriors
- Need to project these posteriors onto Gauss-Hermite PC basis to get PCE for $\boldsymbol{\lambda}$
  - Galerkin projection requires map between posterior samples and $\boldsymbol{\xi}$
  - Rosenblatt transformation provides this mapping
  - `pce_quad` provides this map to project samples onto PCEs

Rosenblatt mapping of quadrature points enables Galerkin projection onto Gauss-Hermite basis



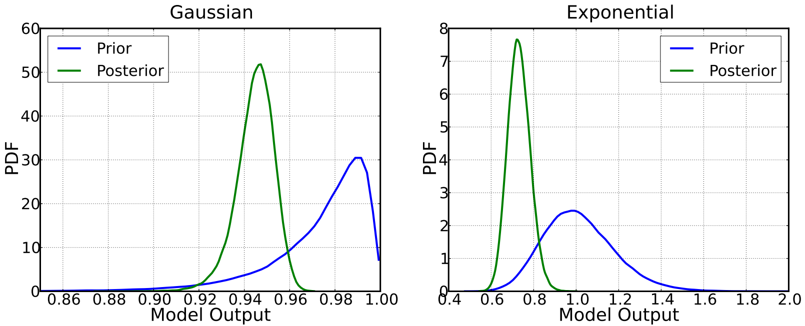
$$\lambda_1 = \sum_{k=0}^{P} \lambda_{1k} \Psi_k(\xi) \qquad \lambda_{1k} \propto \int \underbrace{R_{\lambda_1}^{-1}(\xi)}_{\lambda_1} \Psi_k(\xi) w(\xi) d\xi$$

$$\lambda_2 = \sum_{k=0}^{P} \lambda_{2k} \Psi_k(\xi) \qquad \lambda_{2k} \propto \int \underbrace{R_{\lambda_2}^{-1}(\xi)}_{\lambda_2} \Psi_k(\xi) w(\xi) d\xi$$

Forward propagation uses similar commands as surrogate construction

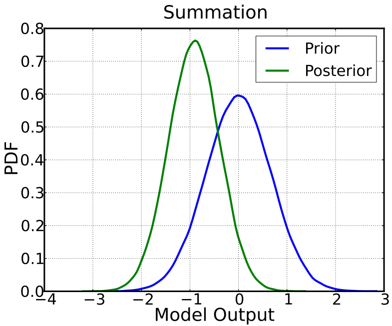
- Use calibrated uncertainty on λ_1, λ_2
- Keep prior uncertainty $N(0, 0.3)$ on $\lambda_3, \lambda_4, \lambda_5$
- Forward models $g(\boldsymbol{\lambda})$:
  - Gaussian: $g_1(\boldsymbol{\lambda}) = f^G(\boldsymbol{\lambda})$
  - Exponential: $g_2(\boldsymbol{\lambda}) = f^E(\boldsymbol{\lambda})$
  - Summation: $g_3(\boldsymbol{\lambda}) = \sum_{i=1}^{d} \lambda_i$
- Quadrature Galerkin projection
  - `generate_quad -d 5 -g HG -x full -p 4`
  - Evaluate model at quadrature points
  - `pce_resp -d 5 -x HG -o 3`

Input calibration reduces output uncertainty



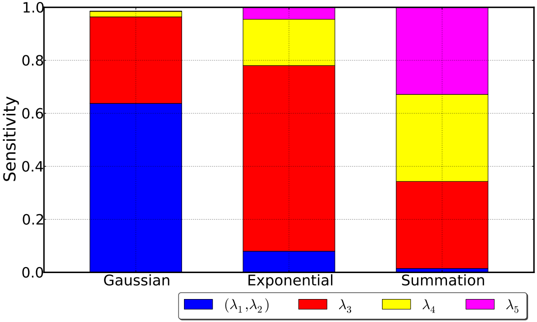
- After calibration: output distributions narrow and shift

Input calibration reduces output uncertainty



- After calibration: output distributions narrow and shift

Attribution relates output uncertainties to specific inputs



- Attribution uses sensitivity analysis tools: `pce_sens`
- For Gaussian model: more data needed to further reduce input uncertainty in λ_1, λ_2
- For other two models, need to calibrate other inputs

Summary

- UQTk provides a powerful set of tools for building general UQ workflows
- Multiple ways to access functionality
  - Direct linking of C++ code
  - Standalone apps
  - Python interface based on Swig (UQTk version 3.0)
- Version 3.0 soon at
  `http://www.sandia.gov/UQToolkit`
- Do not hesitate to contact us
  `uqtk-users@software.sandia.gov`

References

- B. Debusschere, H. Najm, P. Pébay, O. Knio, R. Ghanem and O. Le Maître, “Numerical Challenges in the Use of Polynomial Chaos Representations for Stochastic Processes", *SIAM J. Sci. Comp.*, 26:2, 2004.
- K. Sargsyan, *et al.*, “Dimensionality reduction for complex models via bayesian compressive sensing", *Int. J. of Uncertainty Quantification*, 4, 1:63-93, 2014.
- “Handbook of Uncertainty Quantification", R. Ghanem, D. Higdon, H. Owhadi (Eds.), Springer, 2016, `http://www.springer.com/us/book/9783319123844`
- UQ Tutorials: `http://www.quest-scidac.org/outreach/tutorials/`

Rosenblatt Transformation for Multi-D RVs

- Assume samples of multi-D RVs are (*e.g.* from MCMC sampling of posterior parameter distribution)
- Rosenblatt transformation maps any (not necessarily independent) set of random variables $(\lambda_1, \ldots, \lambda_d)$ to uniform i.i.d.'s $\{\eta_i\}_{i=1}^d$ (Rosenblatt, 1952).

$$
\begin{aligned}
\eta_1 &= F_1(\lambda_1) \\
\eta_2 &= F_{2|1}(\lambda_2|\lambda_1) \\
&\vdots \\
\eta_d &= F_{d|d-1,\ldots,1}(\lambda_d|\lambda_{d-1}, \ldots, \lambda_1)
\end{aligned}
$$

- Rosenblatt transformation is a multi-D generalization of 1D CDF mapping.
- Conditional CDFs are harder to evaluate in high dimensions