

# Gradient-Enhanced Uncertainty Propagation

*Mihai Anitescu and Oleg Roderick,*

*Jean Utke, Paul Hovland (MCS, Argonne)*

*Thomas Fanning (NE, Argonne)*

*Also acknowledging the work of Mihai Alexe (Virginia Tech), Brian Lockwood (Wyoming, CSGF)  
and Yiou Li (IIT)*

*VERSION OF 5/5/2011*

*FOR THE SAMSI WORKSHOP 2011*

# (Some ?) Components of the Uncertainty Quantification.

- Uncertainty analysis of model predictions: given data about uncertainty parameters  $u \in R^p$  and a code that creates output from it  $y = f(u)$  characterize  $y$ .
- Validation: Use data to test whether the UA model is appropriate or otherwise fit it or both.
  
- Challenge one: create model for  $u \in R^p$  from data. (Mihai's definition of UQ) It does not need to be probabilistic (see Helton and Oberkampf RESS special issue) but it tends to be. **What is the statistical model\*?**
- Challenge two: uncertainty propagation. Since  $f$  is expensive to compute, we cannot expect to compute a statistic of  $y$  very accurately from direct simulations alone (and there is also curse of dimensionality; exponential growth of effort with dimension). **How do I propagate the model if the code is very computationally intensive?**
- Challenge three: validation of uncertainty propagation model; or fitting/validation/testing. **What are the statistics I choose to test?**



# Faster Uncertainty Propagation by Using Derivative Information?

- Uncertainty propagation requires multiple runs of a possibly expensive code.
- On the other hand, adjoint differentiation adds a lot more information **per unit of cost** ( $O(p)$ , where  $p$  is the dimension of the uncertainty space; though needs lots of memory).
- Q: **Can I use derivative information in uncertainty propagation to accelerate its precision per unit of computing time.**
- We believe the answer is yes.
- Q: How?
- **By using surrogates built with derivative information.**



# Statistics with derivative information?

- We create surrogates out of expensive codes to carry out uncertainty propagation (truly, an approximation step)

- 

$$y = f(u) \Rightarrow y \approx \tilde{f}(u)$$

- If we do complete uncertainty analysis, we must create an error model for the surrogate itself.

$$f(u) - \tilde{f}(u) \sim \dots$$

- Not unlike other statistical activities, but one big difference with codes versus physical experiments: **I can compute derivatives**

- 

$$\frac{\partial f}{\partial u_i}$$

- So I can use derivatives in the creation of the statistical models, which is a fairly distinct endeavor in statistics (not unheard of but fairly rare).
- When I create a surrogate, I am in effect mining a code (perhaps even my own). With derivatives I do it (semi)-intrusively if I use automatic differentiation.
- Can I mine source code for extracting emerging structure of physical models (not pursued here)?



# Outline

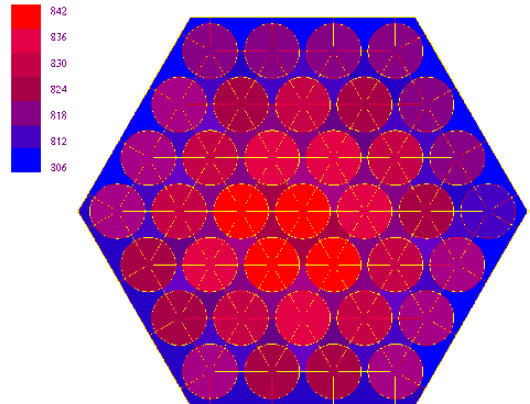
- 1. Polynomial regression with derivative PRD information: Uncertainty propagation using sensitivity information.
- 2. Obtaining derivative information: Automatic Differentiation of Codes with Substantial Legacy components.
- 3. PRD-based uncertainty propagation: Numerical examples.
- 4. What is a good basis in PRD? Limitations and Construction of Tensor Product Bases. Benefits.
- 5. Gaussian Processes for Quantifying Uncertainty Propagation Error.

**1. Polynomial regression with derivative  
PRD information: Uncertainty propagation  
using sensitivity information.**



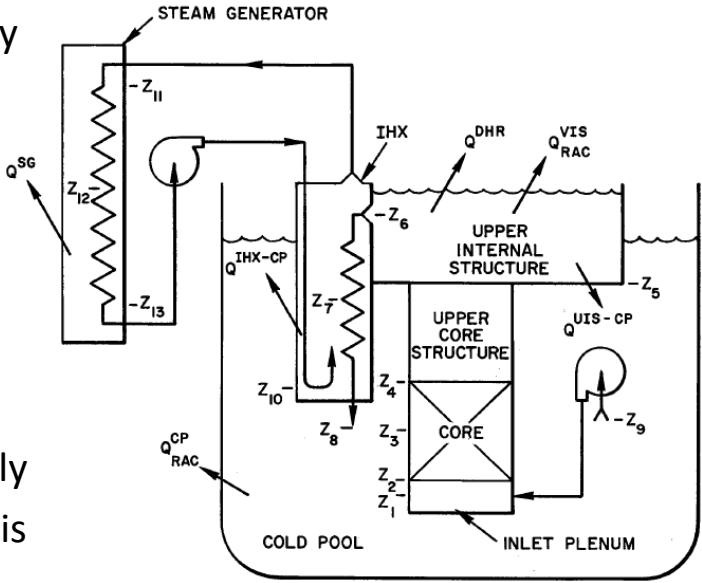
# Uncertainty quantification, subject models

■ Model I. Matlab prototype code: a steady-state 3-dimensional finite-volume model of the reactor core, taking into account heat transport and neutronic diffusion. Parameters with uncertainty are the material properties: heat conductivity, specific coolant heat, heat transfer coefficient, and neutronic parameters: fission, scattering, and absorption-removal cross-sections. Chemical non-homogeneity between fuel pins can be taken into account. Available experimental data is parameterized by 12-66 quantifiers.



■ Model II. MATWS, a functional subset of an industrial complexity code SAS4A/SASSYS-1: point kinetics module with a representation of heat removal system. >10,000 lines of Fortran 77, sparsely documented.

MATWS was used, in combination with a simulation tool Goldsim, to model nuclear reactor accident scenarios. The typical analysis task is to find out if the uncertainty resulting from the error in estimation of neutronic reactivity feedback coefficients is sufficiently small for confidence in safe reactor temperatures. The uncertainty is described by 4-10 parameters.



# Representing Uncertainty

- We use a hierarchical structure. Given a generic model with uncertainty

$$F(T, R) = 0$$

$$R = R(T) \cdot (1 + \Delta R(T, \alpha)) \quad J = J(T)$$

with model state  $T = (T_1, T_2, \dots, T_n)$   $R = (R_1, R_2, \dots, R_N)$

intermediate parameters and inputs

that include errors  $\Delta R = (\Delta R_1, \Delta R_2, \dots, \Delta R_N)$

An output of interest is expressed by the merit function  $J(T)$

The uncertainty is described by a set of stochastic quantifiers  $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_m)$

- We redefine the output as a function of uncertainty quantifiers,  $\mathfrak{J}(\alpha) := J(T)$   
and seek to approximate the unknown function  $\mathfrak{J}(\alpha)$





# Polynomial Regression with Derivatives, PRD

- We approximate the unknown response function by polynomial regression based on a small set of model evaluations. **Both merit function *outputs* and merit function *derivatives with respect to uncertainty quantifiers* are used as fitting conditions.**

- PRD procedure:

- choose a basis of multivariate polynomials  $\{\Psi_q(\alpha)\}$

the unknown function is then approximated by an expansion  $\mathfrak{Z}(\alpha) \approx \sum_q x_q \Psi_q(\alpha)$

- choose training set  $\{A\}$ ;  $A_i = (\alpha_1^i, \alpha_2^i, \dots, \alpha_n^i)$

- evaluate the model and its derivatives for each point in the training set, and enforce the collocation conditions on the training set.

$$\mathfrak{Z}(\alpha^i) = \sum_q x_q \Psi_q(\alpha^i); \frac{\partial}{\partial \alpha_j^i} \mathfrak{Z}(\alpha^i) = \sum_q x_q \frac{\partial}{\partial \alpha_j^i} \Psi_q(\alpha^i), j = 1, 2, \dots, d; i = 1, 2, \dots, N$$

- solve the regression equation (in the least-squares sense) to find coefficients  $x_q$

- Questions (for later):

- How to best choose the polynomial basis?
- How to obtain gradient information at computational cost comparable with that of a model run?



# Polynomial Regression with Derivatives, PRD

- PRD procedure, regression/ collocation\* equations:
- Note: the only interaction with the computationally expensive model is on the right side!
- The polynomial regression approach without derivative information would provide  $(n+1)$  times LESS rows.
- The overall computational savings depend on how cheaply the derivatives can be computed

$$\begin{pmatrix}
 \Psi_1(A_1) & \Psi_2(A_1) & \dots \\
 \frac{d\Psi_1(A_1)}{d\alpha_1} & \frac{d\Psi_2(A_1)}{d\alpha_1} & \dots \\
 \frac{d\Psi_1(A_1)}{d\alpha_2} & \frac{d\Psi_2(A_1)}{d\alpha_2} & \dots \\
 \vdots & \vdots & \vdots \\
 \frac{d\Psi_1(A_1)}{d\alpha_m} & \frac{d\Psi_2(A_1)}{d\alpha_m} & \dots \\
 \Psi_1(A_2) & \Psi_2(A_2) & \dots \\
 \frac{d\Psi_1(A_2)}{d\alpha_1} & \frac{d\Psi_2(A_2)}{d\alpha_1} & \dots \\
 \vdots & \vdots & \vdots \\
 \Psi_1(A_M) & \Psi_2(A_M) & \dots \\
 \vdots & \vdots & \vdots \\
 \frac{d\Psi_1(A_M)}{d\alpha_m} & \frac{d\Psi_2(A_M)}{d\alpha_m} & \dots
 \end{pmatrix} \cdot x = \begin{pmatrix}
 \mathfrak{I}(A_1) \\
 \frac{d\mathfrak{I}(A_1)}{d\alpha_1} \\
 \frac{d\mathfrak{I}(A_1)}{d\alpha_2} \\
 \vdots \\
 \frac{d\mathfrak{I}(A_1)}{d\alpha_m} \\
 \mathfrak{I}(A_2) \\
 \frac{d\mathfrak{I}(A_2)}{d\alpha_1} \\
 \vdots \\
 \mathfrak{I}(A_M) \\
 \vdots \\
 \frac{d\mathfrak{I}(A_M)}{d\alpha_m}
 \end{pmatrix}$$



# Cost versus benefit of using gradient information.

- Theory: Cost of gradient evaluation can be at most 5 times larger than cost of function evaluation. Therefore relative efficiency of using gradient information versus using one more sample point: at least  $d/5$ .
- This bound is achieved by adjoint calculations or reverse automatic differentiation mode.
- Sometimes, the bound is much smaller. Example: Coupling in multiphysics achieved by operator splitting/ Gauss Seidel since Newton method may not converge. Then compute adjoint at converged point.
- When instrumenting code for gradient with AD we are somewhere between intrusive and non-intrusive methods for UQ. Clearly not as simple as brute force sampling, but not as intrusive as Galerkin stochastic FEM methods either. **But payoff of same accuracy for fewer samples is a great driver.**
- In principle, can be operated without knowing what the code does, in practice, the latter helps.





**2. Obtaining derivative information:  
Automatic Differentiation of Codes with  
Substantial Legacy components.**



# PRD, computation of derivatives

- Hand-coding derivatives is error-prone, has large development cost, code maintenance is a problem.
- Finite difference approximations introduce truncation errors, and Cost of gradient  $\sim$  Dimension  $\times$  Cost of the function, and advantage of adjoints is lost.
- For most applied purposes, a more promising approach is Automatic (Algorithmic) Differentiation, AD. It also uses the chain-rule approach, but with minimal human involvement.
- Ideally, the only required processing is to identify inputs and outputs of interest, and resolve the errors at compilation of the model augmented with AD.



# Why not simply discard codes with legacy components?

- Since, focus is on new code. When new code is target, great, just press the button. Though even there the curse of the dimensionality requires some initial thought on whether/how gradient info will be obtained as a part of the design process. But what if “new code” is not the case?
- Reality:
  - High resolution codes are crucial for filling uncertainty gaps; but it is unlikely that system level assessments will be carried out with the highest resolution codes even if we have all the available foreseeable computing power. Or that one can simply run them at low resolution\*.
  - Also, new codes make take long time to get accepted at face value in a regulatory environment.
  - Plus, we will be forced at least to validate any uncertainty findings against a standard code, to demonstrate that at least to compatible resolutions the results are similar.
- **A possible future:** A suite of codes of decreasing resolution but increasing system complexity, with UQ propagating parameters from one to the others. With a true-and-tested systems code at one end.
- In any case, existing tools are likely to need to be at least sampled for comparison; and they are not spared the curse of dimensionality, and they need acceleration of propagation as provided here. Thus it will help tremendously if they are instrumented with gradients.
- Plus, it helps to know if PRD helps with a realistic uncertainty setting.



# Automatic Differentiation, AD

- AD is based on the fact that any program can be viewed as a finite sequence of elementary operations, the derivatives of which are known. A program  $P$  implementing the function  $J$  can be parsed into a sequence of elementary steps:

$$P: J = f_k(f_{k-1}(\dots f_1(\alpha)))$$

The task of AD is to assemble a new program  $P'$  to compute the derivative. In *forward* mode:

$$P': (\nabla_{\alpha} J)_i = \frac{\partial f_k}{\partial f_{k-1}} \cdot \frac{\partial f_{k-1}}{\partial f_{k-2}} \cdot \dots \cdot \frac{\partial f_1}{\partial \alpha_i}$$

- In the forward (or *direct*) mode, the derivative is assembled by the chain rule following computational flow from an input of interest to all outputs. We are more interested in the *reverse* (or *adjoint*) mode that follows the reversed version of the computational flow from an output to all inputs:

$$P': (\nabla_{\alpha} J) = \left( \frac{\partial f_1}{\partial \alpha} \right)^T \cdot \left( \frac{\partial f_2}{\partial f_1} \right)^T \cdot \dots \cdot \left( \frac{\partial f_k}{\partial f_{k-1}} \right)^T$$

In adjoint mode, the complete gradient can be computed in a single run of  $P'$ , as opposed to multiple runs required by the direct mode.

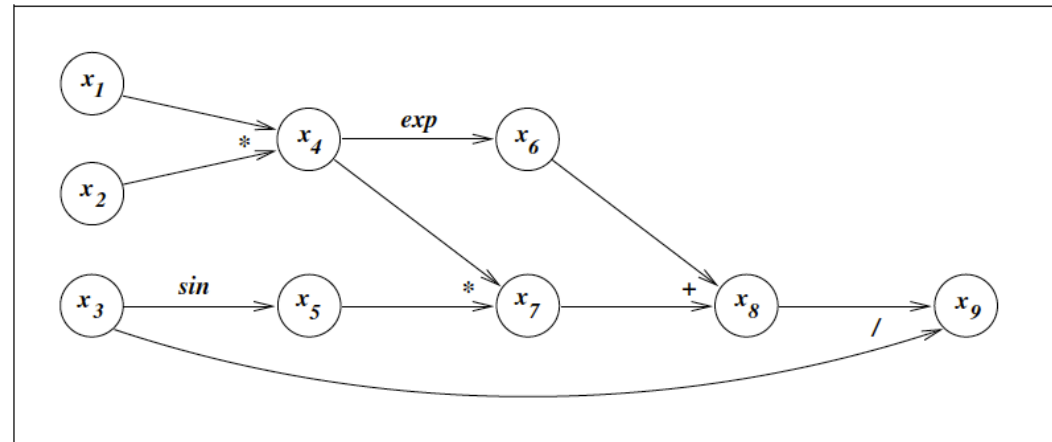
- For inherently non-differentiable components of code, it is possible to construct a smooth interpolation. **THIS is one of the many cases where it helps UQ to be integrated with a physics team which we believe and we practice** (We will not discuss nondifferentiability here).



# How (and why) does AD work, conceptually?

- Build a computational graph (example from Nocedal and Wright) and traverse it recursively left to right (forward) or right to left (adjoint).

$$f(x) = (x_1 x_2 \sin x_3 + e^{x_1 x_2}) / x_3.$$



$$x_4 = x_1 * x_2,$$

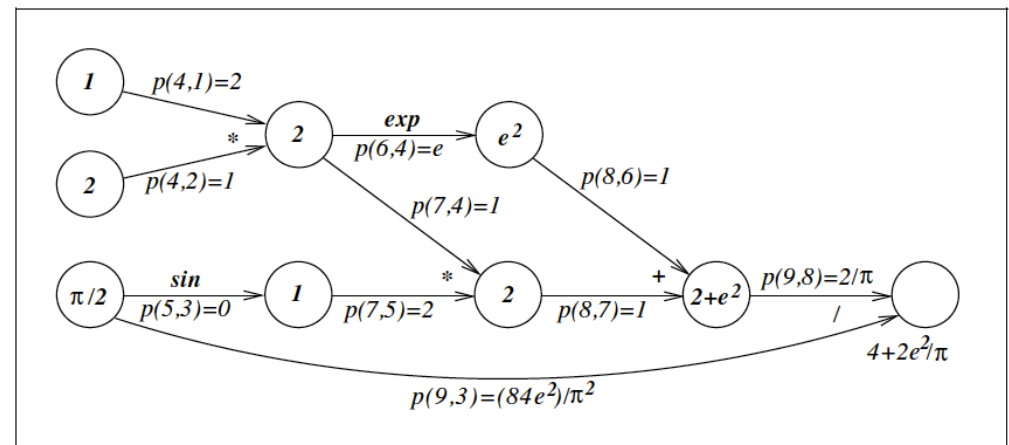
$$x_5 = \sin x_3,$$

$$x_6 = e^{x_4},$$

$$x_7 = x_4 * x_5,$$

$$x_8 = x_6 + x_7,$$

$$x_9 = x_8 / x_3.$$





# AD tools, Fortran

## ■ TAF (FastOpt)

- Commercial tool
- Support for almost all of Fortran 95
- Used extensively in geophysical sciences applications

## ■ Tapenade

- Support for many Fortran 95 features
- Developed by a team with extensive compiler experience

## ■ OpenAD/F

- Support for many Fortran 95 features
- Developed by a team with expertise in combinatorial algorithms, compilers, software engineering, and numerical analysis
- Development driven by climate modeling and astrophysics applications

## ■ ADIFOR

- Mature, very robust tool. Support for all of Fortran 77 :forward and adjoint modes
- Hundreds of users, over 250 citations



# AD tools, Capabilities

- Fast  $O(1)$  computation of
  - Gradient (in adjoint mode)
  - Derivative matrix-vector products
- Efficient computation of full Jacobians and Hessians, able to exploit sparsity, low-rank structure
- Efficient high-order directional derivative computation
- Minuses: it is still not a mature technology (after 30 years !!!) except for very specific cases (e.g codes written entirely in Fortran 77+ STANDARD).
- We believe in (and we practice) close integration with an AD development team (Jean Utke, Mihai Alexe)



# Applying AD to code with major legacy components

- We investigated the following question: are AD tools now at a stage where they can provide derivative information for realistic nuclear engineering codes? Many models of interest are complex, sparsely documented, and developed according to older (Fortran 77) standards.
- Based on our experience with MATWS, the following (Fortran 77) features make application of AD difficult:
  - Not supported by AD tools (since they are nonstandard) /need to be changed.
    - machine-dependence code sections need to be removed (i/o)
    - Direct memory copy operations needs to be rewritten as explicit operations (when LOC is used)
    - COMMON blocks with inconsistent sizes between subroutines need to be renamed
    - Subroutines with variable number of parameters need to be split into separate subroutines
  - EQUIVALENCE, COMMON, IMPLICIT\* definitions are supported by most tools though they have to be changed for some (such as OpenAD). (for Open AD statement functions need to be replaced by subroutine definitions, they are not supported in newer Fortran)
  - Note that the problematic features we encountered have to do with memory allocation and management and i/o, not mathematical structure of the model! We expect that (differentiable) mathematical sequences of any complexity can be differentiated.



# Validation of AD derivative calculation

- Model II, MATWS, subset of SAS4A/SASSYS-1. We show estimates for the derivatives of the fuel and coolant maximum temperatures with respect to the radial core expansion coefficient, obtained by different AD tools, and compared with the Finite Differences approximation, FD.

All results agree with FD within 0.001% (and almost perfectly with each other).

AD tool	Fuel temperature derivative, K	Coolant temperature derivative, K
ADIFOR	18312.5474227	17468.4511373
OpenAD/F	18312.5474227	17468.4511372
TAMC	18312.5474248	17468.4511392
TAPENADE	18312.5474227	17468.4511372
FD	18312.5269537	17468.4315994



### 3. PRD-based uncertainty propagation: Numerical examples.



# Why am I obsessed with really low sample size ?

- We work in project “Simulation-Based High-Efficiency Advanced Reactor Prototyping”;
- Some of the codes that need to be validated run for a few weeks on a supercomputer for one sample.
- So we must have methods that give \*some\* idea of uncertainty for 5-50 samples even for large-ish dimensional uncertainty spaces.

# PRD UQ, tests on subject models 1.

- Model I, Matlab prototype code. Output of interest: maximal fuel centerline temperature.
- We show performance of a version with 12 (most important) uncertainty quantifiers. Performance of PRD approximation with full and truncated basis is compared against random sampling approach (100 samples)\*:

	Sampling	Linear approximation	PRD, full basis	PRD, truncated basis
Full model runs	100	1*	72*	12*
Output range, K	2237.8 2460.5	2227.4 2450.0	2237.8 2460.5	2237.5 2459.6
Error range, K		-10.38 +0.01	-0.02 +0.02	-0.90 +0.90
Error st. deviation		2.99	0.01	0.29

\* derivative evaluations required ~150% overhead



# PRD, basis truncation

- Issue: we would like to use high-order polynomials to represent non-linear relationships in the model. But, even with the use of derivative information, the required size of the training set grows rapidly (**curse of dimensionality in spectral space**)
- We use **a heuristic**: we rank uncertainty quantifiers by importance (a form of sensitivity analysis is already available, for free!) and use an incomplete basis, i.e. polynomials of high degree only in variables of high importance. This allows the use of **some** polynomials of high degree (maybe up to 5?)
- Several versions of the heuristic are available, we choose to fit a given computational budget on the evaluations of the model to form a training set.
- In our first experiments, we use either a complete basis of order up to 3, or its truncated version allowing the size of training set to be within 10-50 evaluations.
- An even better scheme - adaptive basis truncation based on stepwise fitting is developed later, simultaneously with conditions for better algebraic form of multivariate basis,





# Uncertainty quantification, tests on subject models

- Model II, MATWS, subset of SAS4A/SASSYS-1. We repeat the analysis of effects of uncertainty in an accident scenario modeled by MATWS + GoldSim. The task is to estimate statistical distribution of peak fuel temperature.

- We reproduce the distribution of the outputs correctly\*;

regression constructed on 50 model

evaluations thus replaces analysis

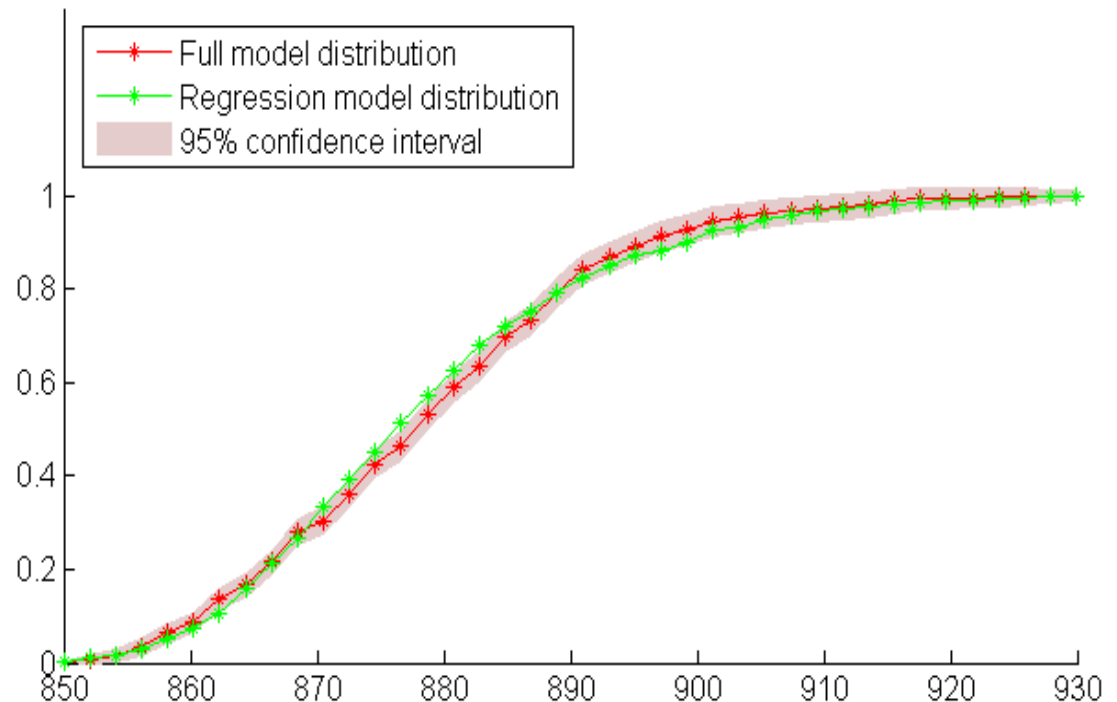
with 1,000 model runs. We show

cumulative distribution of the

peak fuel temperature.

- Note that the PRD approximation is almost entirely within the 95% confidence interval of the sampling-based results.

- Surface response, error model in progress (though control variate done)



## 4. What is a good basis in PRD?



# PRD, selection of better basis

- We inherited the use of Hermite multivariate polynomials as basis from a related method: Stochastic Finite Elements expansion.
- While performance of PRD so far is acceptable, Hermite basis may not be a good choice for constructing a regression matrix with derivative information; it causes poor condition number of linear equations (of the Fischer matrix).
- Hermite polynomials are generated by orthogonalization process, to be orthogonal (in probability measure  $\rho$ ; Gaussian measure is the specific choice):

$$\int_{\Omega} \Psi_j(A) \Psi_h(A) \rho(A) dA = \delta_{jh}$$

- We formulate new orthogonality conditions:

$$\int_{\Omega} \left( \Psi_j(A) \Psi_h(A) + \sum_{i=1}^m \frac{\partial \Psi_j(A)}{\alpha_i} \cdot \frac{\partial \Psi_h(A)}{\alpha_i} \right) \rho(A) dA = \delta_{jh}$$

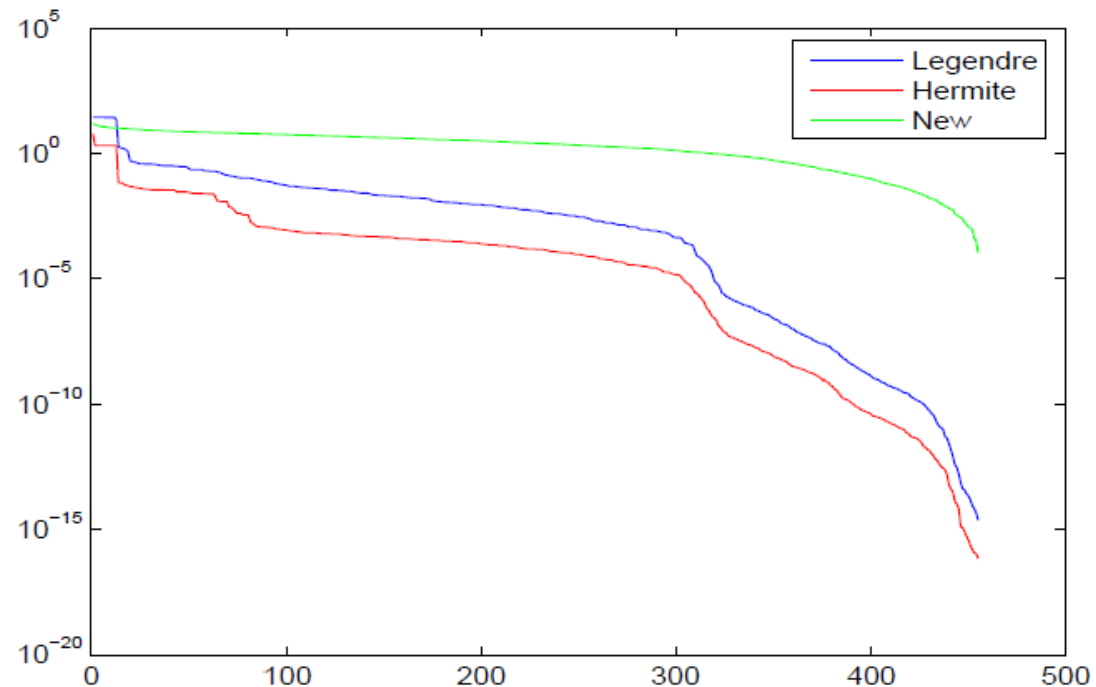
and ask the question: how does a good basis with respect to this inner product looks like?

- **Surprise: We cannot construct tensor product bases of arbitrary order.** We give very tight sufficient conditions and use them.



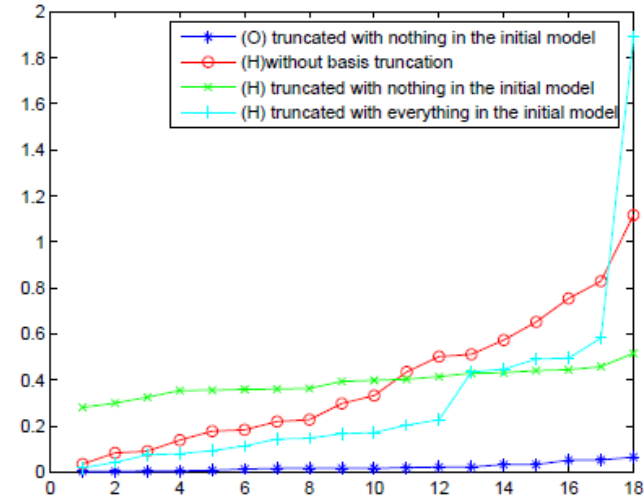
# PRD, selection of better basis

- Model I, Matlab prototype code. We compare the setup of PRD method using Hermite polynomial basis and the improved basis. We observe the improvement in the distribution of singular values of the collocation matrix.
- We compare numerical conditioning for Hermite, Legendre polynomials, and the basis based on new orthogonality conditions.
- **We have  $10^{10}$  improvement in the condition number of the Fischer matrix \*!!! In principle this results in much more robustness of the matrix.**
- This will offer us substantial flexibility in creating the PRD model.

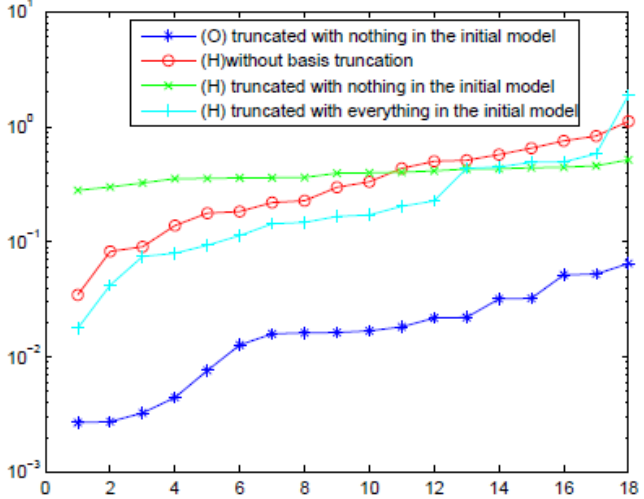


# PRD, adaptive (stepwise fitting) basis truncation

- We use a stepwise fitting procedure (based on F-test):
  1. Create the PRD model as an expansion in the starting set of polynomials
  2. Add one (estimated as most likely) polynomial to the set. An expansion term currently not in the model is added if, out of all candidates, it has the largest likelihood that it would have non-negligible coefficient if added to model.
  3. Remove one (estimated as least likely) polynomial from the set. An expansion term in the model is removed if it has the highest likelihood to have negligible coefficient.
- It is possible to truncate the model starting with a full basis set (of fixed maximal polynomial order) or from an empty basis set (all polynomials of fixed maximal order are candidates to be added).



(Hermite basis error on 20 samples)



(Orthogonal basis error on 20 samples, log<sub>10</sub> plot)

- Orthogonal basis created starting “with nothing” in the expansion results in precision of up to 0.01 degree K (compare with errors of >10 K by linear model).



## 5. Gaussian Processes for Quantifying Uncertainty Propagation Error.



# PRD: need for enhancement, need for error model

- PRD approach has been shown to be a powerful tool, (precision of  $<0.1\%$  ? For a nonlinear 12-dimensional model? Based on a training set of size 10?)
- But it does not address the bias introduced and the clearly when you fit a model PRD, which one knows is not exactly correct. Also, the correlation model is clearly incorrect (error in derivatives uncorrelated with errors in function evaluation? )
- We thus need to improve uncertainty quantification on the uncertainty propagation process.
- We start from good surrogate model –as we demonstrated -- which we enhance with a Gaussian Process model and fit it with max likelihood. If the covariance is smooth enough, I have a consistent model for function and gradient error.
- Then, we use the posterior prediction (kriging) at the test points



# Gaussian-processes based error modeling, preview

- Notation: GP – Gaussian Processes; GEK – gradient-enhanced Kriging
- In the framework of GP we assume that the response of the system can be represented as a Gaussian process with explicit mean function and specified covariance function governed by a set of parameters (hyperparameters). **But Ordinary kriging by itself is not immune to curse of dimensionality !**
- We use an explicit mean model (universal kriging), which, if it is a good approximation, **drastically reduces exposure to the curse**. Finding values of hyperparameters leads to explicit covariance function (an algebraic form has to be assumed).
- Covariance matrix on the training set now also includes covariances between points in the uncertainty space, between points and gradients of the output function, and between gradients and gradients.





# Recap: Modeling Uncertainty Propagation

- Idea: use the few samples to model the system response.
- Ansatz: system response is “smooth” so maybe I can get by with a limited number of samples.
- But since I do uncertainty quantification, **I must model the error I am making.**
- Possible approaches:
  - Use sensitivity approach  $\mathfrak{I}(\alpha) \sim \mathfrak{I}(\alpha_0) + \nabla_{\alpha} \mathfrak{I}(\alpha_0)(\alpha - \alpha_0) + \dots$   
With adjoints,  $O(1)$  “big code” runs, but large bias, and error hard to model.
  - Use L2 regression  $\mathfrak{I}(\alpha) \sim \sum_{k=1, K} a_k \psi_k(\alpha)$ . Robust and least subject to curse of dimensionality, but introduces bias whose error model is hard to gauge.
  - Use Kriging.  $\mathfrak{I}(\alpha) \sim GP(0, k(\alpha, \alpha'; \theta)); \tilde{\mathfrak{I}}(\alpha) \sim GP(0, k(\alpha, \alpha'; \theta)) \left\{ \mathfrak{I}(\alpha_i) \right\}_{i=1}^N$ . The bias goes to 0 under some conditions, but it is an interpolation, so subject to COD.
- But ... **what if we use all 3?** Gradient-enhanced universal Kriging (GEUK).  
Hopefully, small exposure to COD (as regression), small/reducible bias as Kriging  
efficient use of information (as sensitivity).

# Preview of the method

(A) First-order gradient information on the model

**PLUS**

(B) Polynomial mean function

**PLUS**

(C) Uncertainty Propagation Model using Gaussian Processes, Kriging

=

**Gradient-Enhanced Universal (non-constant mean function) Kriging, GEUK**



# Gaussian Processes approach, technical details:

- We assume that the response of the system can be represented as a Gaussian process with explicit mean function and specified covariance function governed by a set of parameters (*hyperparameters*):

$$\left\{ \hat{\mathcal{Z}}(x_1), \hat{\mathcal{Z}}(x_2), \dots, \hat{\mathcal{Z}}(x_N) \right\} \sim N(R(X)a, K(X, X; \theta)); \quad K(x_i, x_j) = k(x_i - x_j)$$

- Covariance matrix with derivative information is given by a block form:

$$K = \text{cov}[Y, Y] = \begin{pmatrix} \text{cov}[J, J] & \text{cov}[J, \nabla J] \\ \text{cov}[\nabla J, J] & \text{cov}[\nabla J, \nabla J] \end{pmatrix}$$

- Regression parameters are computed as  $a = (\Psi^T K^{-1} \Psi)^{-1} \Psi^T K^{-1} Y$   
 or  $a = (H^T K^{-1} H)^{-1} \cdot H^T K^{-1} \cdot Y$ , with  $H = \begin{pmatrix} \Psi \\ \nabla \Psi \end{pmatrix}$   $Y = \begin{pmatrix} J \\ \nabla J \end{pmatrix}$

- The mean and variance of the model are now predicted as

$$\mu[J] = \left( \text{cov}(Y_{i,:}, Y_{:,j}) \quad W \right) \cdot K^{-1} Y + R(x)a$$

$$\text{var}[J] = \text{cov}(S, S) - \left( \text{cov}(Y_{i,:}, Y_{:,j}) \quad W \right) \cdot K^{-1} \cdot \begin{pmatrix} \text{cov}(Y_{i,:}, Y_{:,j}) \\ W \end{pmatrix} + R(x) \left( H^T K^{-1} H \right)^{-1} R(x)^T$$

- We now need to assume a functional form of the covariance function **which must be positive definite** (not a trivial requirement: truncation of a pd function is not pd).

- ,For example: squared exponential:

$$\text{cov}(S_i, S_j; \theta) = \exp \left[ - \left( \frac{S_i - S_j}{\theta_{ij}} \right)^2 \right]$$



# How to compute the covariance of the derivative information

- First, the covariance function must support differentiable realizations. We will consider here only stationary covariance functions.

$$k(x, x') = k(x - x')$$

- The covariance function (of a stationary process) must be differentiable at 0 twice as many times as the realizations.
- E.g: The process is twice differentiable everywhere → the covariance function must be four times differentiable at 0.
- For first-order derivative:

$$\text{cov}(y, y') = k(x, x') \quad \text{cov}\left(\frac{\partial y}{\partial x_k}, y'\right) = \frac{\partial}{\partial x_k} k(x, x') \quad \text{cov}\left(\frac{\partial y}{\partial x_k}, \frac{\partial y'}{\partial x'_l}\right) = \frac{\partial^2}{\partial x_k \partial x'_l} k(x, x').$$

# Gaussian Processes approach, technical details:

- With the functional form of covariance specified, the hyperparameters  $\theta$  are determined by maximizing the marginal likelihood function for the data. The logarithm of the likelihood is given by:

$$\log(p(J|S; \theta) = -\frac{1}{2} Y^T K^{-1} Y + \frac{1}{2} Y^T K^{-1} H (H^T K^{-1} H)^{-1} H^T K Y - \frac{1}{2} \log|K| - \frac{m}{2} \log(2\pi)$$

- The optimization is carried out using standard tools (L-BFGS + active set algorithm).
- Computationally expensive parts of GP process: inverse of the covariance matrix, optimization problem, and very high resolution sampling. **No part of the GP process scales at high resolution in current implementations, due to reliance on explicit, dense Cholesky.**
- Both can be/have been accelerated in future work:
  - Cholesky of covariance matrix, matrix-free calculation of  $Q^{0.5} * N(0, I)$  (Chen, Anitescu, Saad, to appear in SISC) – not really attacked before.
  - Max Likelihood can be done scalably using a trace estimator (Anitescu, Chen, Wang, 2011).
  - These approaches are based on solves with covariance matrix which may be *perfectly* preconditioned (Stein, Chen, Anitescu, 2011).
  - Better optimization solver?
- But in current setup, expensive part is still sampling the code.



# How do I choose the polynomials that represent the mean function?

- ... Using PRD.
- Functional form:  $R(x)a = \sum_{k=1, K} a_k \psi_k(x)$
- Here,  $\{\psi_k(x)\}_{k=1}^K$  is a set of independent polynomials.

# How do I choose the covariance function?

- Squared Exponential:

$$k_i(x_i - x'_i) = e^{-\left(\frac{x_i - x'_i}{\theta_i}\right)^2}$$

- Matern Function  $\nu = \frac{3}{2}$ :

$$k_i(x_i - x'_i) = \left(1 + \sqrt{3} \left| \frac{x_i - x'_i}{\theta_i} \right| \right) e^{-\sqrt{3} \left| \frac{x_i - x'_i}{\theta_i} \right|}$$

- Matern Function  $\nu = \frac{5}{2}$ :

$$k_i(x_i - x'_i) = \left(1 + \sqrt{5} \left| \frac{x_i - x'_i}{\theta_i} \right| + \frac{5}{3} \left| \frac{x_i - x'_i}{\theta_i} \right|^2 \right) e^{-\sqrt{5} \left| \frac{x_i - x'_i}{\theta_i} \right|}$$

- Covariance functions must be “positive definite”.
- The square exponential is one of the most used in machine learning, but also assumes the underlying process is very smooth, which may make the error estimate completely unreliable.
- The Matern function is one of the most robust, for the derivative-free case and it has controllable smoothness.

# How do I choose the covariance function II.

- Cubic Spline 1:

$$k_i(x_i - x'_i) = \begin{cases} 1 - 15 \left| \frac{x_i - x'_i}{\theta_i} \right|^2 + 30 \left| \frac{x_i - x'_i}{\theta_i} \right|^3 & \text{for } 0 \leq \left| \frac{x_i - x'_i}{\theta_i} \right| \leq 0.2 \\ 1.25 \left( 1 - \left| \frac{x_i - x'_i}{\theta_i} \right| \right)^3 & \text{for } 0.2 \leq \left| \frac{x_i - x'_i}{\theta_i} \right| \leq 1 \\ 0 & \text{for } \left| \frac{x_i - x'_i}{\theta_i} \right| \geq 1 \end{cases}$$

- Cubic Spline 2:

$$k_i(x_i - x'_i) = \begin{cases} 1 - 6 \left| \frac{x_i - x'_i}{\theta_i} \right|^2 + 6 \left| \frac{x_i - x'_i}{\theta_i} \right|^3 & \text{for } 0 \leq \left| \frac{x_i - x'_i}{\theta_i} \right| \leq 0.5 \\ 2 \left( 1 - \left| \frac{x_i - x'_i}{\theta_i} \right| \right)^3 & \text{for } 0.5 \leq \left| \frac{x_i - x'_i}{\theta_i} \right| \leq 1 \\ 0 & \text{for } \left| \frac{x_i - x'_i}{\theta_i} \right| \geq 1 \end{cases}$$

- All the previous kernel functions result in DENSE matrices which may be a problem if I need to sample at many points.
- Cubic spline functions are examples of compact kernels, with sparse covariance matrices that can be more easy to manipulate (e.g Cholesky, which is needed in max likelihood and sampling, may be doable)..



# 5.1 DIGRESSION: SCALABLE GP ANALYSIS

# Some responses to computational challenges in GPs.

- How do you compute the likelihood at a million sites? How do you even STORE the matrix?

$$\log(p(J|S;\theta) = -\frac{1}{2}Y^T K^{-1}Y + \frac{1}{2}Y^T K^{-1}H(H^T K^{-1}H)^{-1}H^T KY - \frac{1}{2}\log|K| - \frac{m}{2}\log(2\pi)$$

- Matrix free for determinant? There seems to be no solution here.
- Idea for max likelihood: solve trace equations ....

$$0 = \frac{\partial}{\partial \theta_j} \log p(\mathbf{y} | \boldsymbol{\theta}) = \frac{1}{2} \text{tr} \left( \left( (K(\boldsymbol{\theta})^{-1} \mathbf{y})(K(\boldsymbol{\theta})^{-1} \mathbf{y})^T - K(\boldsymbol{\theta})^{-1} \right) \frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j} \right)$$

- ... using a statistical estimator (Hutchinson) for the trace, with  $u$  a Rademacher random variable.

$$\text{tr}(A) = E_u [u^T A u]; \quad p(u_i = 1) = p(u_i = -1) = \frac{1}{2}$$

# Scalable max likelihood calculation

- Use Hutchinson on the score equations.

$$0 = \sum_{i=1}^N \mathbf{u}_i^T \left( \left( (K(\boldsymbol{\theta})^{-1} \mathbf{y})(K(\boldsymbol{\theta})^{-1} \mathbf{y})^T - K(\boldsymbol{\theta})^{-1} \right) \frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j} \right) \mathbf{u}_i$$

- Seen as the sample average approximation of the stochastic nonlinear equations:

$$\mathbf{0} = \mathbb{E}_{\mathbf{u}} [\mathbf{F}(\boldsymbol{\theta}, \mathbf{u})],$$

where for  $j = 1, 2, \dots, n_{\theta}$ ,

$$F_j(\boldsymbol{\theta}, \mathbf{u}) = \frac{1}{2} \mathbf{u}^T \left( \left( (K(\boldsymbol{\theta})^{-1} \mathbf{y})(K(\boldsymbol{\theta})^{-1} \mathbf{y})^T - K(\boldsymbol{\theta})^{-1} \right) \frac{\partial K(\boldsymbol{\theta})}{\partial \theta_j} \right) \mathbf{u}.$$

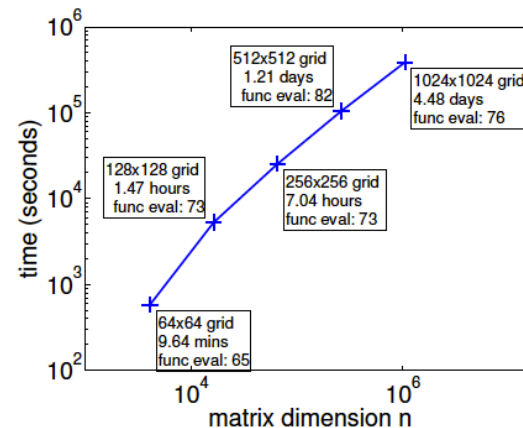
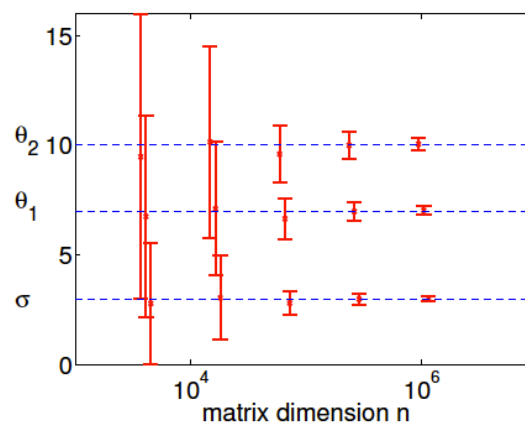
- Which satisfies the convergence in distribution:

$$N^{\frac{1}{2}} [V^N]^{-\frac{1}{2}} (\boldsymbol{\theta}^N(\omega) - \boldsymbol{\theta}^*) \xrightarrow{D} \mathcal{N}(\mathbf{0}, I), \quad V^N = [J^N]^{-T} \Sigma^N [J^N]^{-1}$$

# Scalable max likelihood results

- So (almost) all complexity to solving systems with the covariance matrix, if I I can get by a small sample size of the Rademacher vectors.
- Issues: for efficiency and scalability
  - Preconditioning: optimal preconditioning for excellent preconditioning with BCCB
  - $O(N \log(N))$  matrix-vector multiplication: for dense kernels, we can use fast multipole methods.
- In conclusion: there is hope for  $O(N \log(N))$  complexity and storage for carrying out the max likelihood (and sampling), as opposed to at least  $O(N^3)$  and  $O(N^2)$  for Cholesky !

$$\text{cond}(KM^{-1}) = O(1)$$



# 5.2 NUMERICAL RESULTS FOR GEUK

# Our working intuition re GEUK

- H1: GEUK results in less error compared with  $L_2$  regression (GP with iid noise).
- H2: GEUK results in less error compared with universal Kriging without derivative information. Idea: one gradient evaluation brings  $d/5$  more information.
- H3: GEUK results in less error for the same number of sample values when compared with ordinary Kriging.
- H4. GEUK approximates well the statistics of output, and its predicted covariance is a good or conservative estimate of the error}.
- H5. Covariance matters. It will affect the predictions and usability of the model. Idea: it is best to assume as little differentiability as one can get by with, particularly in the dense limit of samples.
  
- Approach: calibrate with  $N$  samples, report error with 500.

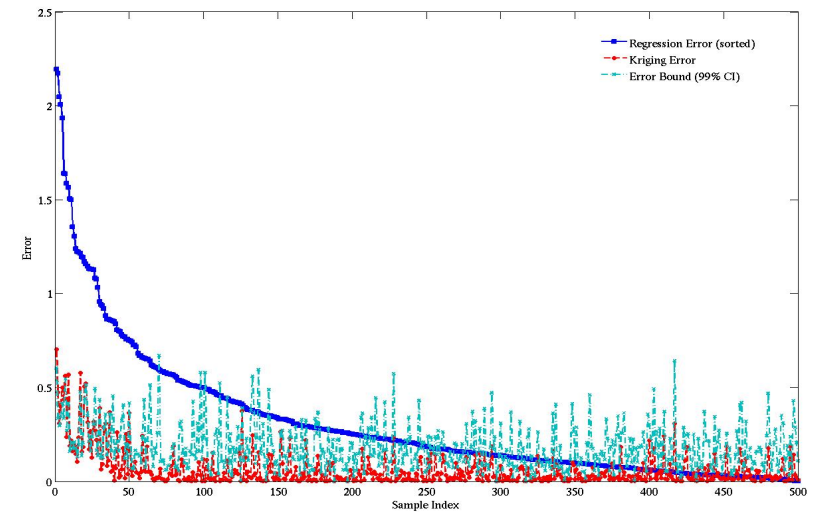
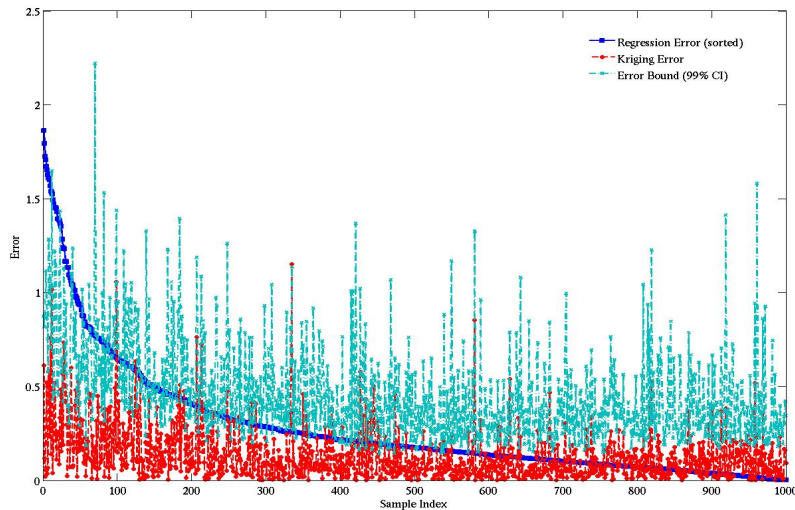
# H1: Kriging versus Regression

Table 3.9: MATWS – cubic spline 2 – comparison of error for Kriging and regression models

Sample Points	Kriging RMS	Regression RMS	Kriging Max	Regression Max
4 (p=2)	3.6433	15.2304	13.7491	56.6632
6 (p=2)	0.5260	3.2833	2.2040	14.0380
8 (p=3, trunc)	0.1841	0.5695	1.1980	3.1272
16	0.0766	0.427	0.747	2.404
24	0.0887	0.405	0.910	1.877
32	0.0995	0.309	1.118	1.959
40	0.0517	0.295	0.437	2.112
50	0.0508	0.251	0.386	1.476
100	0.0337	0.181	0.0998	1.068

Table 3.3: MATLAB – square exponential – Comparison of error for Kriging and regression models

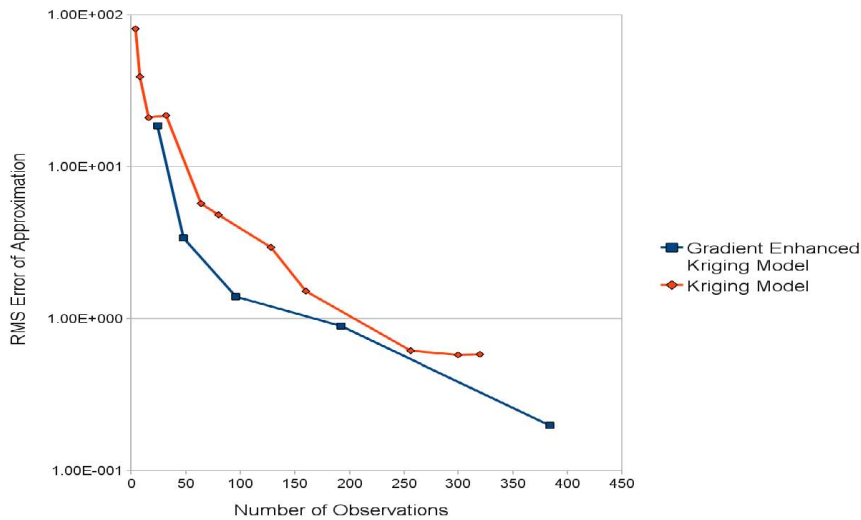
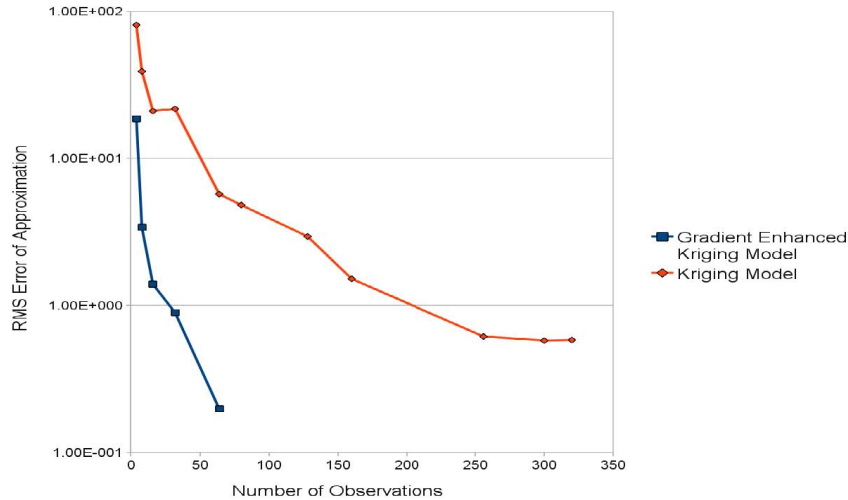
Data Set	Kriging RMS	Regression RMS	Kriging Max	Regression Max
1	0.11554	0.47118	0.70207	2.194
2	0.58351	0.76058	2.5731	3.2553
3	0.77163	1.1982	3.2202	4.8668
4	0.77163	1.289	3.2204	5.0067



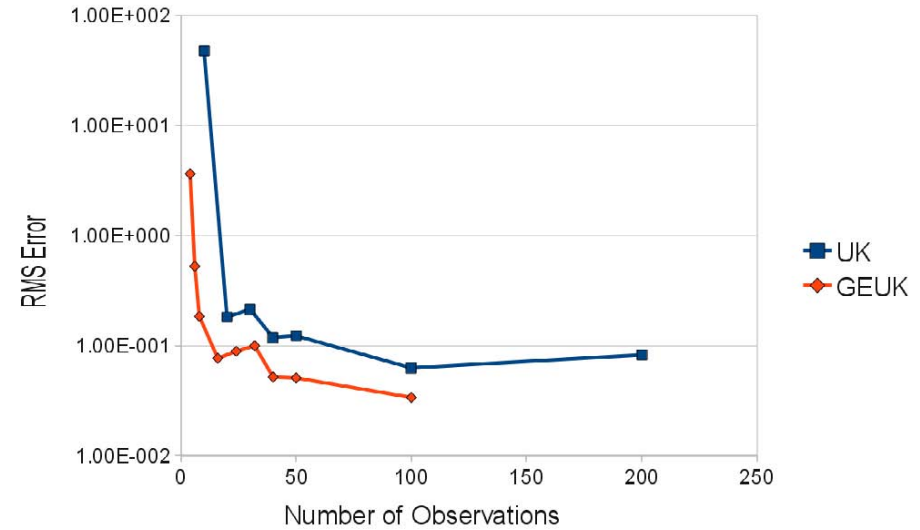
Conclusion H1: GEUK better RMSE – modeling correlation matters

# H2: Effect of gradient information

MATLAB



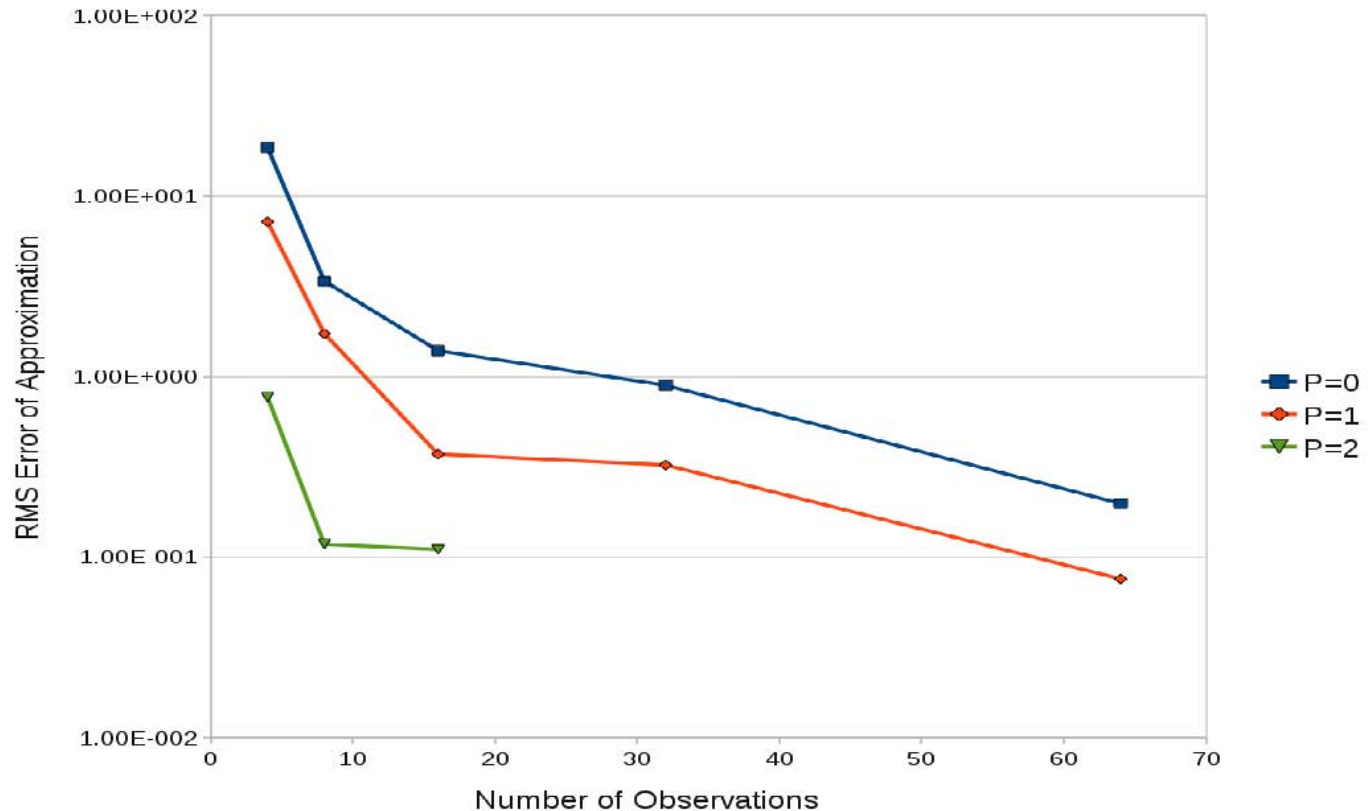
MATWS



Conclusion H2: gradient matters somewhat but we expect it will be even more spectacular for large dimensions.



# H3: Using a mean function versus ordinary kriging



Conclusion H3: Modeling the mean does matter

# H4: Kriging gives me the a good approximation of the error

Table 3.5: MATLAB – square exponential – statistics for Kriging prediction

Data Set	$\pm 1\sigma$	$\pm 2\sigma$	$\pm 3\sigma$
1	0.690	0.882	0.950
2	0.378	0.568	0.668
3	0.362	0.626	0.720
4	0.362	0.626	0.720

Table 3.7: MATLAB – comparison of data distribution between covariance functions

Covariance Function	$\pm 1\sigma$	$\pm 2\sigma$	$\pm 3\sigma$
Cubic Spline 1	0.290	0.592	0.758
Cubic Spline 2	0.776	0.878	0.930
Squared Exponential	0.690	0.882	0.95
Matern-3/2	0.676	0.874	0.932
Matern-5/2	0.704	0.884	0.928

Table 3.16: MATWS – Matern-3/2 – Z scores for Kriging Prediction

Data Set	KS metric	$\pm 1\sigma^*$	$\pm 2\sigma^*$	$\pm 3\sigma^*$
4 (p=2)	0.5580	0.0030	0.0090	0.0150
6 (p=2)	0.2782	0.2510	0.4780	0.6030
8	0.1557	0.4640	0.7070	0.8130
16	0.0645	0.6150	0.8810	0.9510
24	0.0297	0.6890	0.9450	0.9840
32	0.0770	0.8200	0.9690	0.9840
40	0.0269	0.7150	0.9590	0.9900
50	0.0601	0.7890	0.9870	1.0000

Table 3.12: MATWS – cubic spline 2 – statistics for kriging prediction

Data Set	$\pm 1\sigma$	$\pm 2\sigma$	$\pm 3\sigma$
4 (p=2)	0.847	0.977	0.999
6 (p=2)	0.513	0.964	1.000
8 (p=3, trunc)	0.599	0.968	1.000
16	0.697	0.942	1.000
24	0.533	0.857	0.971
32	0.525	0.817	0.942
40	0.475	0.800	0.937
50	0.366	0.685	0.870
100	0.221	0.424	0.600

Conclusion H4: We have good approximation at low sample size, and, with decorrelation, good approximation at larger sample size for some functions.

# H5: The choice of covariance function matters

Table 3.10: MATWS – comparison of covariance functions for 8 pt GEUK model

Covariance Function	RMS Error	Max Error
Cubic Spline 1	0.2083	1.3567
Cubic Spline 2	0.1841	1.1980
Squared Exponential	0.1245	1.0125
Matern-3/2	0.1704	1.0645
Matern-5/2	0.1530	1.0566

Table 3.11: MATWS – comparison of covariance functions for 50 pt GEUK model

Covariance Function	RMS Error	Max Error
Cubic Spline 1	0.0567	0.3963
Cubic Spline 2	0.0528	0.4551
Squared Exponential	0.1487	2.0268
Matern-3/2	0.0398	0.2552
Matern-5/2	0.0749	0.7991

Table 3.8: MATLAB – square exponential – statistics for Kriging prediction using decc

Covariance Function	KS Metric	$\pm 1\sigma^*$	$\pm 2\sigma^*$	$\pm 3\sigma^*$
Cubic Spline 1	0.2939	0.3080	0.4520	0.5960
Cubic Spline 2	0.2766	0.2820	0.3800	0.6540
Squared Exponential	0.1412	0.4880	0.7220	0.8380
Matern-3/2	0.1974	0.4200	0.6820	0.8320
Matern-5/2	0.2383	0.3300	0.5700	0.7160

Conclusion for H5 (see also previous slide) – it does. As in the derivative-free case, Matern 3/2 seems a “safe” choice; squared exponential is all over the place, and compact kernel does not do a good job on the tails at larger N.

# Sampling the surrogate model when propagation uncertainty is included

- For uncertainty parameter  $\mathbf{u}$ , the error propagation is deterministic conditional on  $\mathbf{u}$

$$f(u_1), f(u_2), \dots, f(u_N) | (u_1, u_2, \dots, u_N) \sim \delta(f(u_1), f(u_2), \dots, f(u_N))$$

- When using GP to model surrogate error, we have that

$$f(u_1), f(u_2), \dots, f(u_N) | (u_1, u_2, \dots, u_N) \sim N(m^c(U), K^c(U))$$

# Quantile Estimation

- This is a critical statistic in nuclear engineering.
- Particularly, the 95% statistics with 95 % confidence.
- “Conservative Estimate” using the Uniform distribution and properties of order statistics and uniform distributions quantiles.

Table 4.1: MATLAB – cubic spline 2 – quantile calculation for MATLAB data

Sample Points	Regression Order	Kriging Estimate	Regression Estimate	Training Estimate
4	2	2446.6	2447.2	2323.8
6	2	2448.2	2447.5	2335.2
8	3	2449.1	2448.4	2360.4

Actual Value = 2456.0

Table 4.2: MATWS – cubic spline 2 – quantile calculation for MATWS data

Sample Points	Kriging Estimate	Regression Estimate	Training Estimate
4 (p=2)	865.73	864.78	863.55
6 (p=2)	865.86	871.15	863.55
8	866.08	866.60	863.46
16	865.89	866.51	865.45
24	865.83	866.49	865.56
32	865.87	866.32	865.76
40	865.82	866.37	865.86
50	865.83	866.42	865.86

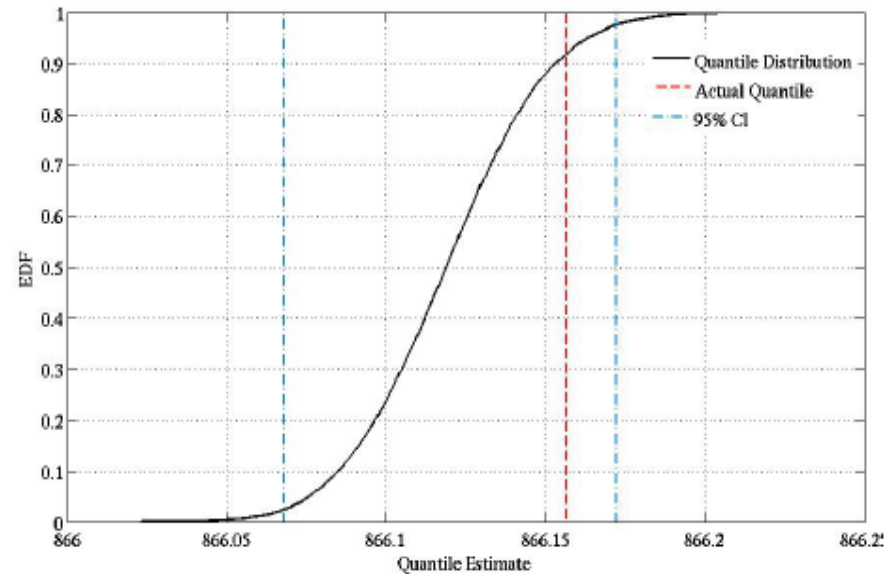
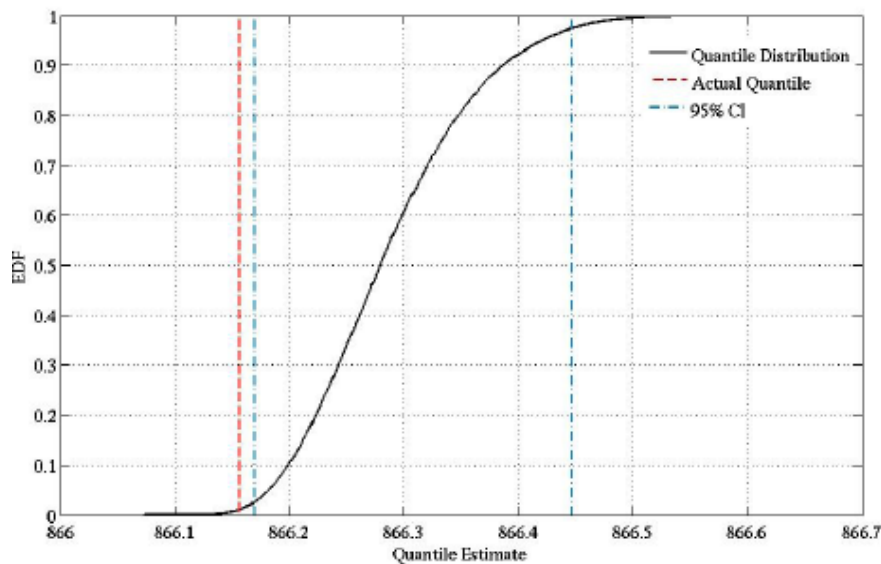
Actual Value = 866.16

# Estimating Quantiles Using Asymptotic tests

Table 4.5: Quantile – Matern-3/2 – Example results for Kriging quantile estimate with confidence interval

DATA	Training Points	Confidence(1- $\alpha$ )	Lower Bound	Upper Bound	Median
MATWS	8	0.95	866.17	866.45	866.28
MATWS	8	0.99	866.17	866.45	866.28
MATWS	50	0.95	866.07	866.17	866.12
MATLAB	8	0.95	2454.3	2455.6	2445.0
MATLAB	8	0.99	2454.2	2.455.8	2455.0

MATWS: 8 samples and 50 samples



# Conclusions

- Gradient-enhanced uncertainty propagation is a first step to a larger effort in learning the behavior of complex models by extracting more information from fewer sample runs.
- An important part of PRD and GEUK is Automatic Differentiation; it can be applied to codes of \*industrial\* complexity.
- We have shown that basis choice makes a difference for PRD.
- Gradient-enhanced universal kriging brings combines the best advantages in sensitivity, regression, and Gaussian processes.
- It can provide good statistics for nuclear engineering codes with 6-8 samples for the limited examples we tried.
- More accurate than regression, more efficient than kriging.
- Future:
  - Larger number of parameters
  - Approximated the gradients of very large scale codes. ]

# References for GEUK

- Brian Lockwood and Mihai Anitescu. "Gradient-Enhanced Universal Kriging for Uncertainty Propagation". [Preprint P1808-1110, Pdf Version.](#)
- Yiou Li, Mihai Anitescu, Oleg Roderick and Fred Hickernell, "ORTHOGONAL BASES FOR POLYNOMIAL REGRESSION WITH DERIVATIVE INFORMATION IN UNCERTAINTY QUANTIFICATION". To appear in the *International Journal for Uncertainty Quantification*. Also, Preprint ANL/MCS P1806-1110.
- Brian A. Lockwood and Mihai Anitescu. Gradient-Enhanced Universal Kriging for Uncertainty Propagation in Nuclear Engineering. To appear in *Transactions of the American Nuclear Society*. Preprint ANL/MCS-P1833-0111.
- Mihai Alexe, Oleg Roderick, Mihai Anitescu, Jean Utke, Thomas Fanning, and Paul Hovland. "Using Automatic Differentiation in Sensitivity Analysis of Nuclear Simulation Models". *Transactions of American Nuclear Society*, volume 102, pages 235-237, (2010)
- Oleg Roderick, Mihai Anitescu, and Paul Fischer. "Stochastic finite element approaches using derivative information for uncertainty quantification" Preprint ANL/MCS 1551-1008. *Nuclear Science and Engineering*, Volume 164 (2), 2010, Pages 122-139.



# References for Scalable GP

- Michael Stein, Jie Chen, and Mihai Anitescu. Difference Filter Preconditioning For Large Covariance Matrices. Preprint ANL/MCS-1888-0511. [Pdf Version.](#)
- Mihai Anitescu , Jie Chen , and Lei Wang , A Matrix-Free Approach For Solving The Gaussian Process Maximum Likelihood Problem. Preprint ANL/MCS-P1857-0311 [. Pdf Version.](#)
- Jie Chen, Mihai Anitescu and Yousef Saad. Computing  $F(a)b$  Via Least Squares Polynomial Approximations .*SIAM Journal on Scientific Computing*. 33, pp. 195-222 (2011).