Directions in Uncertainty Quantification and Massive Data Analysis involving Gaussian Processes

Mihai Anitescu (MCS, Argonne),

With Jean Utke, Paul Hovland, Oleg Roderick, Jie Chen, Emil Constantinescu (MCS, Argonne)
Thomas Fanning (NE, Argonne)

Brian Lockwood (Wyoming, CSGF) Mihai Alexe (Virginia Tech), Yiou Li and Fred Hickernell (IIT)
Michael Stein (UCHICAGO)

VERSION OF 4/13/2012 *At University of Pittsburgh, April 13, 2012*

Outline

1. Uncertainty Propagation using Derivative Information

Context: Uncertainty Propagation for Computationally Expensive Codes.

- 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.

- In this work we are interested in the propagation problem: Given a probability structure for $u \in R^p$ find the distribution of $y = f(u)$

- If 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?

Can Derivative Information Help in Accelerating Uncertainty Propagation?

- Adjoint differentiation adds a lot more information per unit of function evaluation cost (theory: at least p/5 more, where p is the dimension of the uncertainty space).
- Q: Can I use derivative information in uncertainty propagation to accelerate its precision per unit of computing time?
- Working hypothesis (and answer) : yes.

- In nuclear engineering, the answer tends to be either A) Use Monte Carlo or B) Linearized Functions + Adjoint Information.
- Q: How do I use gradient information without introducing the bias from linearization?
- By using surrogates—surface response – output collocation built with derivative information.

New research direction: 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.

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

- 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).
- Therefore some of the typical statistical endeavors (choosing predictors, kernels for Kriging, designs, etc) need to then be studied for the case where derivative information is also used.

How to obtain Derivatives? 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:

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

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

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

- 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:

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

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.

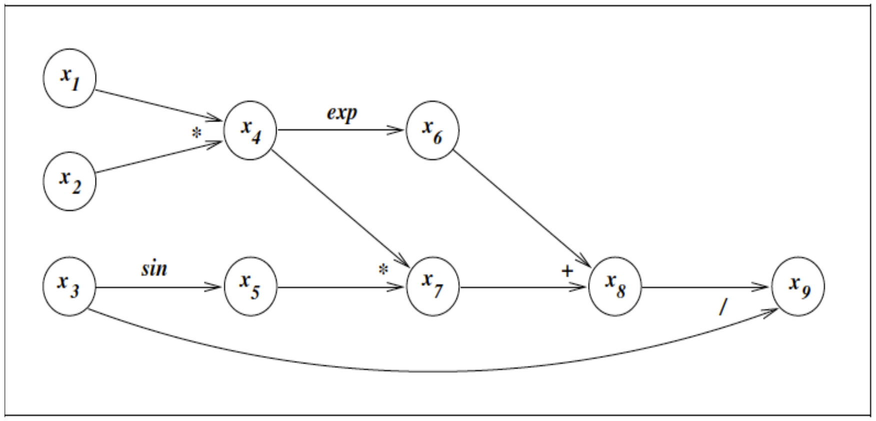
- Theory: Cost of adjoint < 5* Cost of Computing J. For some examples: (Lockwood, Mavriplis, et al.) < .01*Cost of Computing J.

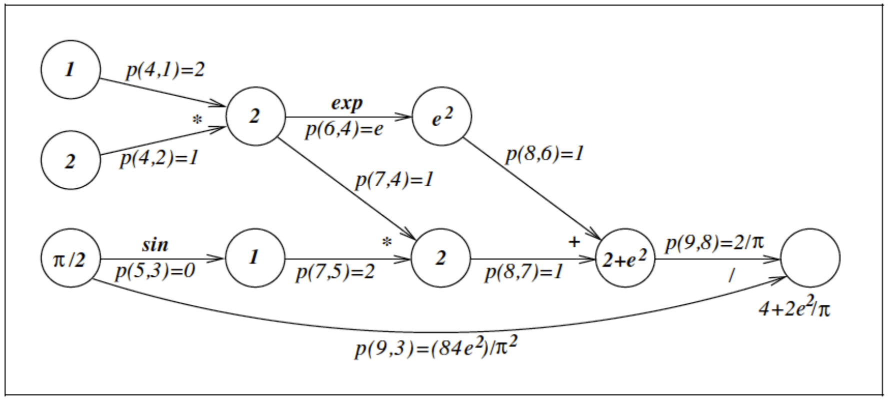
How (and why) does AD work, conceptually?

- Build a computational graph (example from Nocedal and Wright)
- Attach to the node the value of the variable and to the edge the value of "local partial derivative.
- Traverse it left to right (forward sensitivity) or right to left (adjoint sensitivity) and multiply the edge weights you encounter.
- Result: the forward or adjoint sensitivity.

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

$$\begin{aligned} x_4 &= x_1 * x_2, \\ x_5 &= \sin x_3, \\ x_6 &= e^{x_4}, \\ x_7 &= x_4 * x_5, \\ x_8 &= x_6 + x_7, \\ x_9 &= x_8 / x_3. \end{aligned}$$





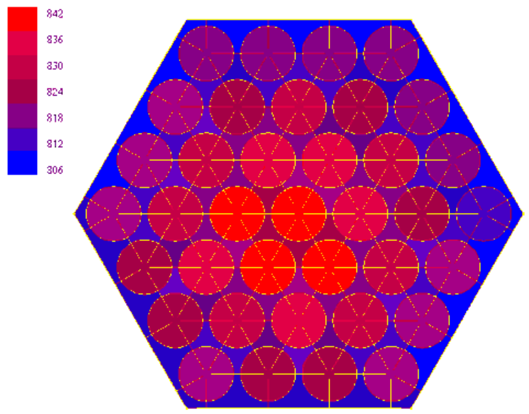
New research in the chain rule?

- The computational consequences of implementing adjoints efficiently are enormous.
- Costs of less than 0.01 of forward simulation have been reported for adjoints of certain physical phenomena (Lockwood, Mavriplis et al, 2012.).
  - How do we deal with storage complexity in adjoint calculation?
  - Should we do some compression, and what are best tools?
  - How do we parallelize and load balance multiphysics adjoints.
- Chain rule: Still a very rich algorithmic research area.

2. Polynomial regression with derivative information: PRD.

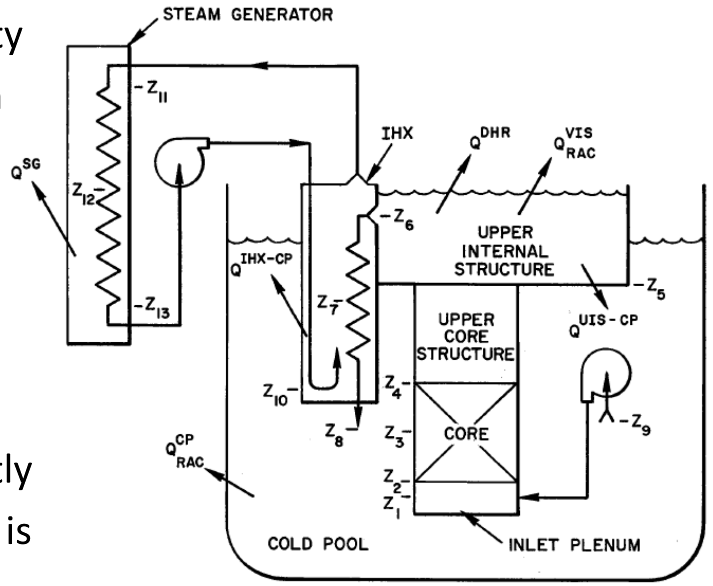
Uncertainty quantification, subject models

- 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 absorbtion-removal cross-sections. Chemical non-homogenuity 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

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

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

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

intermediate parameters and inputs

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

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

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

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

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 $\Im(\alpha) \approx \sum_q x_q \Psi_q(\alpha)$

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

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

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

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.

- Choose the polynomial basis wisely, solve with least squares.

- The overall computational savings depend on how cheaply the derivatives can be computed

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

Why am I obsessed with really low sample size ?

- We work in project “Simulation-Based High-Efficiency Advanced Reactor Prototyping -- SHARP”;
- 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.

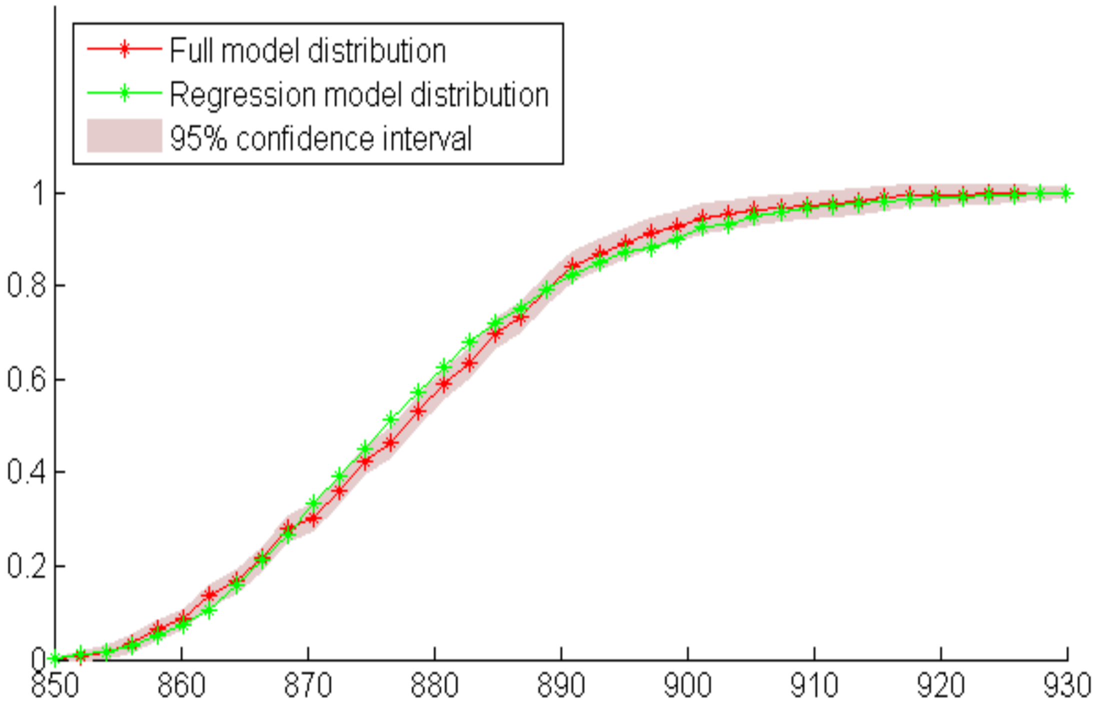
- In our first experiments, we use either a complete basis of order up to 3, or its truncated version allowing the size of training set to be within 10-50 evaluations.

- 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)

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 ρ; Gaussian measure is the specific choice):
$$\int_{\Omega} \Psi_j(A)\Psi_h(A)\rho(A)dA = \delta_{jh}$$

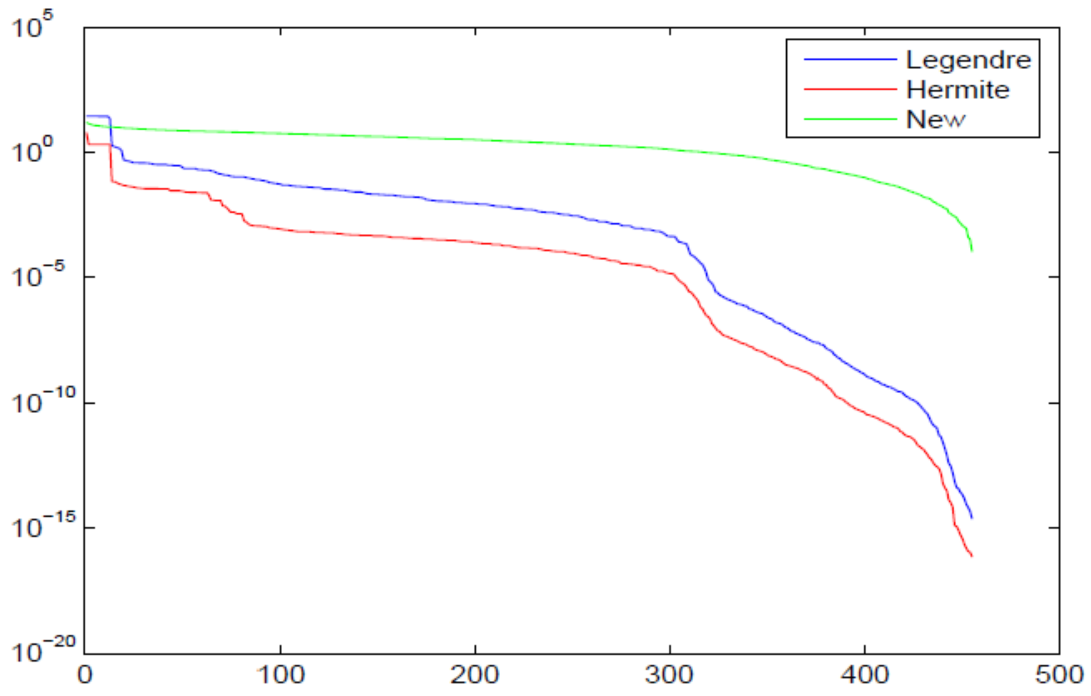
- We formulate new orthogonality conditions:
$$\int_{\Omega}\left(\Psi_j(A)\Psi_h(A) + \sum_{i=1}^{m} \frac{\partial \Psi_j(A)}{\alpha_i} \cdot \frac{\partial \Psi_h(A)}{\alpha_i}\right)\rho(A)dA = \delta_{ih}$$

  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. (Li & al., IJUQ, in press)

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.

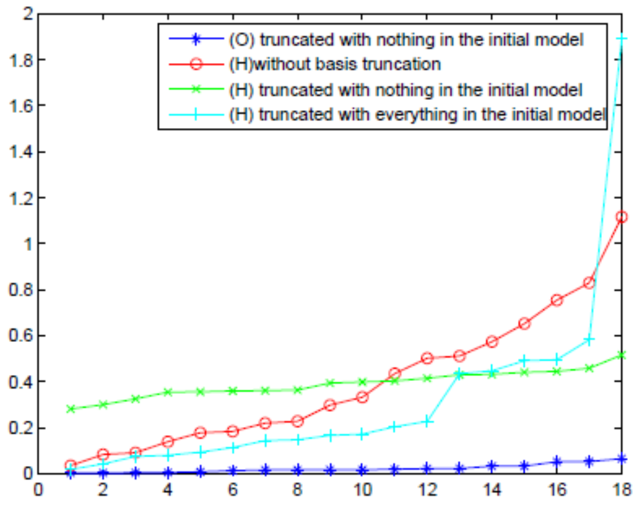


PRD, adaptive (stepwise fitting) basis truncation

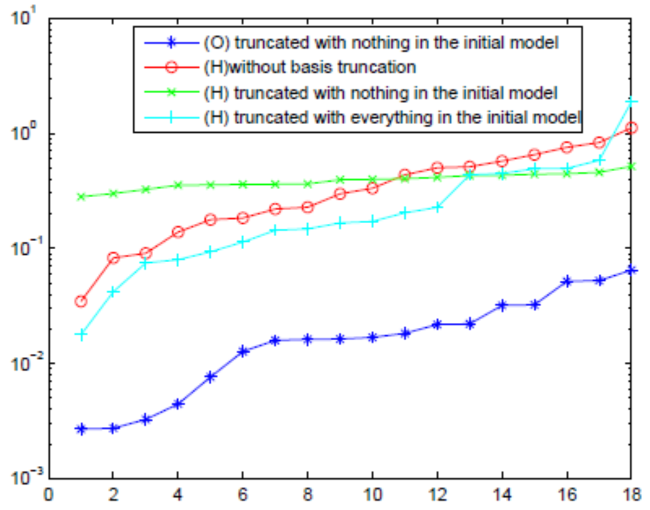
- We use a stepwise fitting procedure (based on F-test):

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_10 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).

3. 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 process regression (kriging): Setup

- Gaussian process (GP): $f(x) \sim \mathcal{N}(m(x), k(x,x'))$

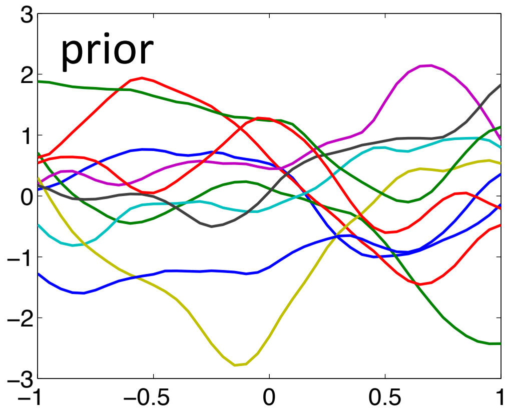
$$\mathrm{E}\{f(x)\} = \overline{f(x)} = m(x)$$
$$\mathrm{Cov}(f(x)) = k(x,x')$$

- Data (observations)/predictions: $y = f(x) + \varepsilon \ / \ y_* = f(x_*)$

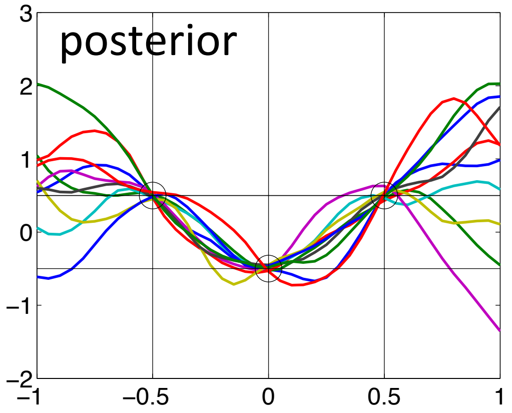
- GP joint distribution: $$\begin{bmatrix} y \\ y_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m}(X) \\ \mathbf{m}(X_*) \end{bmatrix}, \left[\begin{array}{c|c} \mathbf{K}_{11} + \Sigma & \mathbf{K}_{12} \\ \hline \mathbf{K}_{21} & \mathbf{K}_{22} \end{array} \right] \right)$$

- Predictive distribution: $$\overline{\mathbf{y}_* | \mathbf{X}, \mathbf{X}_*, \mathbf{y}} = \mathbf{m}(X_*) + \mathbf{K}_{21} \left(\mathbf{K}_{11} + \Sigma\right)^{-1} (\mathbf{y} - \mathbf{m}(X))$$

$$\mathrm{Cov}(\mathbf{y}_* | \mathbf{X}, \mathbf{X}_*, \mathbf{y}) = \mathbf{K}_{22} - \mathbf{K}_{21} \left(\mathbf{K}_{11} + \Sigma\right)^{-1} \mathbf{K}_{12}$$



$$y = [\ 0.5 \quad -0.5 \quad 0.5\ ]$$
$$x = [\ -0.5 \quad 0 \quad 0.5\ ]$$



Gaussian Processes with Derivatives,

- 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*):

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

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

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

- Regression parameters are computed as $a=(\Psi^T K^{-1}\Psi)\Psi^T K^{-1}Y$ or $a=\left(H^T K^{-1}H\right)^{-1}\cdot H^T K^{-1}\cdot Y$, 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

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

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

- 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:

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

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:

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

Gaussian Processes approach, Parameter Fitting

- With the functional form of covariance specified, the hyperparameters θ 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).

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)..

3.2 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. For full details, see Lockwood and Anitescu 2012.

H1: Kriging versus Regression

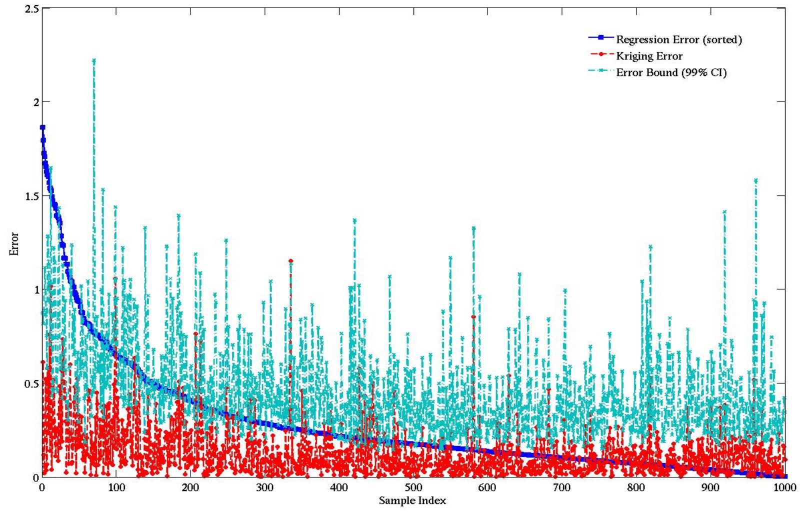
Table 3.9: MATWS – cubic spline 2 – comparison of error for Kriging and regression models

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

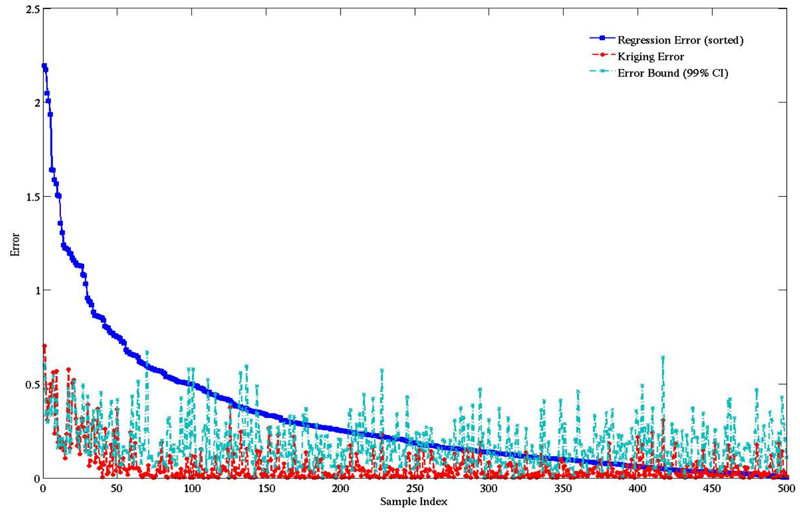
Table 3.3: MATLAB – square exponential – Comparison of error for Kriging and regression models

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

MS-3rd Deg – 16 pts – Cubic Spline 2



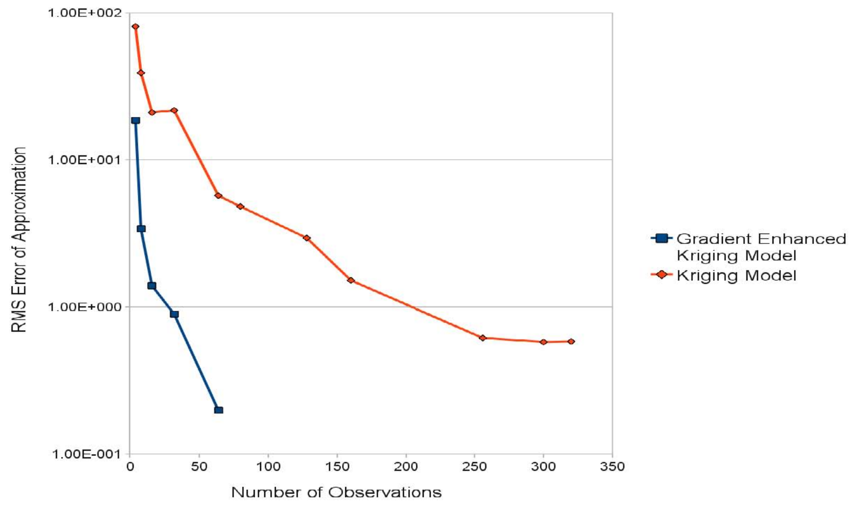
MB-3rd Deg – 8 pts – Square EXp

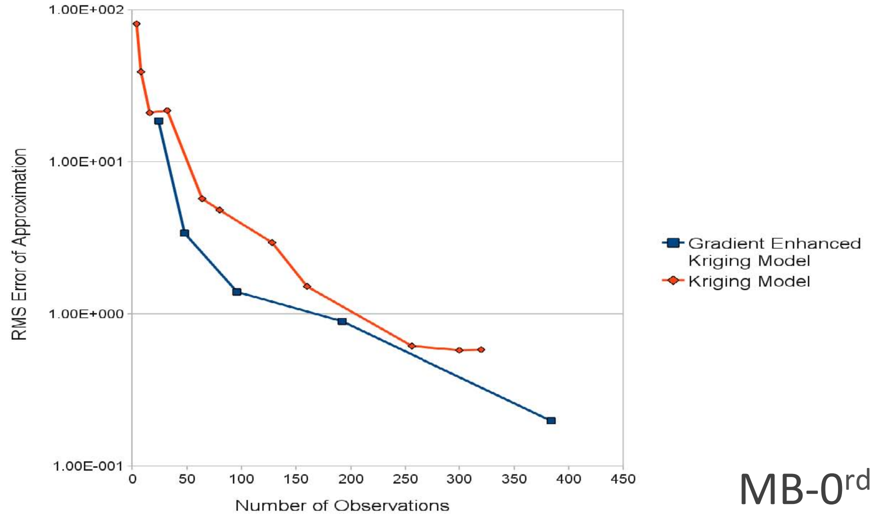


Conclusion H1: GEUK better RMSE – modeling correlation matters

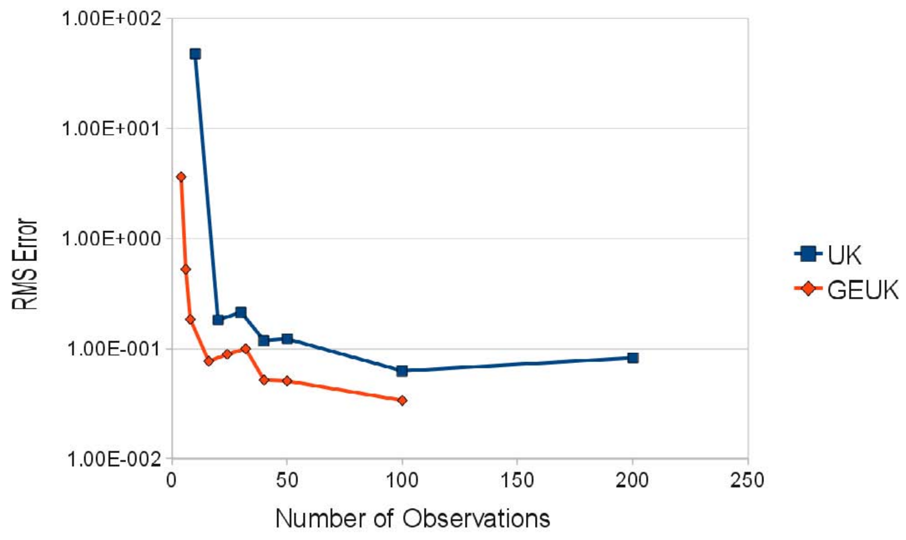
H2: Effect of gradient information

MATLAB





MATWS

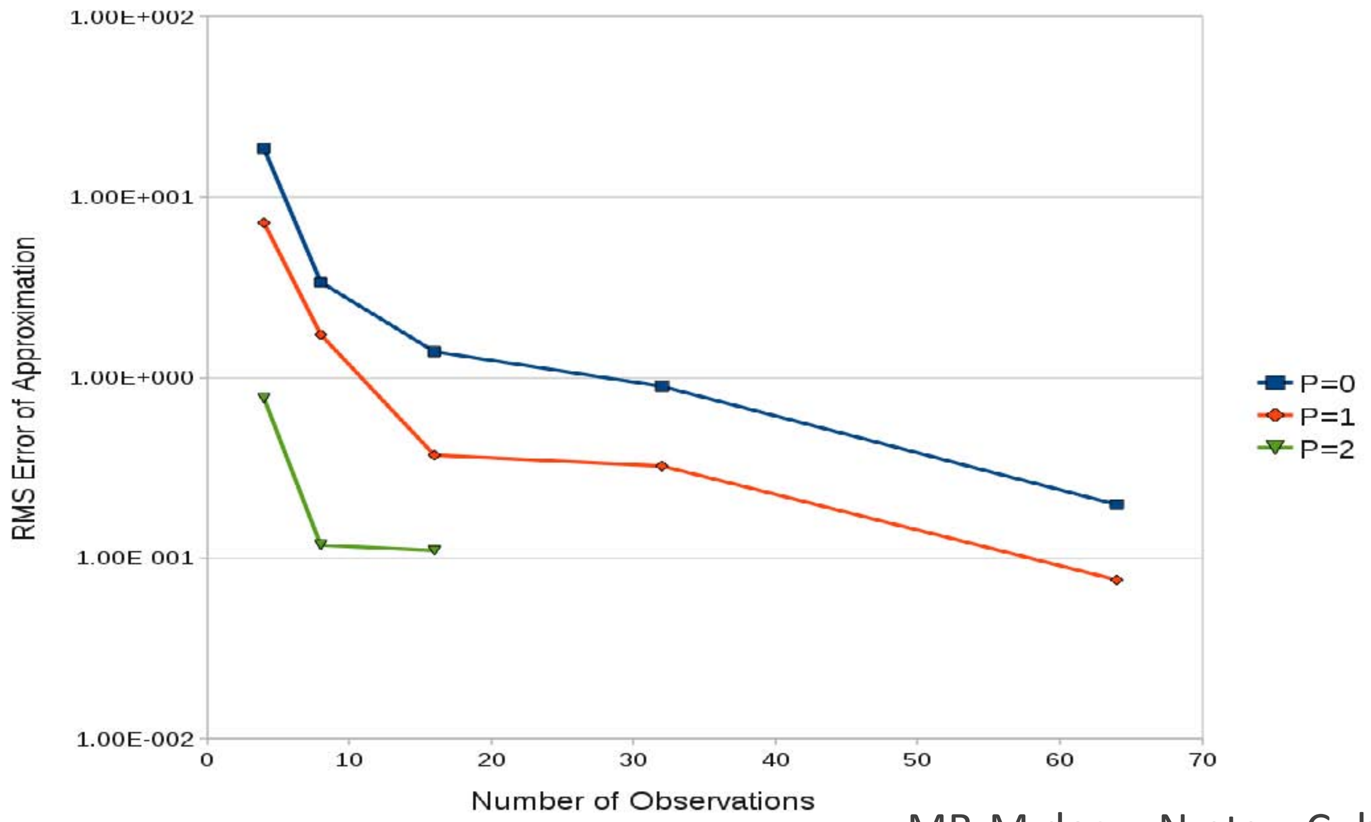


MS-2nd Deg – N pts – Cubic Spline 2

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

MB-0rd Deg – N pts – Cubic Spline 2

H3: Using a mean function versus ordinary kriging



MB-M deg – N pts – Cubic Spline 2

Conclusion H3: Modeling the mean may matter and it definitely does not hurt (it does not help in MS)

H4: Kriging gives a good approximation of the error

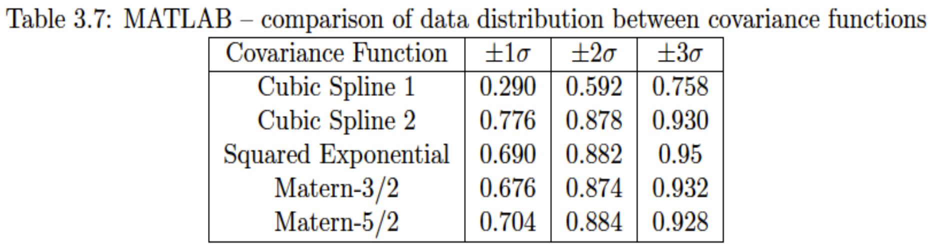


Table 3.7: MATLAB – comparison of data distribution between covariance functions

Covariance Function	$\pm1\sigma$	$\pm2\sigma$	$\pm3\sigma$
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

With Decorrelation

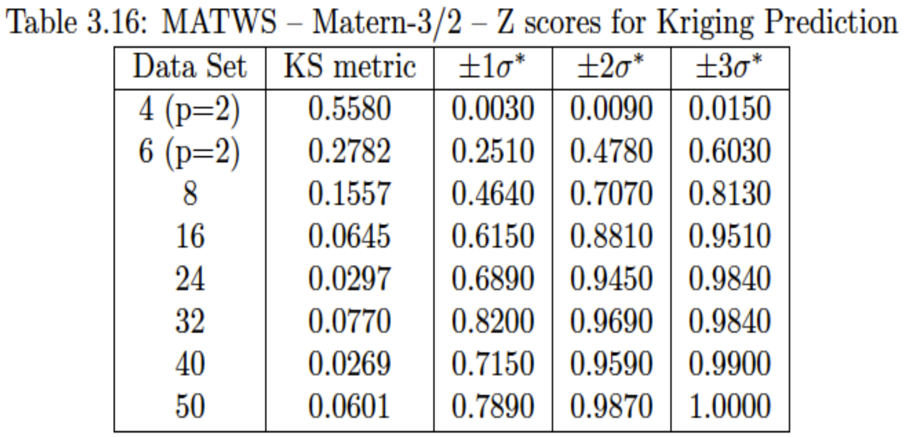


Table 3.16: MATWS – Matern-3/2 – Z scores for Kriging Prediction

Data Set	KS metric	$\pm1\sigma^*$	$\pm2\sigma^*$	$\pm3\sigma^*$
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

Without Decorrelation

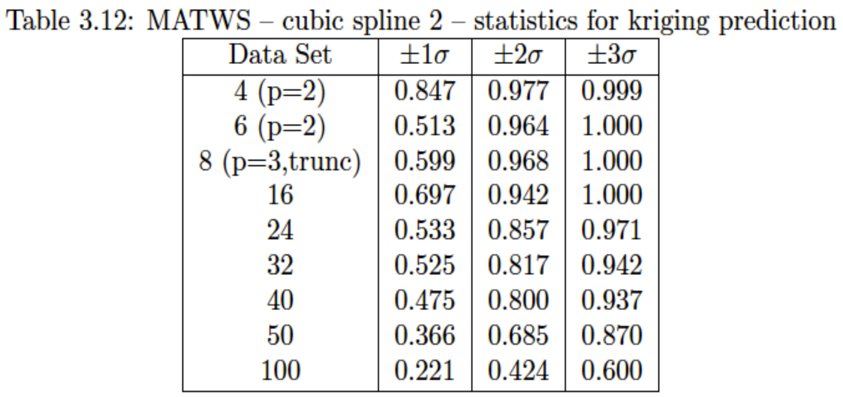


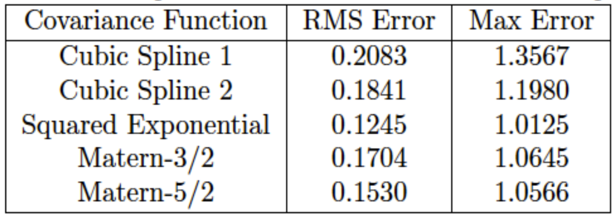
Table 3.12: MATWS – cubic spline 2 – statistics for kriging prediction

Data Set	$\pm1\sigma$	$\pm2\sigma$	$\pm3\sigma$
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. (Square Exponential: not PD at Machine Precision)

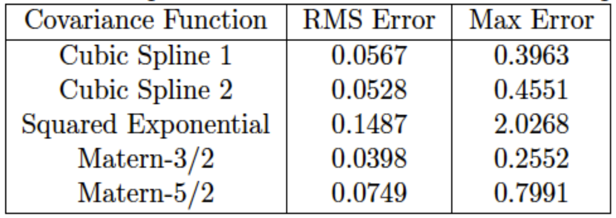
H5: The choice of covariance function matters

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

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 u, the error propagation is deterministic conditional on u

$$f(u_1), f(u_2), \ldots, f(u_N) \mid (u_1, u_2 \ldots, u_N) \sim \delta\left(f(u_1), f(u_2), \ldots, f(u_N)\right)$$

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

$$f(u_1), f(u_2), \ldots, f(u_N) \mid (u_1, u_2 \ldots, u_N) \sim N\left(m^c(U), K^c(U)\right)$$

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

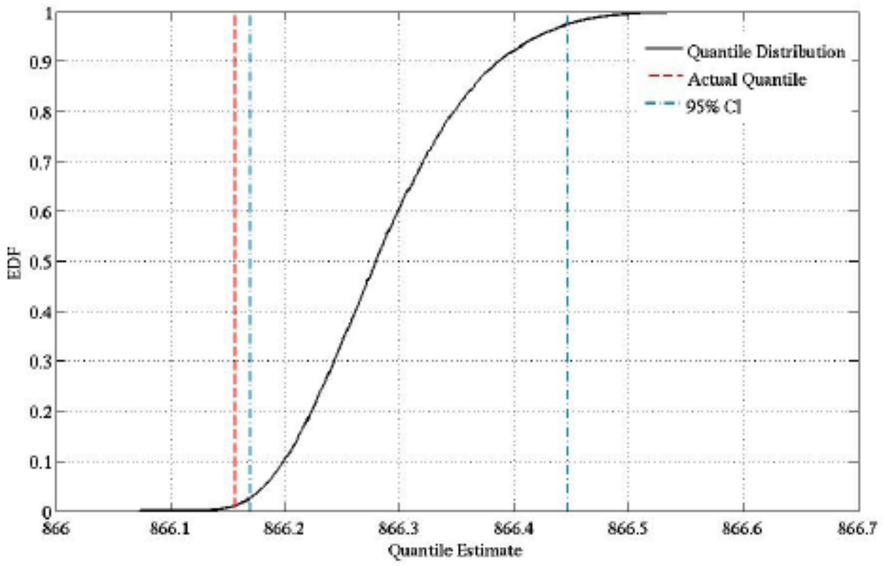
Actual Value = 866.16

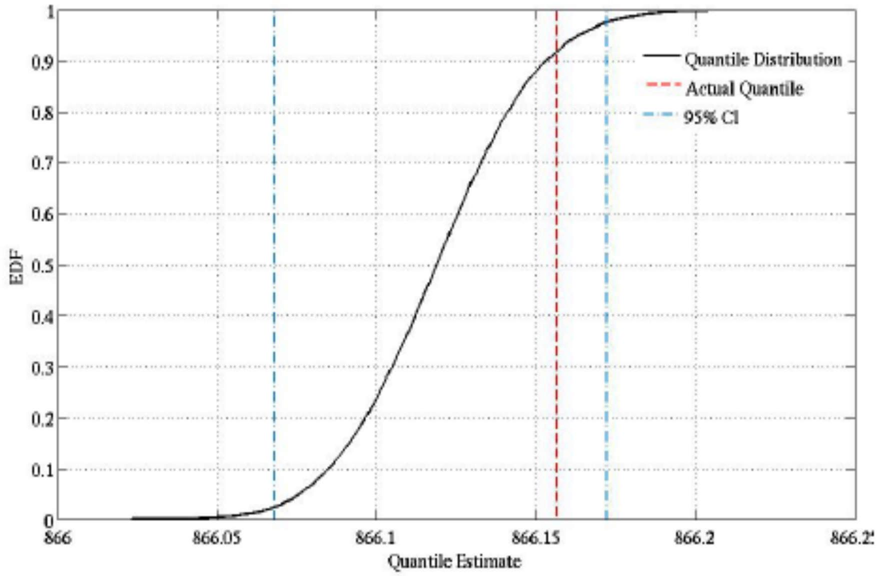
Estimating Quantiles Using Asymptotic tests

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

DATA	Training Points	Confidence(1-α)	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



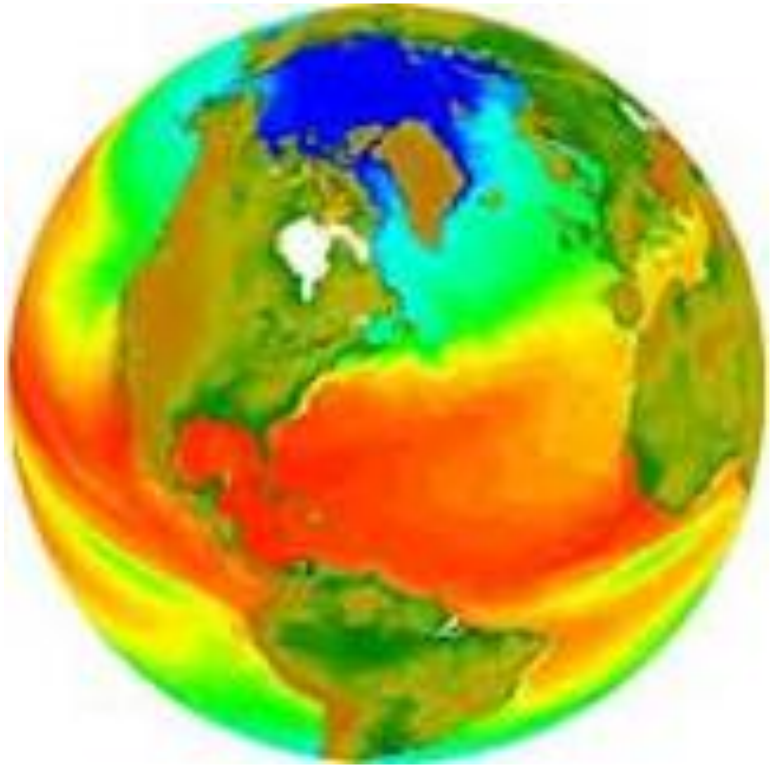


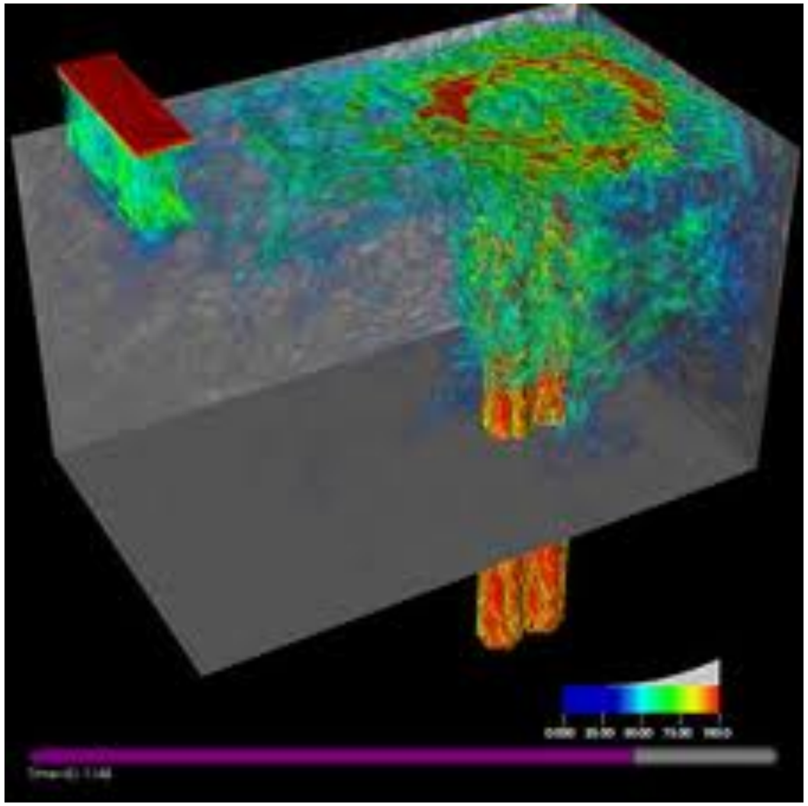
GPs with derivatives: Research Issues

- If I use GPs to approximate a deterministic response under what conditions can the posterior covariance be used as an error estimate?
- What theoretical considerations should guide the choice of the covariance kernel?
- Can the distribution of the parameter induce better choices of the covariance kernel?
- How do we deal with the multiple local minima we see in the likelihood?
- What are good optimal design formulations and their solutions?

4 SCALABLE GP ANALYSIS

Examples





Source: http://www.ccs.ornl.gov/

Tasks and challenges

- Sampling
- Maximum likelihood
- Interpolation/Kriging (solving linear system with K)
- Regression/Classification (solving linear systems with K)
- …

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

$$K = A^T A, \quad \xi \sim N\left(0, I\right), \quad y = M + A\xi \sim N(m, K)$$

- A lot of the basic tasks require matrix computations w.r.t. the covariance matrix K.
- But for 1B data points, you need 8*10^18 bytes to store = 8 EXABYTES, so cannot store K.
- How do you do compute log-det and A without storing the covariance matrix? And Hopefully in O(number data points) operaations?

Maximum Likelihood Estimation (MLE)

- A family of covariance functions parameterized by θ: ϕ(x; θ)
- Maximize the log-likelihood to estimate θ:

$$\max_{\theta} L(\theta) = \log\left\{(2\pi)^{-n/2}(\det K)^{-1/2}\exp(-y^T K^{-1} y/2)\right\}$$
$$= -\frac{1}{2}y^T K^{-1} y - \frac{1}{2}\log(\det K) - \frac{n}{2}\log 2\pi$$

- First order optimality: (also known as score equations)

$$\frac{1}{2}y^T K^{-1}(\partial_j K)K^{-1}y - \frac{1}{2}\mathrm{tr}\left[K^{-1}(\partial_j K)\right] = 0$$

Maximum Likelihood Estimation (MLE)

The log-det term poses a significant challenge for large-scale computations

$$\max_{\theta} \quad -\frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log(\det K) - \frac{n}{2} \log 2\pi$$

- Cholesky of K: Prohibitively expensive!
- log(det K) = tr(log K): Need some matrix function methods to handle the log
- No existing method to evaluate the log-det term in sufficient accuracy

Sample Average Approximation of Maximum Likelihood Estimation (MLE)

We consider approximately solving the first order optimality instead:

$$\frac{1}{2}y^T K^{-1}(\partial_j K)K^{-1}y - \frac{1}{2}\mathrm{tr}\left[K^{-1}(\partial_j K)\right]$$

$$\approx \frac{1}{2}y^T K^{-1}(\partial_j K)K^{-1}y - \frac{1}{2N}\sum_{i=1}^{N} u_i^T\left[K^{-1}(\partial_j K)\right]u_i = 0$$

- A randomized trace estimator tr(A) = E[u^TAu]
  - u has i.i.d. entries taking ±1 with equal probability
- As N tends to infinity, the solution approaches the true estimate
- The variance introduced in approximating the trace is comparable with the variance of the sample y
  - So the approximation does not lose too much accuracy

- Numerically, one must solve linear systems with O(N) right-hand sides.

Stochastic Approximation of Trace

- When entries of u are i.i.d. with mean zero and covariance I

$$\mathrm{tr}(A) = E_u\left[u^T A u\right]$$

- The estimator has a variance

$$\mathrm{var}\left\{u^T A u\right\} = \sum_i \left(E\left[u_i^4\right] - 1\right) A_{ii}^2 + \frac{1}{2} \sum_{i \neq j} (A_{ij} + A_{ji})^2$$

- If each entry of u takes ±1 with equal probability, the variance is the smallest

$$\mathrm{var}\left\{u^T A u\right\} = \frac{1}{2} \sum_{i \neq j} (A_{ij} + A_{ji})^2$$

Convergence of Stochastic Programming - SAA

- Let

$$\theta: \text{ truth}$$

$$\hat{\theta}: \text{ sol of } \frac{1}{2} y^T K^{-1}(\partial_j K) K^{-1} y - \frac{1}{2} \operatorname{tr}\left[K^{-1}(\partial_j K)\right] = 0$$

$$\hat{\theta}^N: \text{ sol of } F = \frac{1}{2} y^T K^{-1}(\partial_j K) K^{-1} y - \frac{1}{2N} \sum_{i=1}^{N} u_i^T \left[K^{-1}(\partial_j K)\right] u_i = 0$$

- First result:

$$[V^N]^{-1/2}(\hat{\theta}^N - \hat{\theta}) \xrightarrow{D} \text{standard normal}, \quad V^N = [J^N]^{-T} \Sigma^N [J^N]^{-1}$$

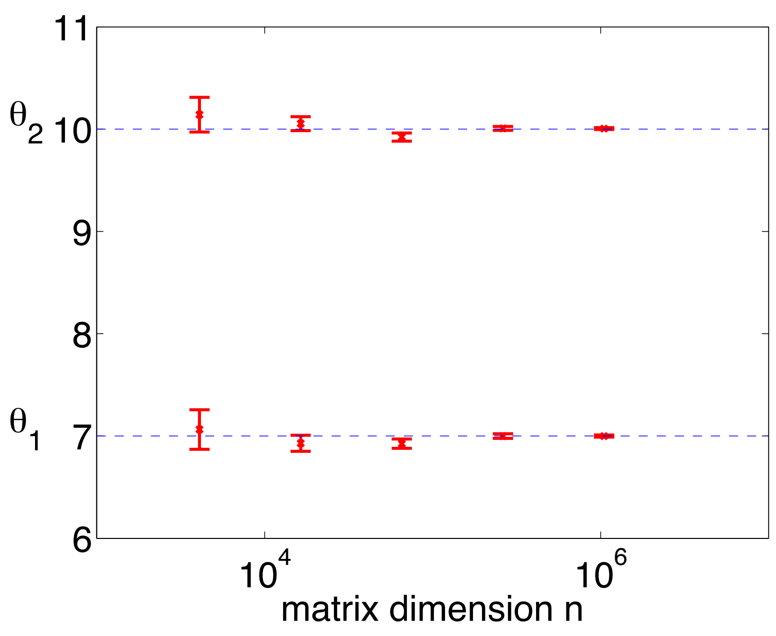
where

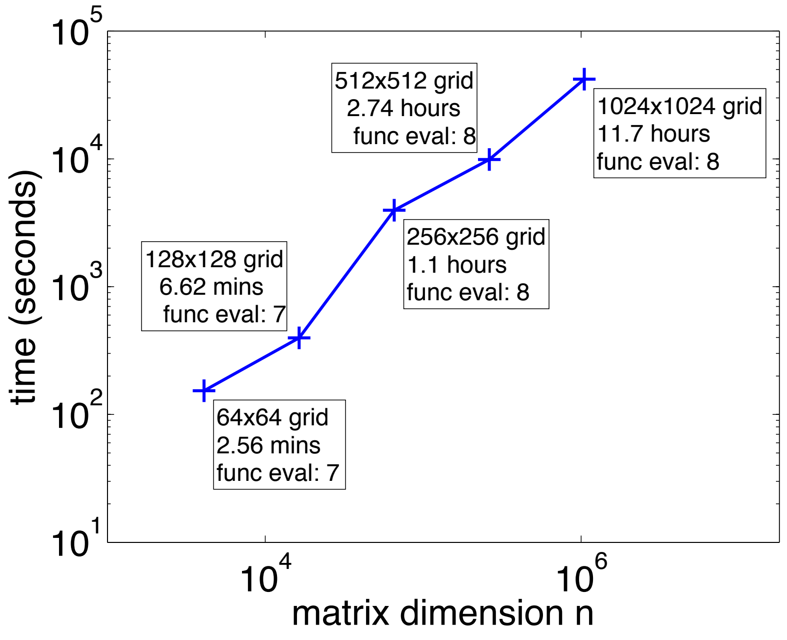
$$J^N = \nabla F(\hat{\theta}^N) \quad \text{and} \quad \Sigma^N = \operatorname{cov}\{F(\hat{\theta}^N)\}$$

- Note: Σ^N decreases in $O(N^{-1})$

Simulation: We scale

- Truth θ = [7, 10], Matern ν = 1.5





“Optimal” Convergence

- Let

$$\theta: \text{ truth}$$

$$\hat{\theta}: \text{ sol of } \frac{1}{2} y^T K^{-1}(\partial_j K) K^{-1} y - \frac{1}{2}\mathrm{tr}\left[K^{-1}(\partial_j K)\right] = 0$$

$$\hat{\theta}^N: \text{ sol of } F = \frac{1}{2} y^T K^{-1}(\partial_j K) K^{-1} y - \frac{1}{2N}\sum_{i=1}^{N} u_i^T \left[K^{-1}(\partial_j K)\right] u_i = 0$$

- Second result:

$$C^{-1/2}(\hat{\theta}^N - \theta) \xrightarrow{D} \text{standard normal}, \quad \mathrm{C} = A^{-T} B A^{-1}$$

where

$$-A = I, \text{ Fisher matrix} \quad \text{and} \quad B = I + \frac{1}{4N} J$$

- Note: J has a bound $J \le I \cdot \dfrac{[\mathrm{cond}(K) + 1]^2}{\mathrm{cond}(K)}$, so C converges to I^{-1} in $O(N^{-1})$ if condition number of K is bounded.

LINEAR ALGEBRA CHALLENGES: PRECONDITIONING AND MATRIX VECTOR MULTIPLICATIONS

We reduced max likelihood calculations to solving linear systems with K.

We next focus on the linear algebra:

- Preconditioning K
- Matrix-vector multiplication with K
- Solving linear system w.r.t. K with multiple right-hand sides

Covariance Model

- Matern covariance function

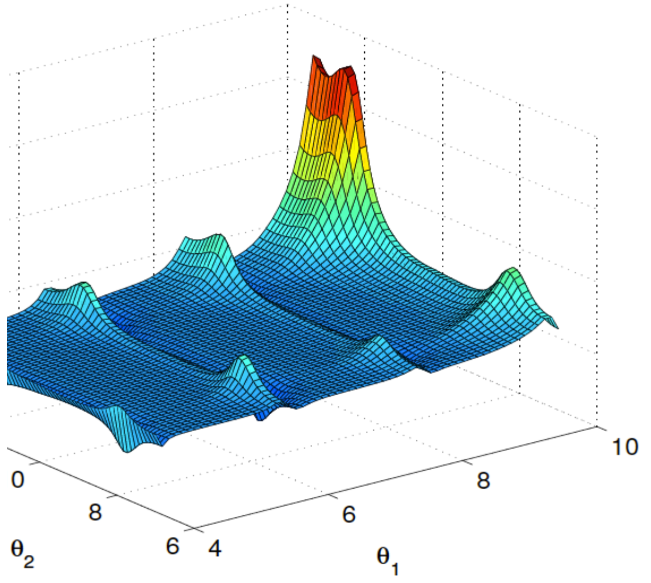
$$\phi(x) = \frac{1}{2^{\nu-1}\Gamma(\nu)}\left(\sqrt{2\nu}r\right)^{\nu} \mathrm{K}_{\nu}\left(\sqrt{2\nu}r\right) \quad \text{where} \quad r = \sqrt{\sum_{j=1}^{d}\frac{x_j^2}{\theta_j^2}}$$

  - ν: Example values 0.5, 1, 1.5, 2
  - θ: Scale parameters to estimate
  - K_ν is the modified Bessel function of the second kind of order ν
- Commonly used in spatial/temporal data modeling.
- The parameter ν is used to model the data with a certain level of smoothness.
- When ν -> ∞, the kernel is the Gaussian kernel.
- Spectral density

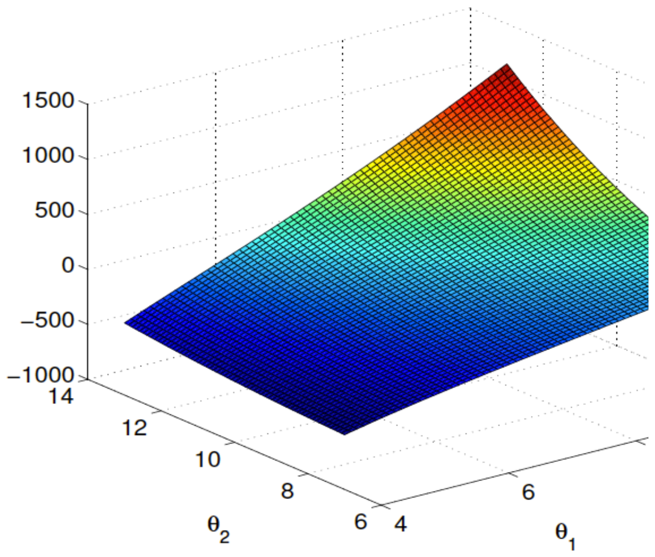
$$f(\omega) \propto \left(2\nu + \rho^2\right)^{-(\nu+d/2)} \quad \text{where} \quad \rho = \sqrt{\sum_{j=1}^{d}(\theta_j\omega_j)^2}$$

Why the Matern Kernel?

- In machine learning, people tend to use the square exponential kernel a lot.
- This assumes that all realizations are infinitely smooth, a fact rarely supported by data, especially high resolution data.
- The Matern Kernel allows one to adjust smoothness.
- The resulting covariance matrix is dense, compared to compact Kernels, but the likelihood surface is much smoother.



(a) Compact kernel.



(b) Matern kernel.

Condition Number

- K is increasingly ill-conditioned.
- If the grid is in a fixed, finite domain $\subset \mathrm{R}^d$, then cond(K) = O($n^{2v/d+1}$)

- On regular grid, K is (multi-level) Toeplitz, hence a circulant preconditioner applies

$$\begin{bmatrix} t_0 & t_{-1} & \cdots & t_{-n+2} & t_{-n+1} \\ t_1 & t_0 & t_{-1} & & t_{-n+2} \\ \vdots & t_1 & t_0 & \ddots & \vdots \\ t_{n-2} & & \ddots & \ddots & t_{-1} \\ t_{n-1} & t_{n-2} & \cdots & t_1 & t_0 \end{bmatrix} \longrightarrow \begin{bmatrix} c_0 & c_{n-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{n-1} & & c_2 \\ \vdots & c_1 & c_0 & \ddots & \vdots \\ c_{n-2} & & \ddots & \ddots & c_{n-1} \\ c_{n-1} & c_{n-2} & \cdots & c_1 & c_0 \end{bmatrix}$$

- More can be done by considering filtering

Condition Number

- Filtering (1D): if f(w)w^2 bounded away from 0 and ∞ as w -> ∞
- Let $0 \le x_0 \le x_1 \le \dots \le x_n \le T$. $d_j = x_j - x_{j-1}$.

$$Y_j^{(1)} = [Z(x_j) - Z(x_{j-1})] / \sqrt{d_j}, \qquad K^{(1)}(j,l) = \operatorname{cov}\left\{Y_j^{(1)}, Y_l^{(1)}\right\}$$

- Then $K^{(1)}$ has a bounded condition number independent of n

- Filtering (1D): if f(w)w^4 bounded away from 0 and ∞ as w -> ∞

$$Y_j^{(2)} = \frac{Z(x_{j+1}) - Z(x_j)}{2d_{j+1}\sqrt{d_{j+1} + d_j}} - \frac{Z(x_j) - Z(x_{j-1})}{2d_j\sqrt{d_{j+1} + d_j}}, \qquad K^{(2)}(j,l) = \operatorname{cov}\left\{Y_j^{(2)}, Y_l^{(2)}\right\}$$

- Then $K^{(2)}$ has a bounded condition number independent of n

Condition Number

- Filtering (high dimension, regular grid): if f(w) is asymptotically (1+|w|)$^{-4\tau}$

$$\Delta Z(x_j) = \sum_{p=1}^{d} Z(x_j - \delta e_p) - 2Z(x_j) + Z(x_j + \delta e_p)$$

$$K^{[\tau]}(j,l) = \operatorname{cov}\left\{\Delta^{[\tau]} Z(x_j), \Delta^{[\tau]} Z(x_l)\right\}$$

- Then K$^{[\tau]}$ has a bounded condition number independent of n

- Use the filter as a preconditioner

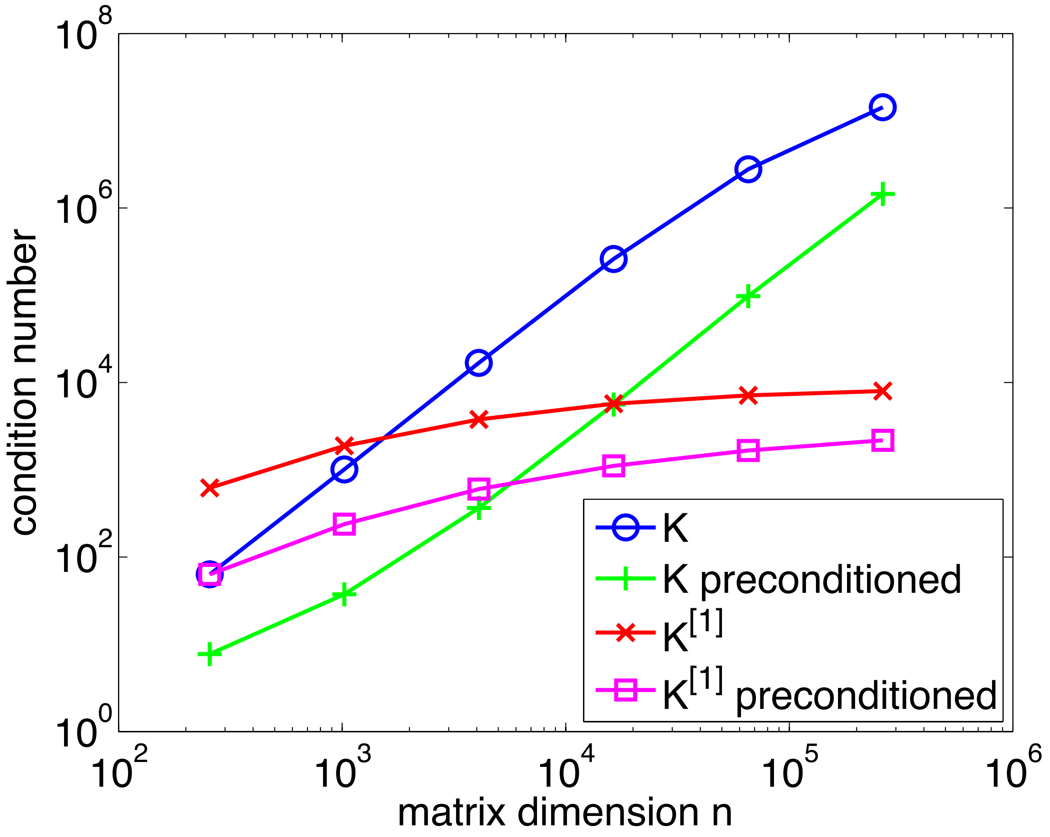
$$K^{[\tau]} = \left[L^{[\tau]}\right]\cdots\left[L^{[2]}\right]\left[L^{[1]}\right]K\left[L^{[1]}\right]^T\left[L^{[2]}\right]^T\cdots\left[L^{[\tau]}\right]^T$$

In 2D, L is the 5-point stencil matrix with rows w.r.t. the grid boundary removed.

- Similarly for the filters in the preceding slide

Condition Number

- Effect of filtering ($K^{[\tau]}$ can be further preconditioned by circulant preconditioner)



Block CG

- Preconditioned Conjugate Gradient (M is preconditioner)

$$Ax = b$$

$$x_{j+1} = x_j + \alpha_j p_j$$
$$r_{j+1} = r_j - \alpha_j A p_j$$
$$p_{j+1} = M r_{j+1} + \beta_j p_j$$

where

$$\alpha_j = r_j^T M r_j / p_j^T A p_j$$
$$\beta_j = r_{j+1}^T M r_{j+1} / r_j^T M r_j$$

$$AX = B \quad \text{(block version)}$$

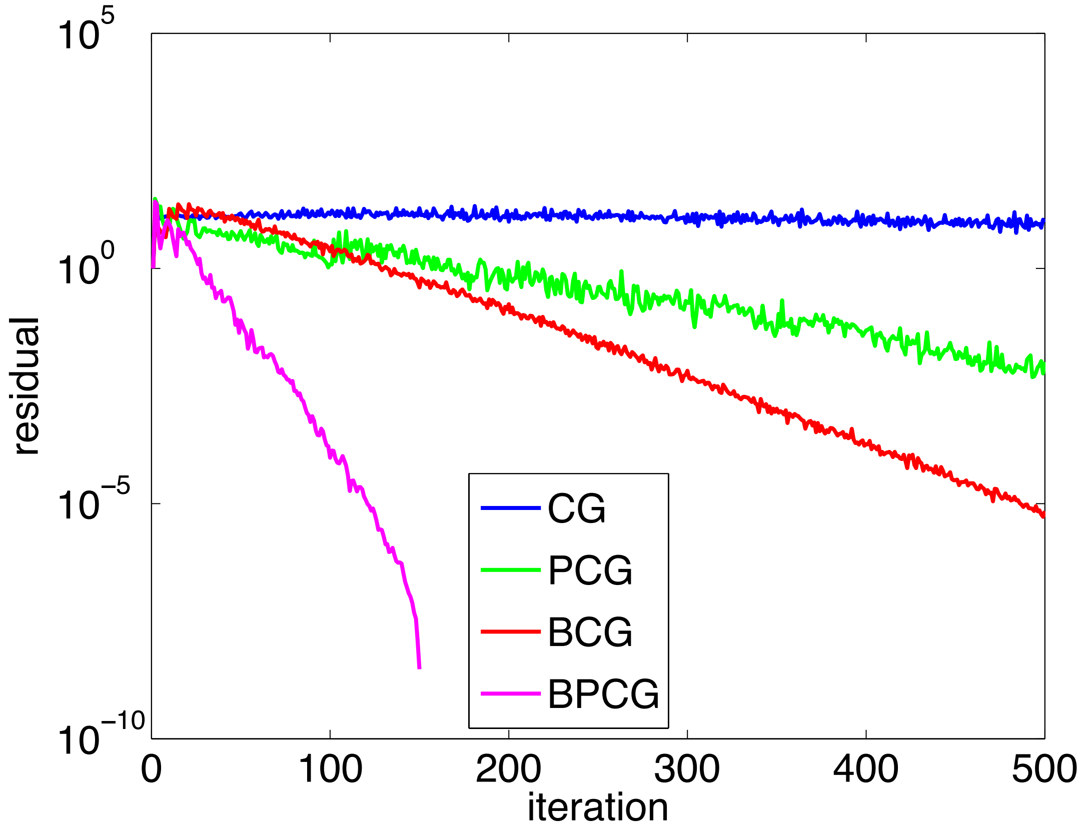
$$X_{j+1} = X_j + P_j \alpha_j$$
$$R_{j+1} = R_j - A P_j \alpha_j$$
$$P_{j+1} = (M R_{j+1} + P_j \beta_j)\gamma_{j+1}$$

where

$$\alpha_j = (P_j^T A P_j)^{-1} \gamma_j^T (R_j^T M R_j)$$
$$\beta_j = \gamma_j^{-1} (R_j^T M R_j)^{-1} (R_{j+1}^T M R_{j+1})$$

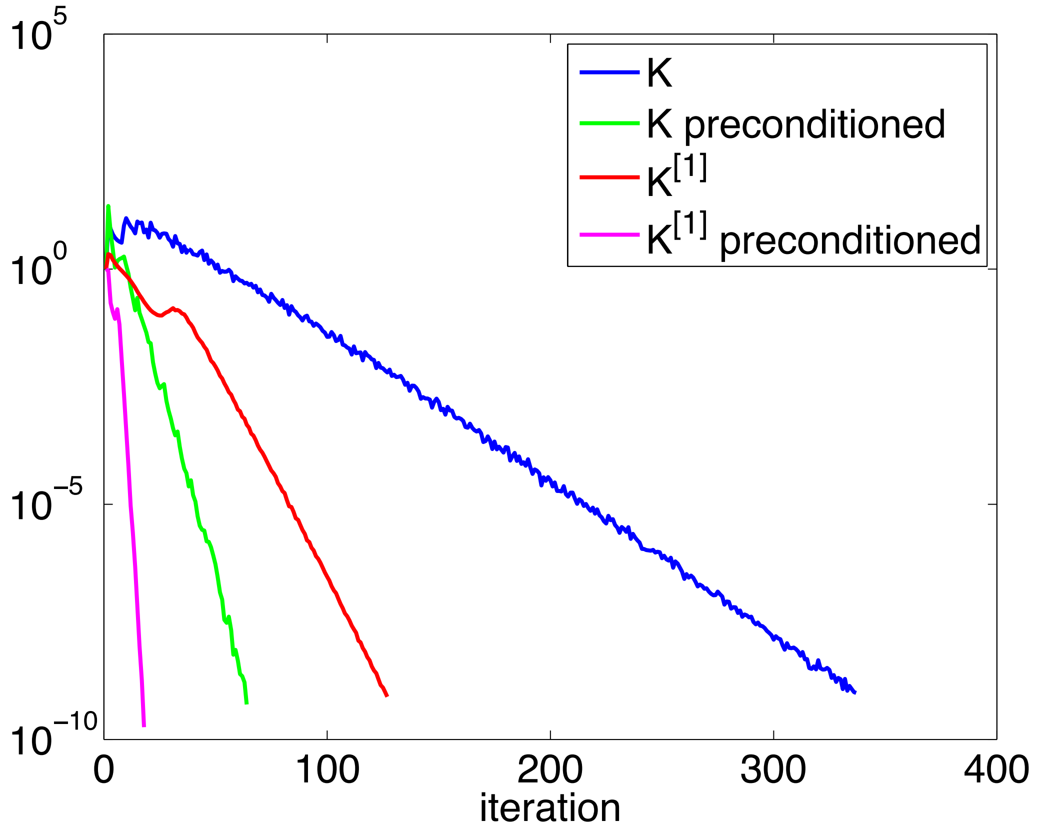
Block CG

- CG, block CG, and the preconditioned versions using circulant preconditioner



Experimental Results

- Combined effect of circulant preconditioning and filtering



MATRIX-VECTOR MULTIPLICATIONS

Matrix-Vector Multiplication

Matrix-vector multiplication s = Kq

- A straightforward implementation requires $O(n^2)$ calculations because K is full.
- We use a tree code to perform the calculation in O(n log n) time.

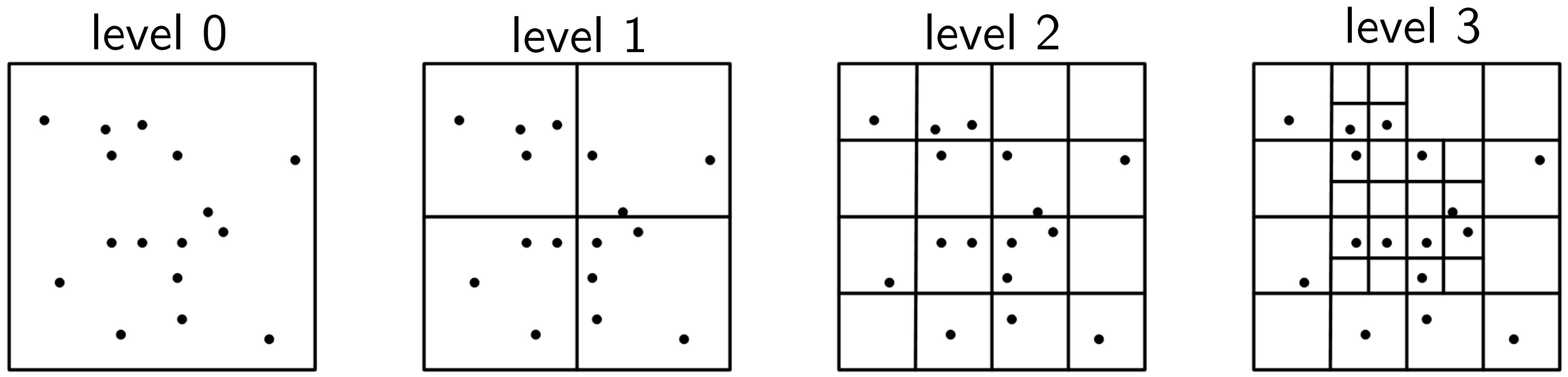
General idea:

- separate the set of points into clusters.
- For each entry of s, separate the summation into cluster sums:

$$s_i = \sum_{j=1}^{n} q_j \phi(x_i - x_j) = \sum_{C} s(x_i, C), \qquad \text{where } s(x_i, C) = \sum_{y_j \in C} q_j \phi(x_i - y_j)$$

- If x_i is close to C, $s(x_i,C)$ is computed in a brute-force manner
- The gain by using tree code is when x_i is far away from C, $s(x_i,C)$ can be approximated by expansion of ϕ, such that calculations are reduced.

Matrix-Vector Multiplication



Tree code:

- Recursively partition the point set
- Start from the top of hierarchy
- If x_i is not sufficiently far away the cluster center, visit the four child clusters
- If x_i is sufficiently far away from a cluster, use expansion to compute $s(x_i,C)$
- If bottom of hierarchy is reached, compute $s(x_i,C)$ in a brute-force manner

Matrix-Vector Multiplication

Taylor expansion around center of C, y_C:

$$s(x_i, C) = \sum_{y_j \in C} q_j \phi(x_i - y_j)$$

$$\approx \sum_{y_j \in C} q_j \sum_{\|k\|=0}^{p} \frac{1}{k!} D_y^k \phi(x_i, y_C)(y_j - y_C)^k$$

$$= \boxed{\sum_{\|k\|=0}^{p} \frac{1}{k!} D_y^k \phi(x_i, y_C)} \boxed{\sum_{y_j \in C} q_j (y_j - y_C)^k}$$

Coefficients, need a smart way to evaluate

Moments, for all i compute only once

Matern kernel: A recurrence relation can be built to compute the coefficients for all k

Matrix-Vector Multiplication

$\psi_1 = \exp(-\sqrt{3}r)/r$, $\psi_2 = \exp(-\sqrt{3}r)$, $\psi_3 = \sqrt{3}r\exp(-\sqrt{3}r)$. Let $b_{\boldsymbol{k}}$, $c_{\boldsymbol{k}}$, $d_{\boldsymbol{k}}$ be their Taylor coefficients, respectively:

$$b_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!} D_{\boldsymbol{y}}^k \psi_1(\boldsymbol{x}_i, \boldsymbol{y}_c), \quad c_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!} D_{\boldsymbol{y}}^k \psi_2(\boldsymbol{x}_i, \boldsymbol{y}_c), \quad d_{\boldsymbol{k}} = \frac{1}{\boldsymbol{k}!} D_{\boldsymbol{y}}^k \psi_3(\boldsymbol{x}_i, \boldsymbol{y}_c).$$

We have

$$a_{\boldsymbol{k}} = c_{\boldsymbol{k}} + d_{\boldsymbol{k}}, \tag{4.4}$$

and the following recurrence formulas:

$$\begin{aligned}
b_{\boldsymbol{k}} &= \frac{1}{L_1L_2||\boldsymbol{k}||r^2}\left((2||\boldsymbol{k}||-1)\sum_{i=1}^{2} L_{j(i)}(x_i-y_i)b_{\boldsymbol{k}-\boldsymbol{e}_i} - (||\boldsymbol{k}||-1)\sum_{i=1}^{2} L_{j(i)} b_{\boldsymbol{k}-\boldsymbol{2e}_i}\right) \\
&\quad + \frac{\sqrt{3}}{L_1L_2||\boldsymbol{k}||r^2}\left(\sum_{i=1}^{2} L_{j(i)}((x_i-y_i)c_{\boldsymbol{k}-\boldsymbol{e_i}} - c_{\boldsymbol{k}-\boldsymbol{2e_i}})\right), && (4.5) \\
c_{\boldsymbol{k}} &= \frac{\sqrt{3}}{||\boldsymbol{k}||}\left(\sum_{i=1}^{2}\frac{1}{L_i}((x_i-y_i)b_{\boldsymbol{k}-\boldsymbol{e_i}} - b_{\boldsymbol{k}-\boldsymbol{2e_i}})\right), && (4.6) \\
d_{\boldsymbol{k}} &= \frac{\sqrt{3}}{||\boldsymbol{k}||}\left(\sum_{i=1}^{2}\frac{x_i-y_i}{L_i}(\sqrt{3}c_{\boldsymbol{k}-\boldsymbol{e}_i} - b_{\boldsymbol{k}-\boldsymbol{e}_i}) - \sum_{i=1}^{2}\frac{1}{L_i}(\sqrt{3}c_{\boldsymbol{k}-\boldsymbol{2e}_i} - b_{\boldsymbol{k}-\boldsymbol{2e}_i})\right), && (4.7)
\end{aligned}$$

Linear Algebra Summary

- When the block CG solver is chosen, two additional issues are:
  - How to do matrix-vector multiplications faster than $O(n^2)$?
  - How to precondition K?

- Regular grid case (K is multi-level Toeplitz):
  - Mat-vec: Fast Fourier Transform
  - Perform filtering to control the condition number
  - Further use a multi-level circulant preconditioner to precondition K

- Scattered points case (on going, ideas):
  - Mat-vec: Fast summation methods (e.g., tree code)
  - Filtering: Construct a discrete Laplace operator for scattered points

Sampling

- Obtaining a sample y:
  - Generate a vector x with i.i.d. standard normal entries
  - Compute some G such that $GG^T = K$
  - Compute y = Gx

- Traditional methods use the Cholesky factor K as G.
- Cholesky is prohibitively expensive in memory and in space for large K.

- We use matrix function techniques to directly compute $K^{1/2}x$.

Function of Matrix Times Vector f(A)b

General techniques

- Krylov subspace methods.
  - Low rank approximation of A by $Q_k H_k Q_k^*$. Then $f(A)b \approx Q_k f(H_k) Q_k^* b$.
  - (+) Extensively studied. (-) Need to store all the vectors; loss of orthogonality.
- Polynomial approximation methods.
  - Approximate f by a polynomial p. Then $f(A)b \approx p(A)b$.
  - (+) Many polynomial basis to choose. (-) Polynomial expansion may need quadrature.
- Rational approximation.
  - Approximate f by rational polynomial p/q. Then $f(A)b \approx q(A)^{-1}p(A)b$.
  - (+) Low degree approximation suffices. (-) Need to solve shifted linear systems.
- Contour integral.
  - Write $f(A) = (2\pi i)^{-1}$ integral $f(z)\,(zI-A)^{-1}\,dz$. Then compute f(A)b by quadrature.
  - (+) and (-) similar to rational approximation. (-) Need design good quadrature points.

Function of Matrix Times Vector f(A)b

We consider approximating f by a least squares polynomial

$$f(t) \approx \varphi_k(t) := \sum_{j=0}^{k} \langle f, P_j \rangle P_j(t)$$

where the orthnormal basis $\{P_j\}$ can be generated by a three term recursion

$$\beta_{j+1} P_{j+1} = tP_j - \alpha_j P_j - \beta_j P_{j-1}, \qquad \alpha_j = \langle tP_j, P_j \rangle, \quad \beta_{j+1} = \text{normalization}$$

- Vectors $v_j = P_j(A)b$ are updated using same recursion until sufficiently close to f(A)b
- Matrix free: Only action of A on b is needed to compute v_j
- Computing α, β may need quadrature. We design a method to avoid this.

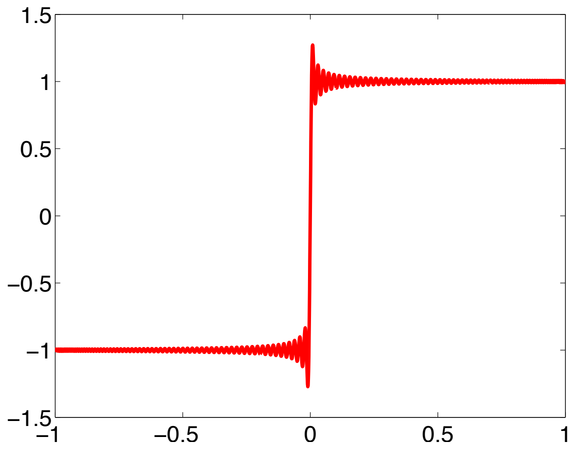
Idea: Use a spline s to approximate f first.

$$f(t) \approx s(t) \approx \varphi_k(t)$$

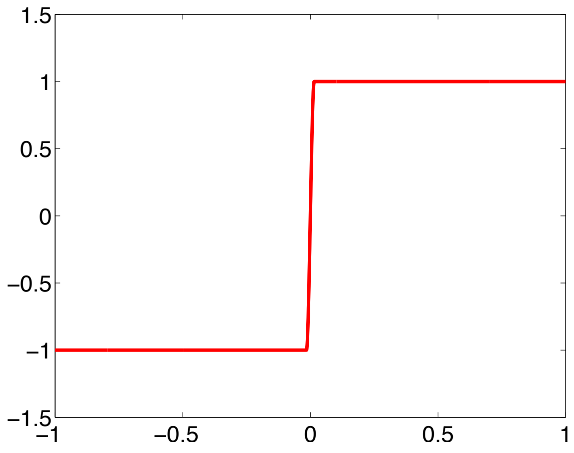
Function of Matrix Times Vector f(A)b

Benefits of using a spline:

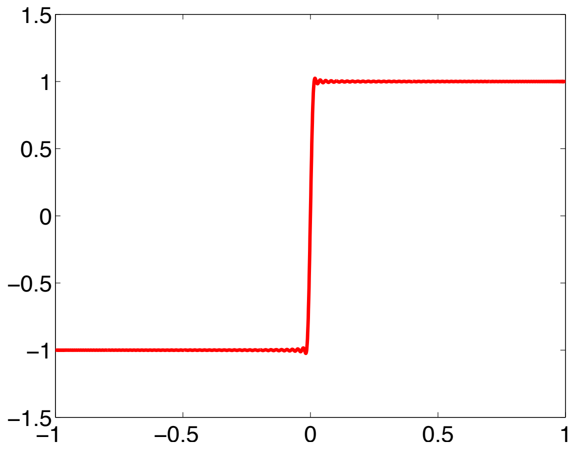
- Quadratures can be computed exactly in each subinterval.
- Alleviate Gibbs phenomenon



(a)



(b)



(c)

(a) polynomial approximation, (b) spline, (c) polynomial approximation to spline

Function of Matrix Times Vector f(A)b

Convergence

$$\|f(A)b - \varphi_k(A)b\|_2 \le \|f - \varphi_k\|_\infty \|b\|_2$$

$$\|f - s\|_\infty = O\left(\|f^{(4)}\|_\infty \max_j (t_{j+1} - t_j)^4\right)$$

$$\|s - \varphi_k\|_\infty = O\left(\frac{C_r (b-a)^\rho}{k^{r+\rho-0.5}}\right)$$

where

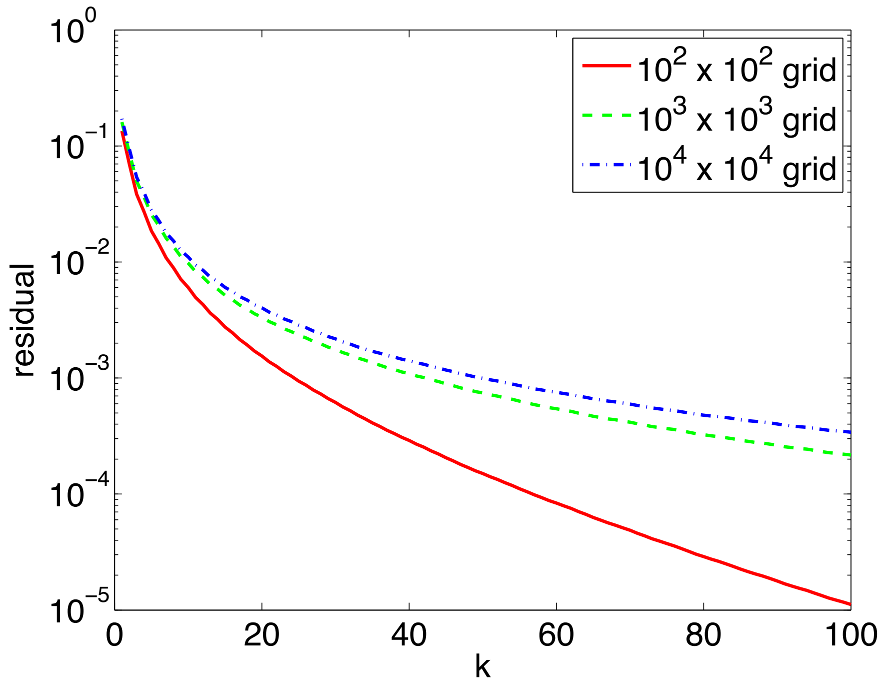
- The r-th derivative of spline s is ρ-Lipschitz with constant C_r
- a and b are the two end points of the spline (encapsulating spectrum of A)
- t_j are spline knots

Sampling error:

$$\|\mathrm{cov}\{K^{1/2}x - \varphi_k(K)x\}\|_2 \le \|f - \varphi_k\|_\infty^2$$

Function of Matrix Times Vector f(A)b

- Numerical results: Standard 2D Laplacian. $f(t) = t^{1/2}$



Function of Matrix Times Vector f(A)b

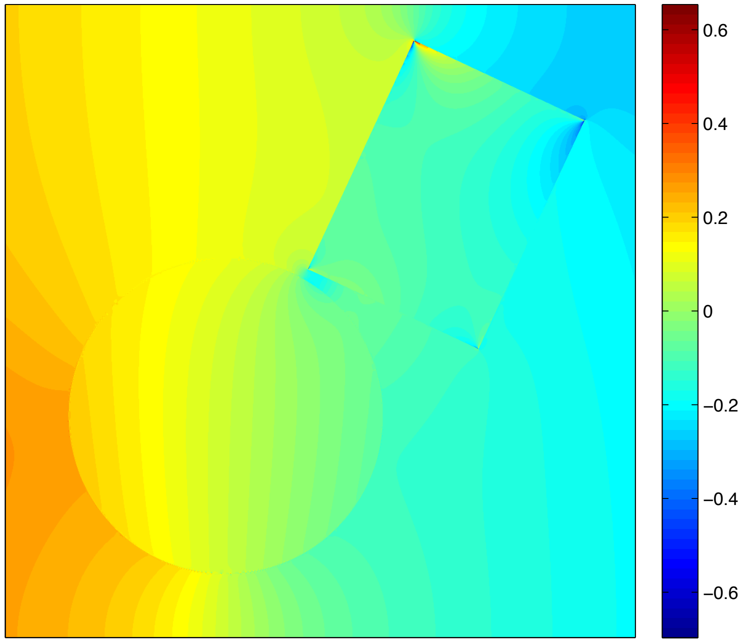
- Numerical results: Wieland Compact Kernel:

Grid	Grid Size	α	β	κ	k	Iter. Diff.	$\max\lvert\phi_{k+1} - f\rvert$
Regular	$10^3 \times 10^3$	6.5	3	35.14	48	9.5703×10^{-11}	1.2106×10^{-9}
Regular	$10^3 \times 10^3$	12.5	3	244.47	122	7.9481×10^{-11}	1.9733×10^{-9}
Regular	$10^3 \times 10^3$	6.5	5	13.14	32	5.6690×10^{-11}	4.2998×10^{-10}
Regular	$10^3 \times 10^3$	12.5	5	88.23	79	8.3043×10^{-11}	1.2952×10^{-9}
Deformed	$10^3 \times 10^3$	6.5	3	15.96	34	4.5546×10^{-11}	4.0252×10^{-10}
Deformed	$10^3 \times 10^3$	12.5	3	107.36	87	8.7497×10^{-11}	1.2524×10^{-9}
Deformed	$10^3 \times 10^3$	6.5	5	6.30	22	5.3848×10^{-11}	3.2586×10^{-10}
Deformed	$10^3 \times 10^3$	12.5	5	38.73	54	6.4973×10^{-11}	9.4807×10^{-10}

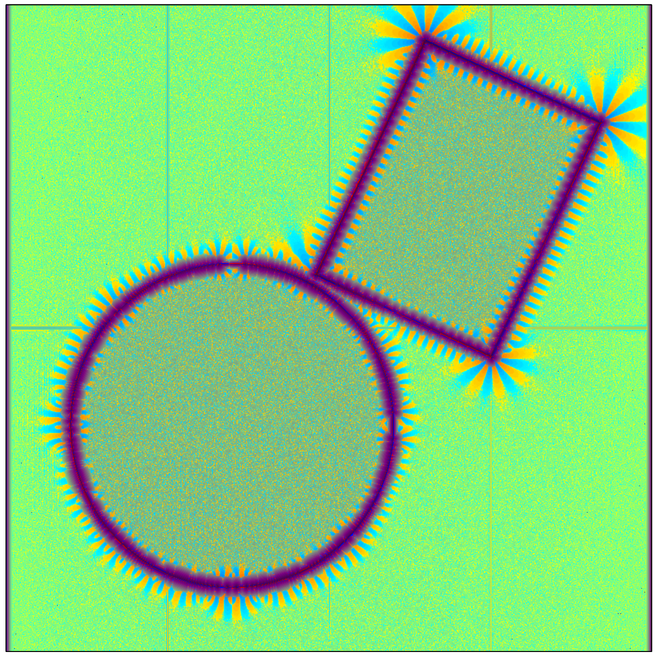
Residual: $$\max_{t \in \Lambda(A)} \left|\phi_{k+1}(t) - s(t)\right|,$$

Experimental Results

Stokes Flow



(a) Pressure field



(b) Filtered pressure field in log-scale

Stokes Flow

Fitted a power-law model

$$\phi(x;\alpha,C) = \Gamma(-\alpha/2)\cdot C\|x\|^{\alpha}$$

Data	Circle	Rectangle	Boundary	Background
# Points	1.5e+5	7.9e+4	3.1e+4	4.7e+5
Fitted α	0.1819	0.3051	0.7768	1.5945
Fitted C	722.54	803.07	3309.4	626180.0
eig(Fisher^{-1})$^{1/2}$	5.04e-4	9.80e-4	2.50e-3	8.11e-4
	1.31e+1	1.86e+1	1.26e+2	5.44e+3

Conclusion GP

- State-of-the-art methods use Cholesky to do sampling and solve ML.
  - Can probably handle data size up to n = $O(10^4)$ or $O(10^5)$.
- We propose a framework to overcome the Cholesky barrier.
  - Use a matrix-free method to do sampling.
  - Reformulate maximum likelihood using stochastic approximation.
  - Use iterative solver to solve linear systems.
  - Use a filtering technique to reduce the condition number.
- On going work
  - Investigating the scaling of parallel FFT for n = $O(10^6)$ and larger computations.
  - For scattered points, investigating a discrete Laplace operator for filtering.
  - Implementing a fast summation method to do mat-vec.

Conclusions Gradient Enhanced UQ

- 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 that regression, more efficient than kriging.
- Future:
  - Larger number of parameters
  - Approximated the gradients of very large scale codes.]