Polynomial Regression with Derivative Information for Uncertainty Quantification of Nuclear Modeling Simulations.

Mihai Anitescu, Oleg Roderick,
Jean Utke, Paul Hovland
(MCS, Argonne)
Thomas Fanning
(NE, Argonne)

Also acknowledging the work of Mihai Alexe (Virginia Tech) and Yiou Li (IIT)

May 2010

(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.
- Error Model: Use data to fix portions of f that are incorrect or unresolved (I have seen this in climate, not so much in NE)

- ❑ 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.
- ❑ 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).

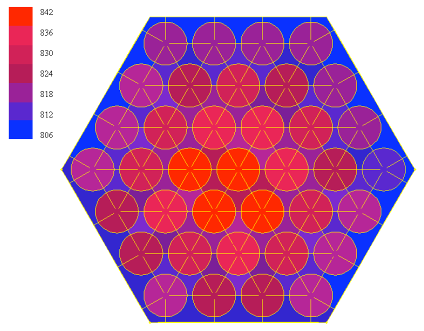
Faster Uncertainty Propagation by Using Derivative Information?

- 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. How?
- We believe the answer is yes.

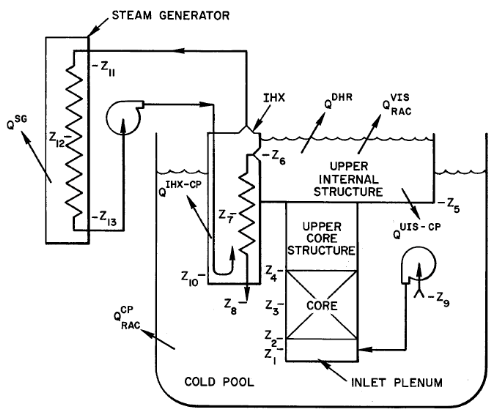
Argonne NATIONAL LABORATORY

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.
Available experimental data is parameterized by 12-38 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.



U.S. DEPARTMENT OF ENERGY

Representing Uncertainty

- Experimental data is scarce. Often, only a few measurements on range, and no record of statistical distribution of uncertainty-induced error are available. We created our own data model (see discussion session)

- We use 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 $T=(T_1,T_2,\ldots,T_n)$ $R=(R_1,R_2,\ldots,R_N)$

  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 $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_m)$

- **Note:** in our approach, the effect of uncertainty is parameterized by stochastic quantities, not randomized with some a priori distribution. The quantifiers are physical, not statistical.
- 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 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

- construct a regression matrix. Each row consists of either the values of the basis polynomials, or the values of derivatives of basis polynomials, at a point in the training set.

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

---

* but how to make the best choice? This is a topic of current investigation

** complete gradient information can be obtained, with limited computational overhead, for a model of any complexity.

Polynomial Regression with Derivatives, PRD

- PRD procedure, regression equations:

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

- Note: the only interaction with the computationally expensive model is on the right side!

- The polynomial regression approach without derivative information would require *(n+1)* times more rows. The overall computational savings depend on how cheaply the derivatives can be computed

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?)

  At the same time, the basis can be truncated to fit a given computational budget on the evaluations of the model to form a training set.

- In practice, 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.

PRD, computation of derivatives

- There is a theoretical limit of 500% on computational overhead required to compute derivatives: the use of derivatives is preferred to additional sampling when the dimension of uncertainty space exceeds 5.

  In practice, the overhead is 100-200%.

- It is possible to design the model with capability to output it own derivatives: the code can be augmented with partial derivatives of each elementary procedure, the gradient is then assembled by chain rule.

  In effect, together with evaluation of the model $J = J(T)\colon \quad F(T, R(T,\alpha)) = 0$
  the equations $(\frac{\partial F}{\partial T} + \frac{\partial F}{\partial R}\cdot\frac{\partial R}{\partial T})\cdot\frac{dT}{d\alpha} + \frac{\partial F}{\partial R}\cdot\frac{\partial R}{\partial \alpha} = 0$ are also solved, for $\frac{dT}{d\alpha}\colon \quad \frac{dJ}{d\alpha} = \frac{dJ}{dT}\cdot\frac{dT}{d\alpha}$

- 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.

  Model re-design is not required!

  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.

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.

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:
- Some features are unsupported by AD since they are not standard (machine-dependent code sections, nonstandard intrinsics such as LOC)
- Some features make the resulting AD adjoint code inefficient but are supported (Equivalence).

Applying AD to code with Fortran legacy components (ctd)

- 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, 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 temperatures with respect to the radial core expansion coefficient ,obtained by different AD tools, and compared with the Finite Differences approximation, FD.

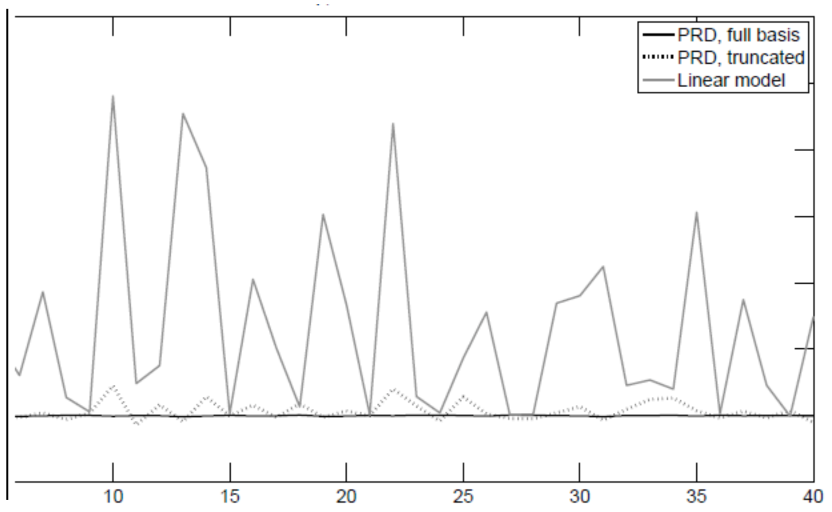
  All results agree with FD within 0.01% (and almost perfectly with each other).

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

PRD UQ, tests on subject models

- 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



(pointwise error, at 40 points)

* derivative evaluations required 150-200% overhead

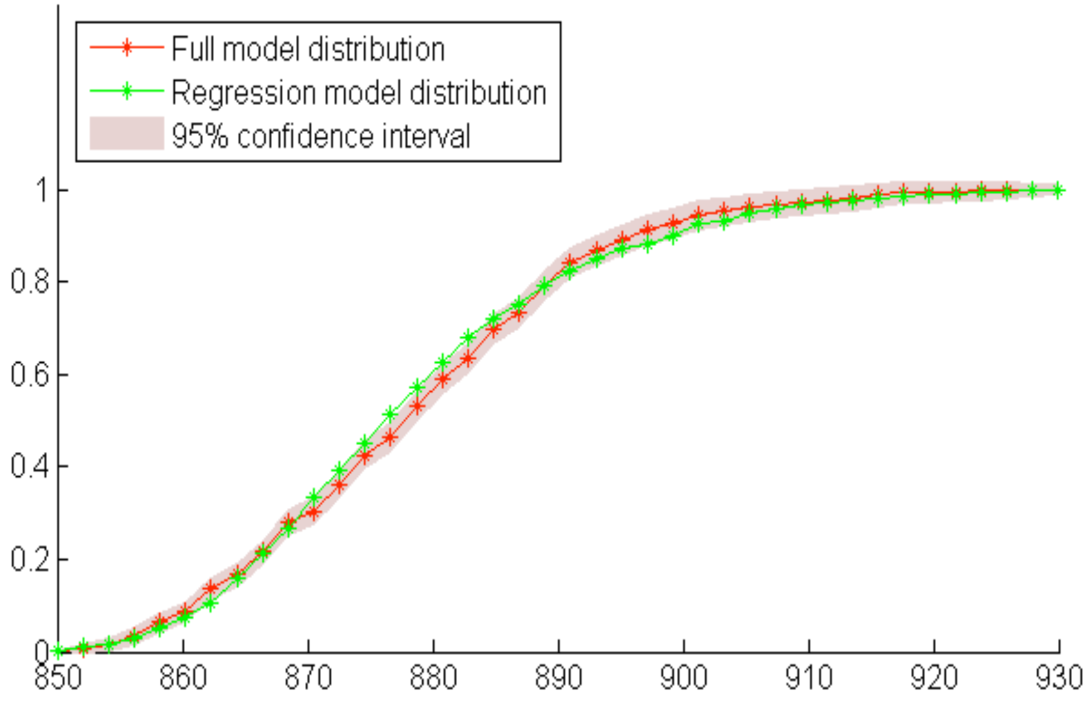
Uncertainty quantification, tests on subject models

- 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.

  Hermite polynomials are most appropriate where the statistical distribution of inputs are known (and normal!) In practical tasks, this is not the case.

  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 apply Gramm-Schmidt.

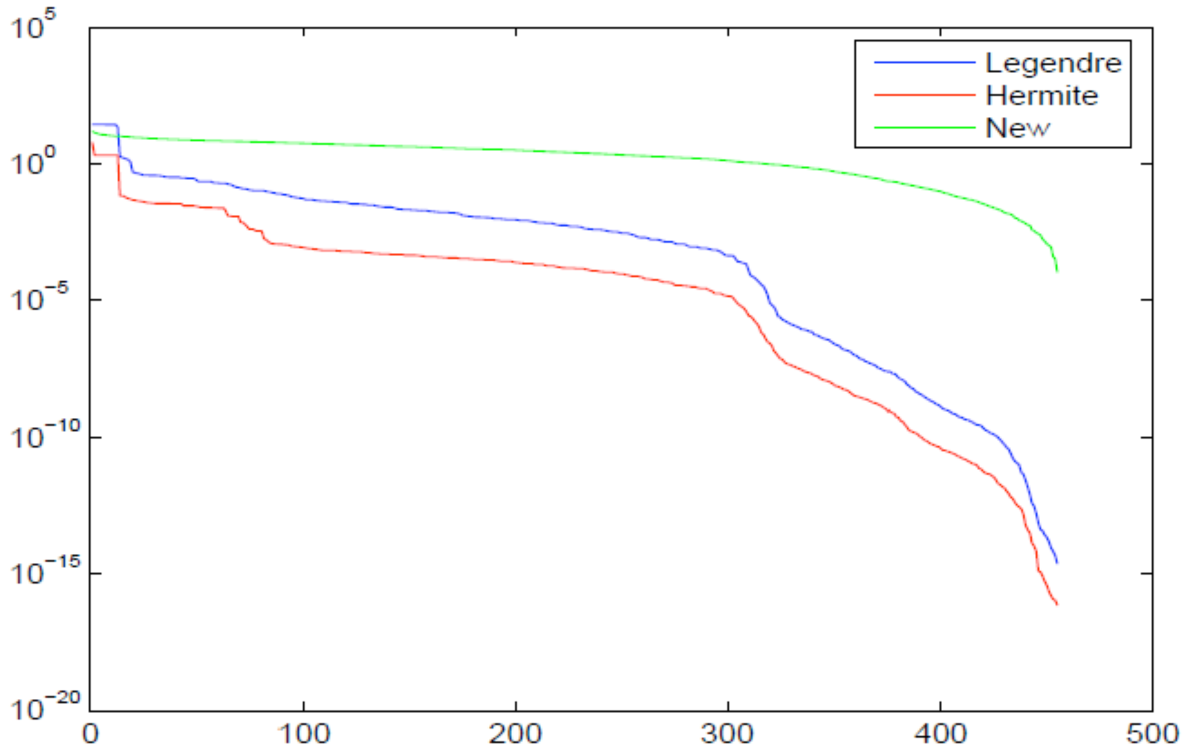
PRD, selection of better basis

- 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.

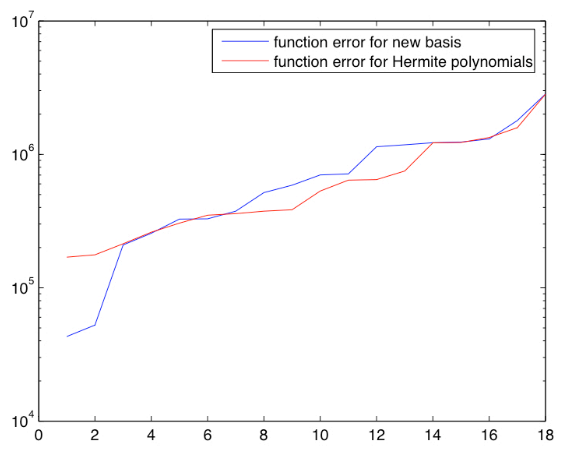
We compare numerical conditioning for Hermite, Legendre polynomials, and the basis based on new orthogonality conditions.

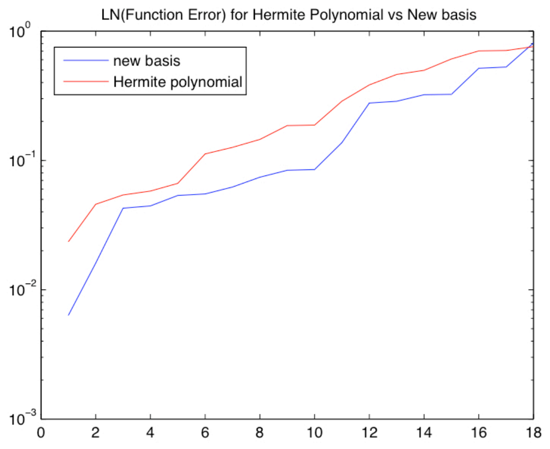
Testing of performance of PRD in this new setup is in progress.

- This will offer us substantial flexibility in creating the surrogate model.



Preliminary results of using reduced model for creating the surrogate





- The optimized base does a better job with less information. Here obtained with using fewer iterations in the steady-state neutronic model.

Conclusions

- PRD is a first step to a larger effort in learning the behavior of complex models by extracting more information from fewer sample runs.

- PRD outperforms classical methods of uncertainty quantification.

- An important part of PRD is Automatic Differentiation; it can be applied to codes of *industrial* complexity. We have some thoughts of choosing the basis.

- The current research topics are:

- selection of better basis

- selection of optimal training set

- prediction of error

- complexity reduction to create convenient representation of uncertainty

- application of PRD to models of higher complexity

PRD, computation of derivatives

- There is a possibility that some calculations in the model are inherently non-smooth

Some of the reasons are:

- incomplete convergence of a differentiable structure
- a switch between branches in the control structure of the code
- very stiff numerical effect

The options are:

- build a smoothing interpolation and differentiate that
- re-design just this model part to be smooth
- re-design just this model part to add capability to output it own (discontinuous?) derivative approximations, maybe by finite differences

- Detection of such places in the model flow requires automatic tracing tools. Fortunately, AD tools have this capacity (with some development effort required to search for specific features).

Uncertainty Quantification, UQ

- Task of UQ: to relate the information on uncertainty in the inputs (and parameters) of the model to the resulting variation in the outputs. Ultimately leads to improvement in efficiency and safety.

Difficulties:

- the information on statistical properties of inputs is scarce, disorganized.
- convenient representation of uncertainty is not available
- input space has large dimension
- model evaluation is computationally expensive
- it is unclear which points in input space are representative
- model code is too complex for direct study
- UQ is misunderstood: customers expect either fast, locally valid, a priori constructed solutions, or extensive code-rewriting.

Uncertainty Quantification, UQ

- Standards for UQ in nuclear engineering simulations can be improved:

“Complex”	means	“coupled system of several parts”,	should mean	“arbitrary complexity”
“Large”	means	“dimension 10, processed at once”,	should mean	“dimension 100, 1000”
“High-precision”	means	“error of 10%, improved by sampling”,	should mean	“error of 1%”

- We propose to efficiently model the propagation of uncertainty through complex simulation models. This is achieved by sampling-based methods, but with additional information for each considered point in the uncertainty space extracted by methods of automated learning.

- The goal is to construct a goal-oriented (a posteriori), globally valid, flexible representation of the effect of uncertainty on the outputs.