## Argonne <br> national laboratory

## Using Automatic Differentiation in Uncertainty Quantification of Nuclear Simulation Models

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

## Uncertainty Propagation.

■ 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.

We assume uncertainty model for $u \in R^{p}$ exists
$\square$ Challenge two: uncertainty propagation. Since $f$ is expensive to compute, we cannot expect to compute a statistic of u very accurately from direct simulations alone (and there is also curse of dimensionality).
We will thus incur an error in propagation, which needs to be quantified and reduced.

$$
y=\widetilde{f}(x)+\epsilon
$$

■ Initial Mathematical Challenges:

- What techniques allow me to work with a small number of samples?
- How do I quantify the error (propagation error) statistically even if the process is deterministic (not unlike weather forecast)?


## 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.

■ Issues
■ 1) How do I use Sensitivity Information in UQ when we have nonlinear models?

- 2) How do I get derivative information?


## Argonne

## 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.



## Argonne

## Representing Uncertainty, Setup

■ We use 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 $\quad T=\left(T_{1}, T_{2}, \ldots, T_{n}\right)$
* intermediate parameters and inputs $R=\left(R_{1}, R_{2}, \ldots, R_{N}\right)$
* that include errors $\Delta R=\left(\Delta R_{1}, \Delta R_{2}, \ldots, \Delta R_{N}\right)$

An output of interest is expressed by the merit function $J(T)$
The uncertainty is described by a set of stochastic quantifiers $\alpha=\left(\alpha_{1}, \alpha_{2}, \ldots, \alpha_{m}\right)$ whose statistical distribution we have.

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

## Argonne

## How to use derivative information?

- Using a linear uncertainty model may not be the answer for nonlinear model.
- But since I accept an error in the propagation term ....

$$
y=\widetilde{f}(x)+\epsilon
$$

■ ... I can use derivatives to obtain a smaller $\varepsilon$ even when f is nonlinear!

- We use Polynomial Regression with Derivative information to construct $\tilde{f}$. Due to the $\mathrm{O}(1)$ adjoint calculation effort independent of dimension of x , we will have better approximation for same cost.


## Argonne

## Polynomial Regression with Derivatives, PRD

- PRD procedure:
- choose a basis of multivariate polynomials* $\left.\Psi_{q}(\alpha)\right\rfloor$
the unknown function is then approximated by an expansion $\mathfrak{I}(\alpha) \approx \sum_{q} x_{q} \Psi_{q}(\alpha)$
- choose training set ${ }^{\star}\{A\}, \quad A_{i}=\left(\alpha_{1}^{i}, \alpha_{2}^{i}, \ldots, \alpha_{n}^{i}\right)$
- 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 equations (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.


## Argonne

## Polynomial Regression with Derivatives, PRD

- PRD procedure, regression equations:
- 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

ENERGY

## Argonne

## 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.

## Argonne

## 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 $J=J(T): \quad F(T, R(T, \alpha))=0$ the equations $\left(\frac{\partial F}{\partial T}+\frac{\partial F}{\partial R} \cdot \frac{\partial R}{\partial T}\right) \cdot \frac{d T}{d \alpha}+\frac{\partial F}{\partial R} \cdot \frac{\partial R}{\partial \alpha}=0$ are also solved, for $\frac{d T}{d \alpha}: \quad \frac{d J}{d \alpha}=\frac{d J}{d T} \cdot \frac{d T}{d \alpha}$
- 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.

## Argonne

## 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:
$P: \quad J=f_{k}\left(f_{k-1}\left(\ldots f_{1}(\alpha)\right)\right)$
The task of AD is to assemble a new program $P^{\prime}$ to compute the derivative. In forward mode:

$$
P^{\prime}: \quad\left(\nabla_{\alpha} J\right)_{i}=\frac{\partial f_{k}}{\partial f_{k-1}} \cdot \frac{\partial f_{k-1}}{\partial f_{k-2}} \cdot \ldots \cdot \frac{\partial f_{1}}{\partial \alpha_{i}}
$$

- 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:
$\left.P^{\prime}: \quad \nabla_{\alpha} J\right)=\left(\frac{\partial f_{1}}{\partial \alpha}\right)^{T} \cdot\left(\frac{\partial f_{2}}{\partial f_{1}}\right)^{T} \cdot \ldots \cdot\left(\frac{\partial f_{k}}{\partial f_{k-1}}\right)^{T}$
In adjoint mode, the complete gradient can be computed in a single run of $\mathrm{P}^{\prime}$, as opposed to multiple runs required by the direct mode.


## Tools: Fortran

- Fortran 95: (forward and reverse; source transformation)
- TAF (FastOpt)
- Commercial tool
- Support for (almost) all of Fortran 95
- Used extensively in geophysical sciences applications
- Tapenade (INRIA)
- Support for many Fortran 95 features
- Developed by a team with extensive compiler experience
- OpenAD/F (Argonne/UChicago/Rice)
- Support for many Fortran 95 features
- Developed by a team with expertise in combintorial algorithms, compilers, software engineering, and numerical analysis
- Development driven by climate model \& astrophysics code
- ADIFOR (Rice/Argonne) Mature and very robust tool, Support for all of Fortran 77, Forward and (adequate) reverse modes
- Hundreds of users; ~250 citations


## Argonne

## 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) issues 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).


## Argonne

## 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
$\square$ 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.


## Argonne <br> national laboratory

## 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, <br> K | Coolant temperature derivative, <br> 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 |

## Argonne

## 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):

|  | Samplin <br> $g$ | Linear <br> approximation | PRD, full <br> basis | PRD, <br> truncated <br> basis |
| :--- | :--- | :--- | :--- | :--- |
| Full model <br> runs | 100 | $1^{*}$ | $72^{*}$ | $12^{*}$ |
| Output <br> range, K | 2237.8 | 2227.4 | 2237.8 | 2237.5 |
| Error <br> range, K |  | -10.38 | -0.02 | -0.90 |
| Error st. <br> deviation |  | 2450.0 | 2460.5 | 2459.6 |

## Argonne

## 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.
$\square$ Surface response, error model

in progress (though control variate done)
(0) ENERGY

## Outstanding math issues

- AD:
- How do I handle nondifferentiability/discontinuity and still produce a model?

■ How do I deal with adaptive procedures?

- PRD:

■ How do I choose the polynomial basis (we use Hermite, but ...)

- How do I choose the sampling point?
- What is the appropriate error model for few samples?


## Argonne

## Conclusions

- PRD allows for UQ that leverages sensitivity calculations even for nonlinear models, and, as a results outperforms other UQ techniques.
- An important part of PRD is Automatic Differentiation; it can be applied to codes of *industrial* complexity.
- We have some ideas what to do even when only piecewise differentiability.

■ Future: bigger codes, fewer samples . $\qquad$

## Why Automatic Differentiation? (cont.)

- Alternative \#1:hand-coded derivatives
- hand-coding is tedious and error-prone
- coding time grows with program size and complexity
- automatically generated code may be faster
- no natural way to compute derivative matrix-vector products (Jv, J ${ }^{\top}$ v, Hv) without forming full matrix
- maintenance is a problem (must maintain consistency)
- Alternative \#2: finite difference approximations
- introduce truncation error that in the best case halves the digits of accuracy
- cost grows with number of independents
- no natural way to compute J J $v$ products


## Example: a simple function

```
#include <math.h>
#include <stream.h>
void func(double *f, double x, double y){
    double a,b;
    if (x > y) {
        a = cos(x);
        b = sin(y)*y*y;
    } else {
        a = x*sin(x)/y;
        b = exp (y);
    }
    *f = exp(a*b);
}
```


## Example: AD via operator overloading

```
#include <math.h>
#include <stream.h>
#include "adouble.hxx"
void func(a_double *f, a_double x, a_double y){
    a_double a,b;
    if (x > y) {
        a = cos(x);
        b = sin(y)*y*y;
    } else {
        a = x*sin(x)/y;
        b = exp(y);
    }
    *f = exp(a*b);
}
```


## Example: ADIC output

```
#include "ad_deriv.h"
#include <math.h>
#include "adintrinsics.h"
void ad_func(DERIV_TYPE *f,DERIV_TYPE x,DERIV_TYPE y) {
DERIV_TYPE a, b, ad_var_0, ad_var_1, ad_var_2;
doubl\overline{e} ad_adji_0,ad_\_loc_0, ad_\overline{loc_1},ad_a\overline{dj_0,ad_adj_1,ad_adj_2,ad_adj_3;}
if (DERIV_val(x) > DERIV_val(y)) {
    DERIV_val(a) = cos( DERIV_val(x)); /*cos*/
        ad_adjji_0 = -sin( DERIV_val(x));
            {
                ad_grad_axpy_1(&(a), ad_adji_0, &(x));
            }
    DERIV_val(ad_var_0) = sin( DERIV_val(y)); /*sin*/
        ad_adji_0 = cos( DERIV_val(y));
            {
            ad_grad_axpy_1(&(ad_var_0), ad_adji_0, &(y));
            }
            {
            ad_loc_0 = DERIV_val(ad_var_0) * DERIV_val(y);
            ad_loc_1 = ad_loc
            ad_adj_0 = DERIV_val(ad_var_0) * DERIV_val(y);
            ad_adj_1 = DERIV_val(y) * DERIV_val(y);
            ad_grad_axpy_3(&\overline{(b), ad_adj_1, &}(ad_var_0), ad_adj_0, &(y), ad_loc_0, &(y));
            DERIV_val(b) = ad_loc_1;
        }
    }
    else {
        // ...
```


## Capabilities

- Fast (O(1) function evaluation) computation of
- gradient (reverse)
- Jacobian-vector product (forward)
- transposed-Jacobian-vector product (reverse)
- Hessian-vector product (F+R, R+F, R+R)
- Efficient computation of full Jacobians and Hessians, when able to exploit:
- sparsity (combine structurally orthogonal columns/rows)
- low-rank structure (scarcity or ESM or ...)
- combinations of (near) sparsity and low-rank structure (scarcity/WIP)
- Efficient high-order directional derivatives
- Can compute high-order derivative tensors for modest number of independent variables (cost essentially proportional to number of unique entries in tensor)


## Challenges

- Full Jacobian or Hessian with no underlying structure (rare) can be expensive to compute
- Reverse mode requires storage of intermediate states
- Worst case storage proportional to \# flops in function evaluation
- In practice, use combination of storage/recomputation to reduce storage
- Well-known strategies for time-stepping computations, achieving (worst-case) logarithmic growth in cost of gradient
- Best performance usually requires an application-specific strategy
- Implementation of source transformation tools requires robust compiler infrastructure and compiler analyses
- Open-source, industrial-strength compiler infrastructures are rare
- Infrastructures often do not include full set of required analyses
- AD theory for full Fortran 90 / C++ languages not yet developed
- Provides only local information
- Must be combined with sampling to provide global information
- Note: local = linear
- Computes (chain-rule) derivatives of code


## Argonne

## 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 $\rho$; Gaussian measure is the specific choice): $\int_{\Omega} \Psi_{j}(A) \Psi_{h}(A) \rho(A) d A=\delta_{j h}$
$\square$ We formulate new orthogonality conditions:

$$
\int_{\Omega}\left(\Psi_{j}(A) \Psi_{h}(A)+\sum_{i=1}^{m} \frac{\partial \Psi_{j}(A)}{\alpha_{i}} \cdot \frac{\partial \Psi_{h}(A)}{\alpha_{i}}\right) \rho(A) d A=\delta_{i h}
$$

and apply Gramm-Schmidt.

## Argonne

## 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.
$\square$ This will offer us substantial flexibility in creating the surrogate model.


ENERGY

## Argonne

national laboratory

## 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.

## Argonne

## 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).


## Argonne

## 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.


## Argonne

## 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.


# UQ Challenges for Complex Nuclear Models DISCUSSION 

Mihai Anitescu and Hany Abdel-Khalik

## (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 )
$\square$ Challenge one: create model for $u \in R^{p} \quad$ 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.
$\square$ 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).
$\square$ Therefore, in UP, one must assess the error made in approximating the system response by $\widetilde{f}$


## Argonne

## C1: Quantification of Uncertainty Challenges

- Previous data processing approaches were tuned towards point estimates (see liquid Na $\mathrm{Cp})$. Nowadays we need functional estimates, to do uncertainty propagation through multiply coupled codes, where condition (such as T here) vary.
- The original data, however, may not be available, and experiments are expensive.
$\square$ Q1: Can we reconstruct realistic covariance information, using other constraining factors (such as smoothness here)?
$\square$ Q2: Can we mesh empirical point estimates with code-based higher order moments?
$\square$ Q3: What are better data models for UQ with high-fidelity physics?

(0) ENERGY


## C2: Propagation of uncertainty challenges

■ Once the probabilistic model exists, we need to propagate it through the code. $y=f(u)$ Nevertheless, expensive codes make it likely that we will not be able to reach the desired accuracy for a statistic of $y$, through sampling alone.

- We must then create a model of the response from a limited number of samples, which has its own error

$$
y=\widetilde{f}(x)+\epsilon
$$

$\square$ What is the appropriate statistical error model $\quad \mathcal{\text { of the approximation and how do we }}$ get the best approximation from a given amount of computational power (use derivatives ?)
$\square$ Can one use multiple codes with different physics resolution to do UP at a smaller cost? (and which is still of high quality)?
$\square$ Can I create high quality reduced model for the same purpose?
$\square$ Is full-system adjoint calculation ready for prime time, and can it be used to this end?

## How to Design Computational Experiments to reduce Uncertainty Propagation Error ?

- If running the computational experiments is expensive, then we must choose the best points at which to run them, the points which will result in the smallest error.

What is the best strategy for running such experiments?
Can we gain something by considering hierarchies of models with different physics? How should these combined experiments be designed?

## Spatio-temporal statistics

- How do I create high fidelity uncertainty fields for statistical downscaling?
- Can I use this to understand errors in approximate closure models, such as LES?
- Possible Solution Gaussian Processes (Kriging):
- Perception: Gaussian Distributions are easy:
- Fact: Computations of large scale Gaussian distributions is not resolved. Need matrix-free square root and determinant of Covariance Matrix.

$$
\log p(y \mid \theta)=-\frac{1}{2} W\left(\theta_{1}\right)^{T} K\left(\theta_{2}\right)^{-1} W\left(\theta_{1}\right)-\frac{1}{2} \log \left|K\left(\theta_{2}\right)\right|-\frac{n}{2} \log 2 \pi
$$

## Argonne <br> national laboratory

## How do I simulate from GP?

- Recommended solution: Take Cholesky of The
Covariance matrix, cannot possibly scale.
- Our solution: Matrix-free calculation of $\mathrm{Q}^{\wedge} 0.5^{*} \mathrm{~N}(0, \mathrm{I})$ (Chen, Anitescu, Saad) not really attacked before.
- Example: GP with $10^{\wedge} 6$ data sites.
- Future: How does it scale on multicore architectures?


ENERGY

