SAND2009-5069C

A Comparison of Intrusive Stochastic Galerkin Methods for Uncertainty Quantification of Stochastic PDEs

Eric Phipps
Optimization and Uncertainty Quantification Department
Sandia National Laboratories
Albuquerque, NM USA
etphipp@sandia.gov

10th US National Congress on Computational Mechanics
July 16, 2009

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94AL85000.

Sandia National Laboratories

Motivation For This Work

- Understand benefits of *intrusive* stochastic expansion methods in complex nonlinear PDEs
  - Uncertainty quantification
  - Multi-physics coupling
  - Optimization under uncertainty
- Provide software tools for studying these methods in complex nonlinear PDEs
- Focus here on
  - Approaches for generating stochastic Galerkin residual/Jacobian coefficients
  - How these methods compare for simple PDEs

Intrusive Stochastic Galerkin Uncertainty Quantification Methods

- **Steady-state stochastic problem:**

$$\text{Find } u(\xi) \text{ such that } f(u,\xi) = 0,\ \xi : \Omega \rightarrow \Gamma \subset R^M,\ \text{density } \rho$$

- **Stochastic Galerkin method (Ghanem and many, many others…):**

$$\hat{u}(\xi) = \sum_{i=0}^{P} u_i \psi_i(\xi) \rightarrow F_i(u_0, \ldots, u_P) = \int_\Gamma f(\hat{u}(y), y)\psi_i(y)\rho(y)dy = 0, \quad i = 0, \ldots, P$$

- **Basis polynomials are *usually* tensor products of 1-D orthogonal polynomials of degree N**

- **Method generates new coupled spatial-stochastic nonlinear problem**

$$0 = F(U) = \begin{bmatrix} F_0 \\ F_1 \\ \vdots \\ F_P \end{bmatrix}, \quad U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_P \end{bmatrix}$$

- **Total size grows rapidly with degree or dimension**

$$P = \frac{(M+N)!}{M!N!}$$

Sandia National Laboratories

Challenges for Intrusive SG

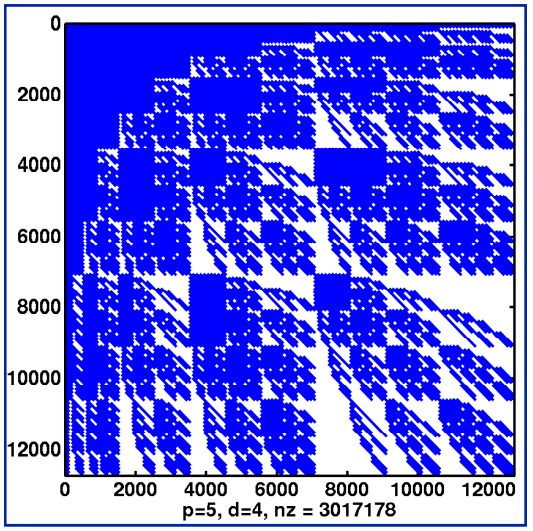
- Generating SG residual & Jacobian entries in nonlinear simulation codes

$$F_i = \int_\Gamma f(\hat{u}(y), y)\psi_i(y)\rho(y)dy, \quad \langle\cdot\rangle = \int_\Gamma \cdot\,\rho(y)dy$$

$$\frac{\partial F_i}{\partial u_j} \approx \sum_{k=0}^{P} J_k \langle \psi_i \psi_j \psi_k \rangle, \quad J_k = \frac{1}{\langle \psi_k^2 \rangle} \int_\Gamma \frac{\partial f}{\partial u}(\hat{u}(y), y)\psi_k(y)\rho(y)dy$$

- Solving resulting fully-coupled spatial-stochastic problem

- Study 3 methods for generating coefficients
  - Global assembly quadrature
  - Local assembly quadrature
  - Scalar operation propagation



Sandia National Laboratories

Semi-Intrusive Methods for Generating SG Residual and Jacobian Entries

- **Quadrature at global residual/Jacobian assembly level**

$$F_i \approx \sum_{l=0}^{Q} w_l f(\hat{u}(y_l), y_l)\psi_i(y_l), \quad J_i \approx \frac{1}{\langle \psi_i^2 \rangle} \sum_{l=0}^{Q} w_l \frac{\partial f}{\partial u}(\hat{u}(y_l), y_l)\psi_i(y_l), \quad i = 0, \ldots, P$$

  - Repeatedly call code's global residual/Jacobian fill interface
  - Parallel data structures only need add and scale functions
  - Take advantage of sparse-grid quadrature technology

- **Quadrature at local residual/Jacobian assembly level**

$$f(u, y) = \sum_{e=1}^{N_e} S_e^T f_e(G_e u, y) \implies F_i \approx \sum_{e=1}^{N_e} S_e^T \sum_{l=0}^{Q} w_l f_e(G_e \hat{u}(y_l), y_l)\psi_i(y_l),$$

$$G_e \hat{u}(y_l) = \sum_{i=0}^{P} (u_e)_i \psi_i(y_l)$$

  - Repeatedly call code's local residual/Jacobian fill interface
  - Allows use of BLAS for polynomial evaluation/integration

Sandia National Laboratories

Computing SG Residuals/Jacobians via Automatic Differentiation (AD)

- **Technology for computing analytic derivatives in simulation codes**
  - **Propagates derivatives at the scalar-operation level**
  - **Good tools available**
- **Provides deep interface into application code**
- **Leverage AD interface for any computation that can be done in an operation by operation manner**

$$y = \sin(e^x + x \log x), \quad x = 2$$

		x	$\frac{d}{dx}$
$x \leftarrow 2$	$\frac{dx}{dx} \leftarrow 1$	2.000	1.000
$t \leftarrow e^x$	$\frac{dt}{dx} \leftarrow t\frac{dx}{dx}$	7.389	7.389
$u \leftarrow \log x$	$\frac{du}{dx} \leftarrow \frac{1}{x}\frac{dx}{dx}$	0.301	0.500
$v \leftarrow xu$	$\frac{dv}{dx} \leftarrow u\frac{dx}{dx} + x\frac{du}{dx}$	0.602	1.301
$w \leftarrow t + v$	$\frac{dw}{dx} \leftarrow \frac{dt}{dx} + \frac{dv}{dx}$	7.991	8.690
$y \leftarrow \sin w$	$\frac{dy}{dx} \leftarrow \cos(w)\frac{dw}{dx}$	0.991	-1.188

SG Projections of Arithmetic Operations

- Assume that SG expansions for two intermediate variables a and b have been computed, and we wish to compute a third c

$$\text{Given } a(y) = \sum_{i=0}^{P} a_i \psi_i(y),\ b = \sum_{i=0}^{P} b_i \psi_i(y),\ \text{find } c(y) = \sum_{i=0}^{P} c_i \psi_i(y)$$

$$\text{such that } \int_{\Gamma} (c(y) \quad \phi(a(y), b(y))) \psi_i(y) \rho(y) dy = 0, \quad i = 0, \ldots, P$$

- Addition/subtraction

$$c = a \pm b \Rightarrow c_i = a_i \pm b_i$$

- Multiplication

$$c = a \times b \Rightarrow \sum_i c_i \psi_i = \sum_i \sum_j a_i b_j \psi_i \psi_j \rightarrow c_k = \sum_i \sum_j a_i b_j \frac{\langle \psi_i \psi_j \psi_k \rangle}{\langle \psi_k^2 \rangle}$$

- Division

$$c = a/b \Rightarrow \sum_i \sum_j c_i b_j \psi_i \psi_j = \sum_i a_i \psi_i \rightarrow \sum_i \sum_j c_i b_j \langle \psi_i \psi_j \psi_k \rangle = a_k \langle \psi_k^2 \rangle$$

Projections of Transcendental Operations e.g., c = exp(a)

- Taylor series approximations (Debusschere *et al*, UQ Toolkit)

$$c \approx \sum_{k=0}^{n} \frac{a^k}{k!}$$

  - Use arithmetic rules for evaluating Taylor polynomial
  - Convergence can cause problems

- Time integration (Debusschere *et al*, UQ Toolkit)

$$u(x) = \exp(x) \text{ is a solution to ODE } \frac{du}{dx} = u$$

  - Translate this to an ODE on coefficients of c
  - Call time integration package (e.g., CVODE)
  - More accurate and robust, but more expensive

- Quadrature

$$c_k = \frac{1}{\langle \psi_k^2 \rangle} \int_\Gamma \exp(a(y))\rho(y)dy \approx \sum_{l=0}^{Q} w_l \exp(a(y_l))$$

  - Simple implementation
  - Take advantage of sparse-grid technology
  - Call BLAS for polynomial evaluation and integration

Sacado: AD Tools for C++ Applications

- AD via operator overloading and C++ templating
  - Transform to template code & instantiate on Sacado AD types
  - Easy to add new AD types to a code
- Designed for use in complex C++ codes
  - Sacado::FEApp example demonstrates approach
- Very successful in enabling analytic sensitivity calculations in large-scale simulation codes
  - Charon, Aria, Xyce, Alegra, LAMMPS, Albany

- http://trilinos.sandia.gov
- Algorithms and enabling technologies
- Large-scale scientific and engineering applications
- C++ Object oriented framework

Stokhos: Trilinos Tools for Intrusive Stochastic Galerkin UQ Methods

- **Sacado overloaded operators for SG propagation**
  - Taylor & time integration approaches (UQ Toolkit – Najm, Debusschere, Knio, …)
  - Tensor product and sparse grid quadrature (Dakota – Mike Eldred and John Burkardt)

http://trilinos.sandia.gov

- **Tools solving SG linear systems**
  - Jacobian-free (Ghanem) or fully assembled
  - Mean-based preconditioning
  - Hooks to Trilinos parallel linear solvers

$$\frac{\partial F_i}{\partial u_j} \approx \sum_{k=0}^{P} J_k \langle \psi_i \psi_j \psi_k \rangle \implies$$

$$\left(\frac{\partial F}{\partial U} V\right)_i \approx \sum_{j=0}^{P} \sum_{k=0}^{P} J_k v_j \langle \psi_i \psi_j \psi_k \rangle$$

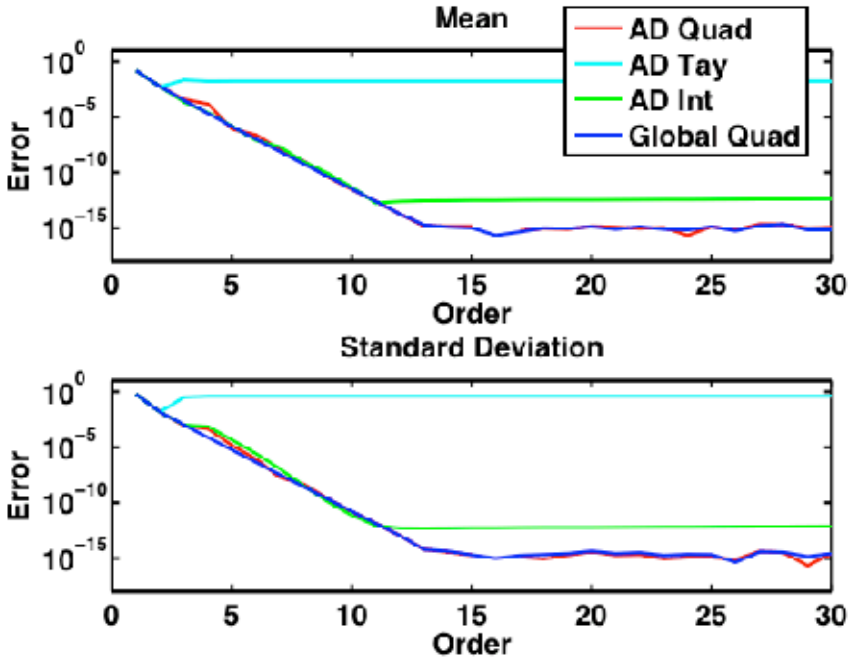
- **Nonlinear SG application code interface**
  - Interface to nonlinear solver, time integrator, optimizer
  - Provides global-level SG expansion method

Sandia National Laboratories

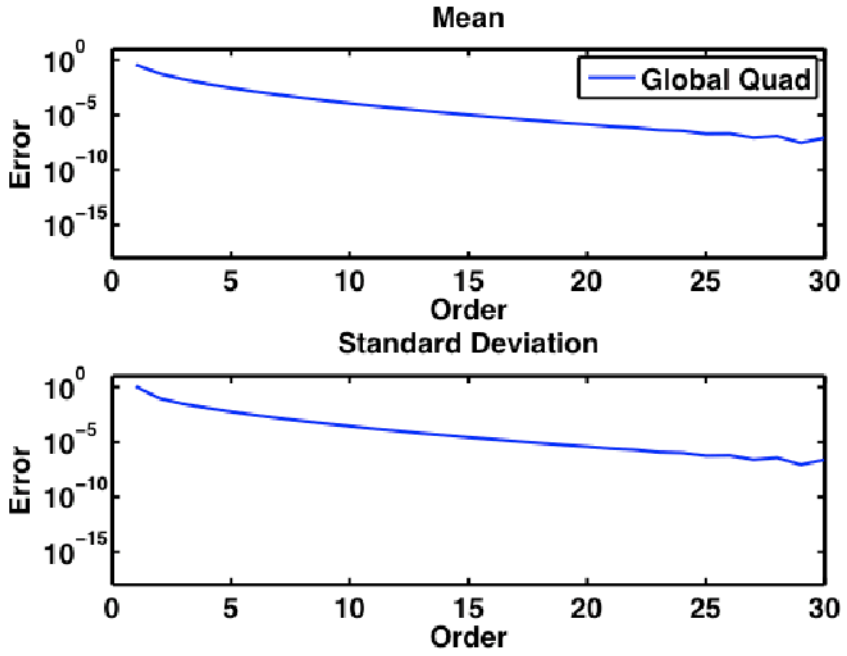
Accuracy of AD Approach

$$u = \log\left(\frac{1}{1 + (e^x)^2}\right)$$

Uniform U(-1,1) x



Gaussian N(0,1) x



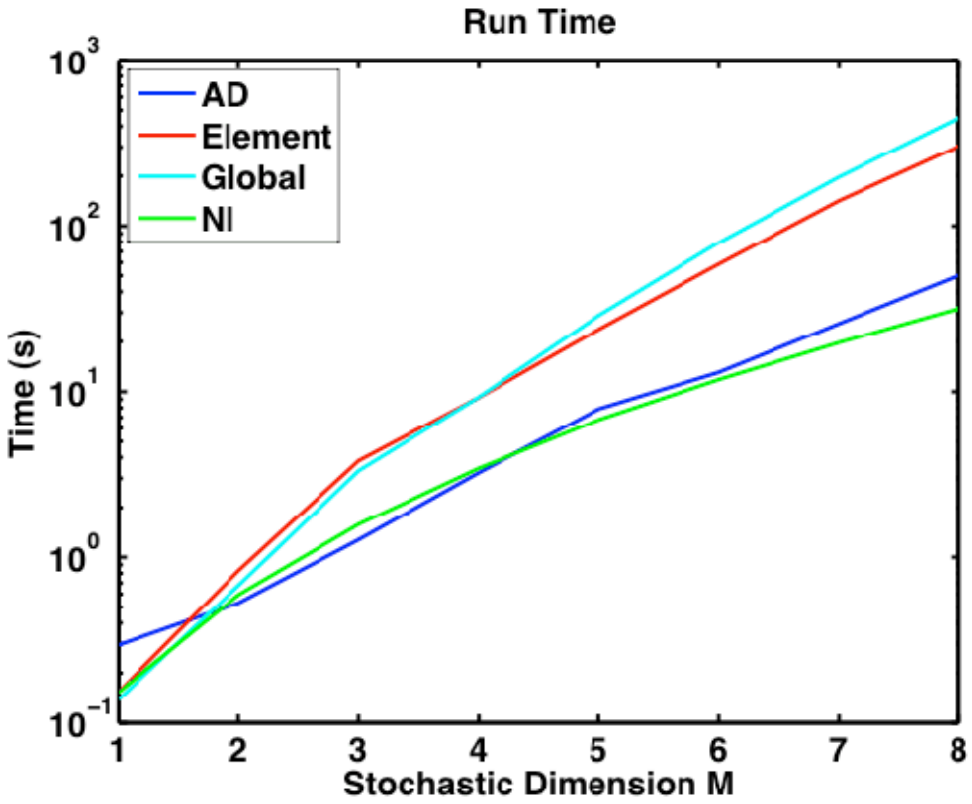
All 3 AD approaches fail

- AD approach is usually accurate
- Truncation error *can* cause catastrophic failure

Sandia National Laboratories

1-D Quadratic PDE

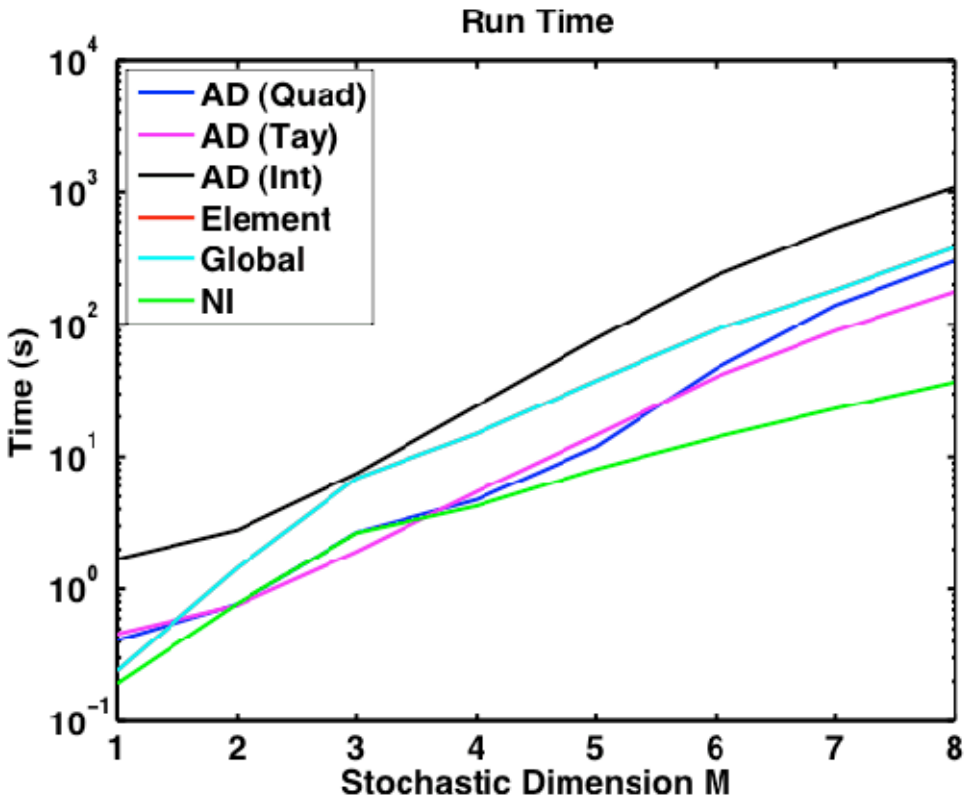
$$\frac{d^2u}{dx^2} + \frac{\alpha_1 + \cdots + \alpha_M}{M} u^2 = 0, \quad u \subset [\ \ 1, 1], \quad \alpha_i = U(1, 3)$$



- Sacado FEApp
- Clenshaw-Curtis sparse grid quadrature
- Algorithmic parameters chosen to give 1e-6 accuracy in 2nd moment
- AD approach significantly more efficient than element or global
- Intrusive times for larger M's dominated by mat-vec
- Preconditioner computation time not significant

1-D Exponential PDE

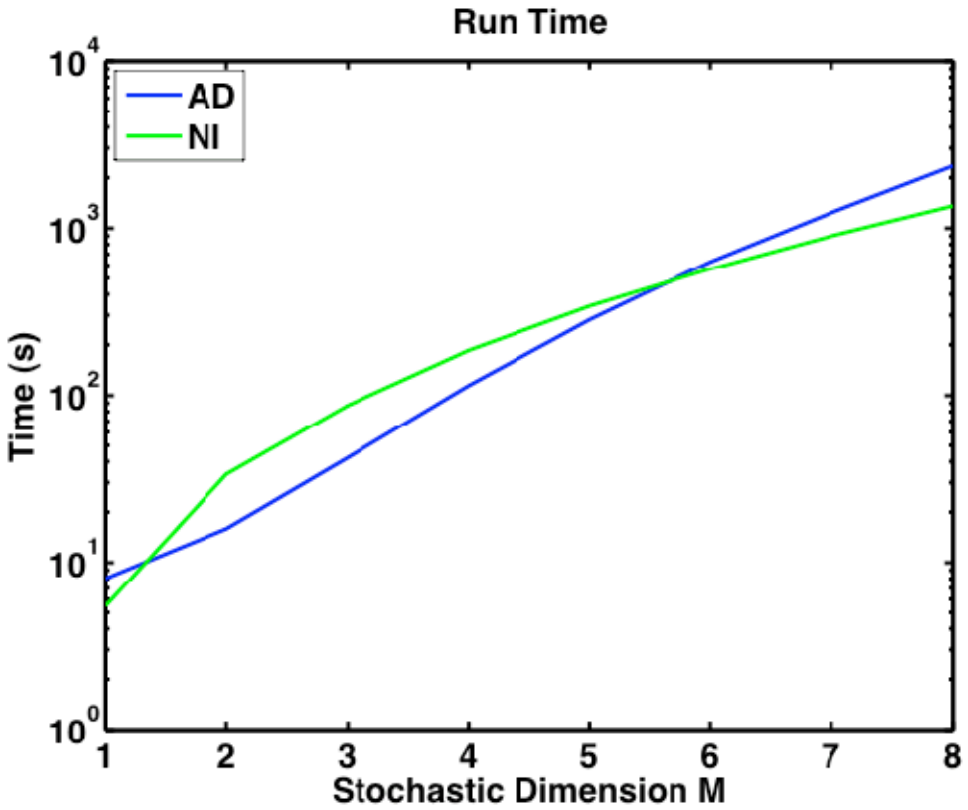
$$\frac{d^2u}{dx^2} + \frac{\alpha_1 + \cdots + \alpha_M}{M} e^u = 0, \quad \alpha_i = U(1,3)$$



- Intrusive significantly more expensive than non-intrusive
  - Fill cost is important
- AD approaches still more efficient than element or global
  - Likely diminish with more terms
- Quadrature fill times dominated by polynomial evaluation & integration

2-D Quadratic PDE

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\alpha_1 + \cdots + \alpha_M}{M} u^2 = 0, \quad u \subset [\ \ 1, 1] \times [\ \ 1, 1], \quad \alpha_i = U(1, 3)$$



- New Sandia Albany code
- More expensive preconditioner improves performance of intrusive approach
- Window likely increases for 3-D PDEs

Concluding Remarks

- Intrusive AD approach appears efficient for problems with polynomial nonlinearities
  - Many interesting PDEs fall into this category
- For transcendental nonlinearities, quadrature provides nice balance between efficiency and robustness
  - Difference between AD/Element/Global diminishes with more transcendental terms
  - New AD type to propagate all quadrature points simultaneously
- Now have capabilities to incorporate these ideas into complex codes
  - 3-D PDEs
  - Multi-physics coupling
- Trilinos packages Stokhos and Sacado
  - http://trilinos.sandia.gov
  - Sacado is currently available
  - Stokhos will be release with Trilinos 10.0, this fall

Auxiliary Slides

What is Automatic Differentiation (AD)?

- Technique to compute analytic derivatives without hand-coding the derivative computation
- How does it work -- freshman calculus
  - Computations are composition of simple operations (+, *, sin(), etc…) with known derivatives
  - Derivatives computed line-by-line, combined via chain rule
- Derivatives accurate as original computation
  - No finite-difference truncation errors
- Provides analytic derivatives without the time and effort of hand-coding them

$$y = \sin(e^x + x \log x), \quad x = 2$$

		x	$\frac{d}{dx}$
$x \leftarrow 2$	$\frac{dx}{dx} \leftarrow 1$	2.000	1.000
$t \leftarrow e^x$	$\frac{dt}{dx} \leftarrow t\frac{dx}{dx}$	7.389	7.389
$u \leftarrow \log x$	$\frac{du}{dx} \leftarrow \frac{1}{x}\frac{dx}{dx}$	0.301	0.500
$v \leftarrow xu$	$\frac{dv}{dx} \leftarrow u\frac{dx}{dx} + x\frac{du}{dx}$	0.602	1.301
$w \leftarrow t + v$	$\frac{dw}{dx} \leftarrow \frac{dt}{dx} + \frac{dv}{dx}$	7.991	8.690
$y \leftarrow \sin w$	$\frac{dy}{dx} \leftarrow \cos(w)\frac{dw}{dx}$	0.991	-1.188

Sandia National Laboratories

AD Takes Three Basic Forms

$$x \in \mathrm{R}^n,\ f : \mathrm{R}^n \rightarrow \mathrm{R}^m$$

- **Forward Mode:**

$$(x,\ V) \longrightarrow \left(f,\ \frac{\partial f}{\partial x} V\right)$$

  - Propagate derivatives of intermediate variables w.r.t. independent variables forward
  - Directional derivatives, tangent vectors, square Jacobians, $\partial f / \partial x$ when $m \geq n$

- **Reverse Mode:**

$$(x,\ W) \longrightarrow \left(f,\ W^T \frac{\partial f}{\partial x}\right)$$

  - Propagate derivatives of dependent variables w.r.t. intermediate variables backwards
  - Gradient of a scalar value function with complexity $\approx 4\ \mathrm{ops}(f)$
  - Gradients, Jacobian-transpose products (adjoints), $\partial f / \partial x$ when $n > m$

- **Taylor polynomial mode:**

$$x(t) = \sum_{k=0}^{d} x_k t^k \longrightarrow \sum_{k=0}^{d} f_k t^k = f(x(t)) + O(t^{d+1}),\quad f_k = \frac{1}{k!}\frac{d^k}{dt^k} f(x(t))$$

- **Basic modes combined for higher derivatives:**

$$\frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x} V_1\right) V_2,\quad W^T \frac{\partial^2 f}{\partial x^2} V,\quad \frac{\partial f_k}{\partial x_0}$$

Our AD Research is Distinguished by Tools & Approach for Large-Scale Codes

- Many AD tools and research projects
  - × Most geared towards Fortran (ADIFOR, OpenAD)
  - × Most C++ tools are slow (ADOL-C)
  - × Most applied in black-box fashion
- Sacado: Operator overloading AD tools for C++ applications
  - ✓ Multiple highly-optimized AD data types
  - ✓ Transform to template code & instantiate on Sacado AD types
  - ✓ Apply AD only at the "element level"
- This is the only successful, sustainable approach for large-scale C++ codes!
- Directly impacting QASPR through Charon
  - ✓ Analytic Jacobians and parameter derivatives

Basic Sacado C++ Example

```
#include "Sacado.hpp"

// The function to differentiate
template <typename ScalarT>
ScalarT func(const ScalarT& a, const ScalarT& b, const ScalarT& c) {
  ScalarT r = c*std::log(b+1.)/std::sin(a);

  return r;
}

int main(int argc, char **argv) {
  double a = std::atan(1.0);                    // pi/4
  double b = 2.0;
  double c = 3.0;
  int num_deriv = 2;                            // Number of independent variables

  // Fad objects
  Sacado::Fad::DFad<double> afad(num_deriv, 0, a); // First (0) indep. var
  Sacado::Fad::DFad<double> bfad(num_deriv, 1, b); // Second (1) indep. var
  Sacado::Fad::DFad<double> cfad(c);               // Passive variable
  Sacado::Fad::DFad<double> rfad;                  // Result

  // Compute function
  double r = func(a, b, c);

  // Compute function and derivative with AD
  rfad = func(afad, bfad, cfad);

  // Extract value and derivatives
  double r_ad = rfad.val();     // r
  double drda_ad = rfad.dx(0);  // dr/da
  double drdb_ad = rfad.dx(1);  // dr/db
```

Efficiency of AD in Charon

Set of N hypothetical chemical species:

$$2X_j \rightleftharpoons X_{j-1} + X_{j+1}, \quad j = 2, \ldots, N-1$$

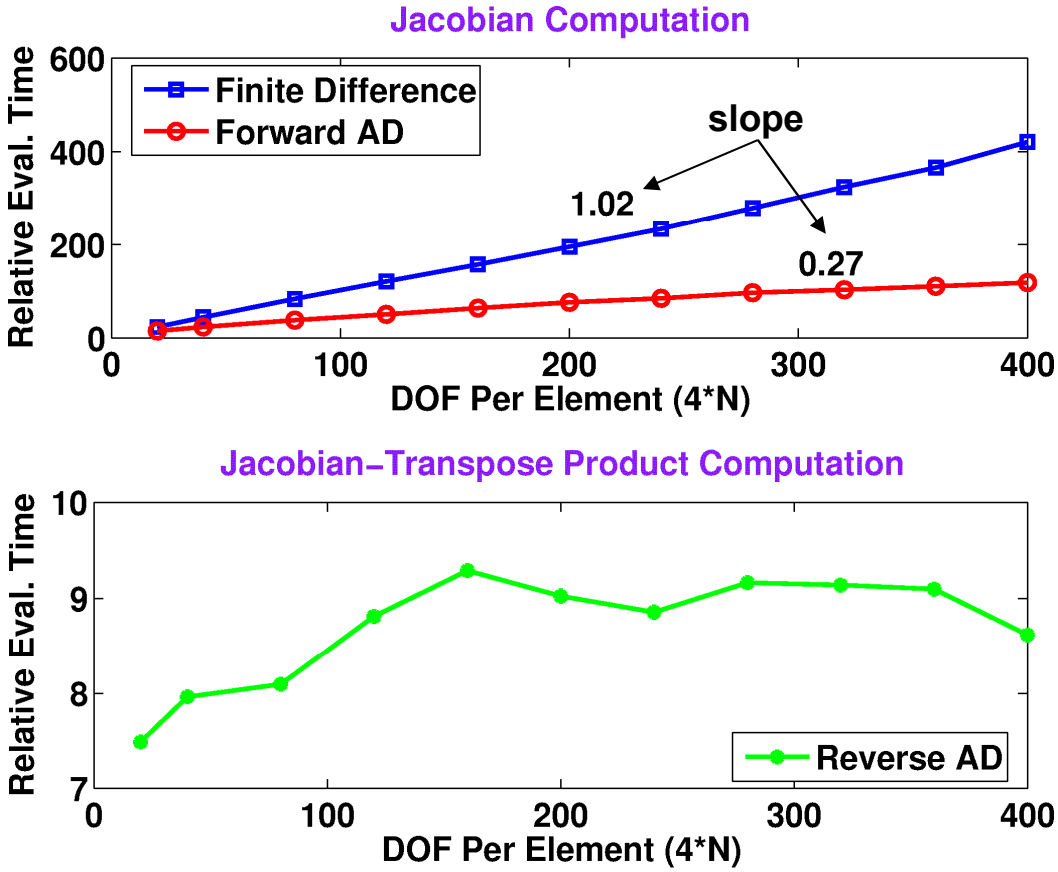
Steady-state mass transfer equations:

$$\nabla^2 Y_j + \mathrm{u} \cdot \nabla Y_j = \dot{\omega}_j, \quad j = 1, \ldots, N-1$$

$$\sum_{j=1}^{N} Y_j = 1$$

- **Forward mode AD**
  - **Faster than FD**
  - **Better scalability in number of PDEs**
  - **Analytic derivative**
  - **Provides Jacobian for all Charon physics**
- **Reverse mode AD**
  - **Scalable adjoint/gradient** $J^T w = \nabla(w^T f(x))$

Efficiency of the element-level derivative computation



Sandia National Laboratories

Charon Drift-Diffusion Formulation with Defects

CHARON

QASPR
QUALIFICATION ALTERNATIVES TO SPR

Current Conservation for e- and h+

$$\frac{\partial n}{\partial t} - \nabla \cdot J_n = -R_n(\psi, n, p, Y_1, \ldots, Y_N), \quad J_n = -n\mu_n \nabla\psi + D_n \nabla n$$

$$\frac{\partial p}{\partial t} + \nabla \cdot J_p = -R_p(\psi, n, p, Y_1, \ldots, Y_N), \quad J_p = -p\mu_p \nabla\psi - D_p \nabla p$$

Defect Continuity

$$\frac{\partial Y_i}{\partial t} + \nabla \cdot J_{Y_i} = -R_{Y_i}(\psi, n, p, Y_1, \ldots, Y_N), \quad J_{Y_i} = -\mu_i Y_i \nabla\psi - D_i \nabla Y_i$$

Electric potential

$$-\nabla(\varepsilon \nabla\psi(x)) = -q\left(p(x) - n(x) + N_D^+(x) - N_A^-(x)\right) - \sum_{i=1}^{N} q_i Y_i(x)$$

Recombination/ generation source terms

R_X

Include electron capture and hole capture by defect species and reactions between various defect species

Electron emission/capture

$Z^i \leftrightarrow Z^{i+1} + e^-$

$$R_{[Z^i \to Z^{i+1} + e^-]} \propto \sigma_{[Z^i \to Z^{i+1} + e^-]} Z^i \exp\left(\frac{\Delta E_{[Z^i \to Z^{i+1} + e^-]}}{kT}\right)$$

Activation Energy

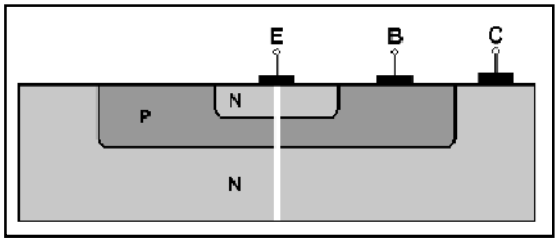
Cross section

Sandia National Laboratories

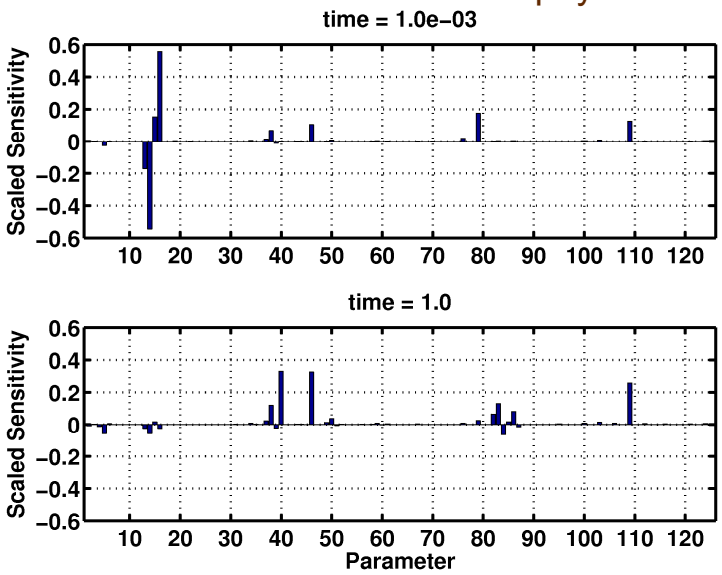
Rythmos Sensitivity Analysis Capability Demonstrated on the QASPR Simple Prototype*

*Phipps *et al*

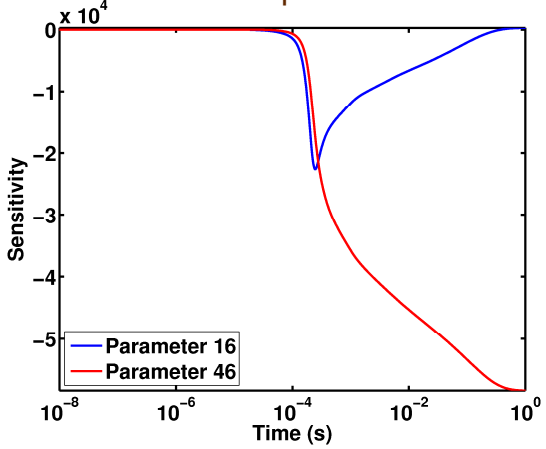
- Bipolar Junction Transistor
- Pseudo 1D strip (9x0.1 micron)
- Full defect physics
- 126 parameters



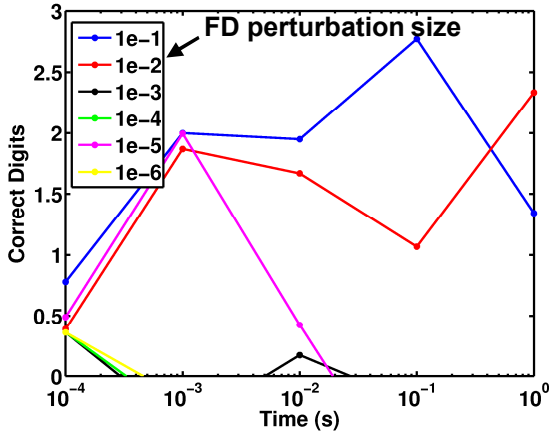
Sensitivities show dominant physics



Sensitivities computed at all times



1st-order Finite Difference Accuracy



Comparison to FD:

- ✓ Sensitivities at all time points
- ✓ More accurate
- ✓ More robust
- ✓ 14x faster!