A Unified Unintrusive Framework for Sensitivity Analysis and Uncertainty Propagation

Sanjay R. Mathur
Purdue University

Annual Review
October 25 and 26, 2010
Unified Approach

Tangent Mode Sensitivity Analysis

\[ J = \begin{bmatrix} \frac{\partial y_m}{\partial x_n} \end{bmatrix}_{m \times n} \]

- Start with discrete pde
- Consider code as set of unary and binary operations
- Propagate derivatives from inputs to outputs

\[
\begin{align*}
  z &= f(u) \\
  z' &= \frac{\partial f}{\partial u} u' \\
  z &= f(u,v) \\
  z' &= \frac{\partial f}{\partial u} u' + \frac{\partial f}{\partial v} v'
\end{align*}
\]

- Exploit C++ language features such as templating and operator overloading to achieve this elegantly

Generalized Polynomial Chaos (gPC)

- Start with discrete pde
- Consider code as set of unary and binary operations
- Expand all code variables in polynomial bases with unknown coefficients
- Galerkin-project all operations in code, from input to output, propagating pdfs
- Exploit C++ language features such as templating and operator overloading to achieve this elegantly
Sensitivity Analysis: Simple Example

Original Functions:

\[ p = 3 * x^2 + \sin(y) \quad q = p / y \]

x, y: inputs    p, q: outputs

Elemental Decomposition

\[ t_1 = x * x \]
\[ t_2 = 3 * t_1 \]
\[ t_3 = \sin(y) \]
\[ p = t_2 + t_3 \]
\[ q = p / y \]

Elemental Derivatives

\[ t_1' = x * x' + x' * x \]
\[ t_2' = 3 * t_1' \]
\[ t_3' = \cos(y) * y' \]
\[ p' = t_2' + t_3' \]
\[ q' = (p' * y - y' * p) / y^2 \]
Simple Example (Cont’d)

Inputs:
\( x = 10, \ y = \pi/3, \ x' = 1, \ y' = 0 \)

Outputs:
\[
\begin{align*}
\partial' &= 60.0 = \frac{\partial p}{\partial x} \bigg|_{x=10, y=\pi/3} \\
\partial' &= 273.8795 = \frac{\partial q}{\partial y} \bigg|_{x=10, y=\pi/3}
\end{align*}
\]

Derivatives are exact!

Inputs:
\( x = 10, \ y = \pi/3, \ x' = 0, \ y' = 1 \)
What did we get?

- **Exact** value of derivative wrt variable whose prime was set to unity
  - Not subject to truncation
- Each variable and its prime must be stored
- Obtains numerical value of derivative, not symbolic
- Process works through loops, conditionals and iterations
A bit of gPC...

- Expand in a polynomial basis

\[ k = \sum_i k_i(x) \]

- Substitute into governing equation

\[ \nabla \cdot k(\varepsilon) \nabla T = 0 \]
\[ k = 1 + \varepsilon x \]

- Perform Galerkin projection to obtain separate pdes for unknown coefficients \( T_i \)

- Solve pdes, and post-process polynomial expansion to find mean, variance and higher moments of \( T \) as necessary

Very intrusive!

- Need new pdes for the coefficients
- New solution methods for coupled non-linear pdes
- Each new class of physical models needs new coding
- All new pdes need to be parallelized
gPC in MEMOSA

- Expand all variables in code in polynomial bases

\[ a = \sum_i a_i P_i(\varepsilon) \quad b = \sum_i b_i P_i(\varepsilon) \quad c = \sum_i c_i P_i(\varepsilon) \]

- Each variable is now templated as being of the PC class
  - Carries coefficients \( a_i, b_i, c_i \)
  - To compute \( c = a \cdot b \) for example,
    - Overload '*' operation to multiply two series, do a Galerkin projection and isolate \( c_i \)
  - Need to define all operators (*, /, =)
    - Done using UQ Toolkit from Sandia

- MEMOSA looks like a regular deterministic CFD solver
  - Can choose gPC version at compile time
  - No need to re-code everything
  - Readable code – looks like deterministic code
  - No need to address each new model/physics separately
  - No new discretization or solution procedures
  - No extra parallelization work

What about accuracy? Speed?
Let the compiler do it!

- Use C++ language features
  - User-defined data types (classes)
  - Operator overloading
  - Templates and template meta-programming
  - Static initialization
  - Shared libraries

We define a template for the whole code which can be instantiated for any well defined algebra
C++ Implementation

Original C++ code

```cpp
void myfunc (const double& x, const double& y, double& p, double& q){
    p=3*x*x+sin(y);
    q=p/y;
}
```

Templated C++ code

```cpp
Template <class T>
void myfunc (const T& x, const T& y, T& p, T& q){
    p=3*x*x+sin(y);
    q=p/y;
}
```

*Tangent* class T contains value and derivative

*PC* class T contains variable and all coefficients of polynomial expansion

Operators overloaded appropriately

Can compile tangent or gPC version & choose gPC order and number of random vars

---

Tangent class T contains value and derivative

PC class T contains variable and all coefficients of polynomial expansion

Operators overloaded appropriately

Can compile tangent or gPC version & choose gPC order and number of random vars
Verification: 1D Heat Conduction with Random Thermal Conductivity

\[
\nabla \cdot k(\varepsilon) \nabla T = 0
\]

\[
k = 1 + \varepsilon x
\]

Gaussian with Mean=0; SD=0.1

Comparison with exact solution well under 0.001%
Driven Cavity with Random Viscosity

Gaussian with Mean=1; SD=0.1

- MEMOSA expands all variables in 3rd order polynomial expansion in Hermite polynomials
- Comparison with collocation gPC and Monte Carlo
- Collocation:
  - Construct response surface of velocity field using 2nd order polynomial expansion in Legendre polynomials
  - Determine coefficients in expansion using collocation with Smolyak sparse grid
- Monte Carlo sampling with 500 and 1000 samples
- Gaussian pdf for viscosity
U-Velocity Mean and SD: Vertical Centerline

Mean of u velocity

Standard deviation of u velocity

$\text{Re}=100$
V-Velocity Mean and SD: Vertical Centerline

Mean of $v$ velocity

Standard deviation of $v$ velocity

$Re=100$
Closure

• Flexible and versatile code basis
• Architecture admits sensitivity analysis and uncertainty propagation easily
• Carries over to all new physics that may be added to the code with no extra work
• Works automatically on parallel platforms
• More testing underway for non-linear problems and to establish timing