Code-Based Sensitivities for Verification and Validation

Adifor at LANL

Mike Fagan
Dept. of Computational and Applied Mathematics
Rice University

http://lacsi.rice.edu/review/slides_2006

LACSI

What's Coming Up

- Code-based Sensitivity Background
- Code-based Sensitivity for VnV
- Some Research Results
- Application to Truchas
- Near and Far Term Possibilities

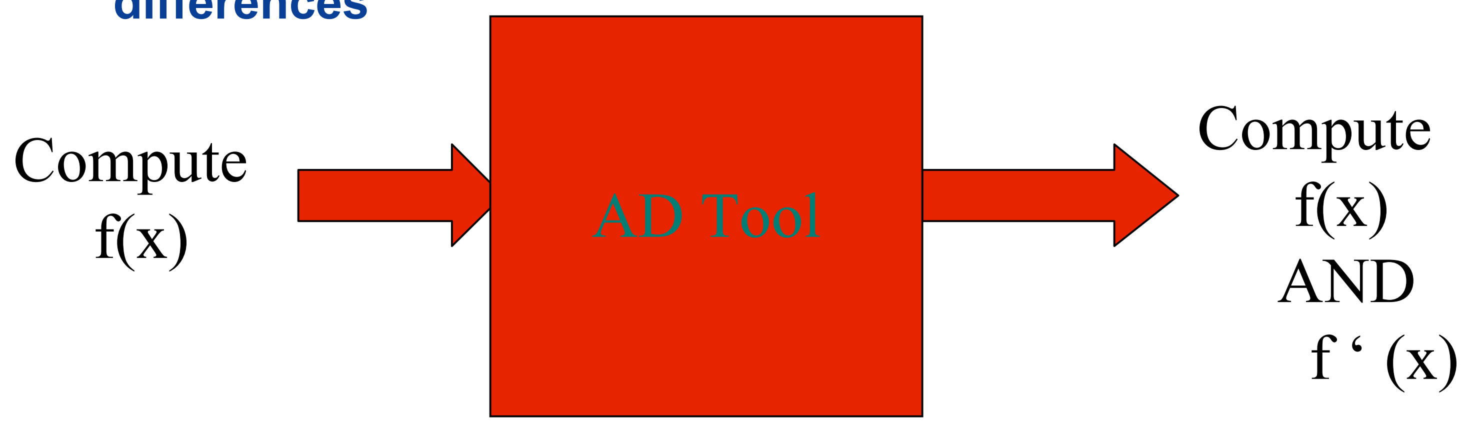
Sensitivity Calculation Methods

- **Finite Differences**
  - —**Development time is minimal +**
  - —**Choosing a perturbation ("h") –**
  - —**Inaccurate and/or inefficient –**
  - —**No reverse/adjoint mode –**
- **By Hand**
  - —*Can be* **accurate and efficient +**
    (depends on the programmer)
  - —**Development time is long –**
  - —**Maintaining derivatives an additional burden –**

Is there anything else ? …

What is Code-based Sensitivity?

- **Combines the best of finite differences and by hand sensitivity calculation**
- **Program generation tool**
  - —**Short development time**
- **Note on vocabulary: Automatic differentiation (AD) is synonymous**
- **Derivatives computed this way are**
  - —**Analytically accurate**
  - —**Always faster than central differences, frequently faster than 1-sided differences**



How does it work?

- **Each assignment statement is augmented with derivatives**
- **Chain rule assures propagation is correct**

```
Y = A * X ** 2 + B
```

↓

```
P_A = 2 * X
P_X = A
P_B  = 1.0
CALL ACCUM(G_Y,P_A,G_A,P_X,G_X,1.0,G_B)

Y = A * X ** 2 + B
```

Verification and Validation

Validation and Verification using Code-based Sensitivity

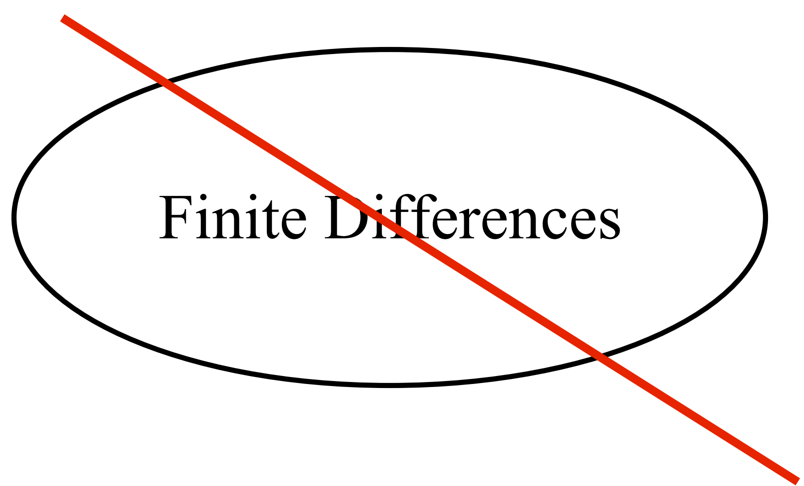
- Validation by inspection
- Validation by regression
- Method of Manufactured Solutions
- Running error bounds

Validation by inspection

- Informal, but valuable method used by physicists/modelers/engineers everywhere
- Complex simulations have many parameters:
  - —Material properties / equations of state
  - —Geometry
  - —Boundary conditions
- Some of the simulation parameters are known with great accuracy, some not
- Similarly, some of the parameters have a big effect on the output, others not so much
- The effect of a given parameter = sensitivity of out w.r.t. parameter

Validation by inspection, cont.

- **Physicists/modelers/engineers validate output by inspecting values and sensitivities**
  - —Output might be “off” because a highly sensitive parameter has not been accurately measured
  - —Intuition about the sensitivities themselves aids validation process
- **Code-based sensitivity computes analytic derivative values, so**



LACSI

Validation by Regression

- **More formal validation methodology**
  - —Separate “real world” data into 2 partitions: “tuning” and “testing”
  - —*Optimize the parameter settings on the “tuning” data to minimize simulation vs “real world”
  - —Assuming the error in the tuned simulation is “small”
    - – Run the tuned simulation on the “testing” data set
    - – Check for “small” error
- **Many variations on this methodology**
  - —How to separate data
  - —How to determine “small”

Validation by Regression, cont.

- The tuning step of this validation method can use Newton's method to obtain optimal values
- Newton's method runs best with analytic derivatives
- Code-based sensitivity supplies the derivatives

Method of Manufactured Solutions (MMS)

- Way of verifying differential equation solvers
- Given a solver S, a differential operator D, and a forcing function F
  - —S(D,F) computes f s.t. D(f) = F (approximately)
- MMS
  - —"manufacture" an f
  - —compute D(f)(x) for several x, use this as the manufactured F
  - —Now check S(D,F) vs f. Can verify order of accuracy, etc.
- Use code-based sensitivity to compute D(f), for moderately complex subroutines f

Running Error Bounds

- Wilkinson idea: estimate the roundoff error inherent in any assignment statement
- Not exactly the same as derivatives, but similar source augmentation
- Caveat: rules for intrinsics (like sin,cos) not so well known
- Caveat 2: roundoff error for sin,cos usu not as important as truncation error

$$z = a + b$$
$$eb1 = a - (a+b) + b$$

Current Research Results

LACSI

Code-based Sensitivity for Fortran 90 Programs

- Adifor works well on Fortran 77
- Fortran 90, however, has substantial language features
  - —Dynamic memory allocation
  - —Derived types (=structures)
  - —Pointers
  - —Operator and interface overloading
  - —Modules
- Adifor90 prototype works on Fortran 90 programs

Activity Analysis for Fortran 90

- Some variables in a computation may not need sensitivities
  - —Example: geometry might be constant
- Variables whose derivatives are provably 0 need not be computed
- Adifor activity analysis extended to Fortran 90

By Name/ By Address

- **Program derivatives represented in 2 ways:**
  - —By name:
    Another variable holds the derivatives: x → g_x
    augment calls with additional args: call f(x) → call g_f(x,g_x)
  - —By address:
    All active variables (or components) have a derived type:
    real → active real == { real v; real d }
    procedures signatures are changed (but call sites not changed): sub f(real x) → sub g_f(active_real x)
- **By name is smoother for languages with derived types and array slicing operations (F90)**

```
x(1:10) → g_x(1:10)      !! By name
x(1:10) → x(1:10)%v      !! Attempt By address - Not valid !!
```

LACSI

By Name / By Address, cont.

- By address is smoother for constant interface functions (like mpi_reduce)

  call mpi_reduce(sendbuf,recvbuf,cnt,dtatype,op,root,comm,ierr)

  cannot add a g_sendbuf, etc

- Found a way to do by-address for F90 (also works for F77!)
- Also found a way to do by-name for C

Holomorphic Functions

- Rules of calculus the same, so complex valued functions are no problem UNLESS
  - Use abs, or real, imag
- Sometimes, programs written using non-holo primitives are still holomorphic
- Found a way to preserve this
- Side benefit: you can computationally check the cauchy conditions for your code

Adifor90 on Truchas

- **During the week of 23 Jan, I installed Adifor90 prototype on CCS-2 machine, and have begun differentiating Truchas system**
- **Truchas is a metal casting code (and MORE – Jim Sicilian)**

Truchas Properties

- 267 files (not including some package components)
- 2542 functions/subroutines
- 104629 lines of code = 70500 non comments (approx)
- Uses derived types, memory allocation, pointers, overloading via interface blocks, modules, and local subprograms
- Does NOT use equivalence or common blocks

Truchas Checkout

- 25 routines checked out (more by time I give this talk)
- Sample results from an elliptic integral routine
  elk(0.5) = 1.854074677301372
  fd (0.001) = 0.8481413948864258

  ad = 0.8472143556167433

Near Term

- Finish all of Truchas in black-box mode by end of 2006 contract
  - —Differentiate pgslib (semi-auto)
- Investigate how to avoid solver differentiation in Truchas
- Generalize both of these tasks (upgrade to full auto)
- Continue to improve the storage efficiency of reverse mode

Future Possibilities

- Differentiation of other languages of interest
  - —Ajax system
    - – FLAG code
  - —C / C++
  - —Python
  - —Machine code (ie source unavailable)
- Differentiate Stochastic simulations
  - —Stochastic calculi
  - —If statements get different treatment
- Other sensitivity
  - —Intervals
  - —Probability distributions

Future Possibilities, cont.

- **Improve performance by enabling actual Newton methods**

F(x + t*v) – F(x) / t ! Directional derivatives

Replace with

G_F(x,v)