Automatic Differentiation: Beyond Adifor 2.0

Chris Bischof (Argonne)

Alan Carle (Rice)

Jason Abate, Mike Fagan

Paul Hovland, Lucas Roh

Andrew Mauer-Oats

May 21, 1997

Science and engineering needs derivatives

- Numerical Methods
  - Solutions of ODE, PDE
- Sensitivity Analysis
  - Assess parameter impact, model validity & robustness
- Inverse Problems: adjust model parameters to fit observed data
  - Parameter Identification: e.g., reaction coeff's
  - Data Assimilation: e.g., adjoint methods for adjusting initial states of PDE-based systems
- Derivative-based (design) optimization

How to compute derivatives

The hard way:
Accuracy and speed require human effort

	Work	Accuracy	Speed
Handcoding	☹☹	✔	✔✔
Divided Differences	☺	???	✔
Symbolic-Assisted	☹	✔	✔

The easy way:
Fast & accurate derivatives, fast

Automatic Differentiation	☺☺	✔	✔
AD+Brains	☺	✔	✔✔

Automatic Differentiation

- Automatic Differentiation (AD)
  - Efficient derivative programs with very little human effort.
- Computational Differentiation = AD + Brains
  - Methodologies for exploiting high-level user knowledge about program structure and mathematical and algorithmic underpinnings.

Accurate to machine precision, applicable to programs of arbitrary length and complexity, trivial to update derivative codes

What we have delivered, so far

- **ADIFOR 2.0 (Fortran 77)**
  - **350 requests**
  - **Supported on AIX, HPUX, IRIX, Linux, Solaris, Sun OS, Windows 95/NT**
  - **Book to be finished Summer 97**
- **ADIC (ANSI-C, some C++)**
  - **Version 1.0 released**

Lots of interactions with application groups including NASA MDOB, UTRC, HP, TI, PNL, Ford, ...

AD research issues

- Getting the desired answer (ODE's, conj. grad.)
  - Explaining AD results (derivatives of programs)
  - Post-processing AD results
- Getting answers faster
  - Exploit mathematics and program structure
  - Context-sensitive approaches to AD
  - Adjoints
  - Using parallelism
- New AD functionality
  - Hessians
- Integration with PSE's and libraries (NEOS, PETSc)

PSE's

- Adifor integrated into NEOS Network-Enabled Optimization Server
  - SparsLinC allows efficient computation of gradients for large, partially separable, problems.
- Automated generation of derivatives required for standard PETSc solvers both for Fortran and C
  - User only needs to provide the (serial) "patch update" function.
  - AD drivers encapsulated in PETSc functions.

AD development issues

- Lots of other languages to support
  - Fortran 77, ANSI-C
  - Fortran 90/95, HPF, MPI, PVM,SMP
  - C++
  - JAVA
- Too few FTE’s...

Develop an AD architecture that couples language-specific frontends and backends with language-independent shared modules.

ADIFOR-generated Code

```
w = z(1)*z(2)*z(3)*z(4) =>
```

```
r1_v=z(1)*z(2)
r2_v=r1_v*z(3)     dw/dz(4)
r1_b=z(4)*z(3)
r2_b=z(4)*r1_v     dw/dz(3)
r3_b=r1_b*z(2)     dw/dz(1)
r4_b=r1_b*z(1)     dw/dz(2)

do g_i = 1,g_p
   g_w(g_i) =   r3_b * g_z(g_i,1)
              + r4_b * g_z(g_i,2)
              + r2_b * g_z(g_i,3)
              + r2_v * g_z(g_i,4)
enddo

w = r2_v * z(4)
```

reverse mode

forward mode

function value

ADIC-generated Code

```
w = z[1]*z[2]*z[3]*z[4]  =>
```

```
ad_loc_0 = DERIV_VAL(z[1])*DERIV_VAL(z[2]);
ad_loc_1 = ad_loc_0 * DERIV_VAL(z[3]);      dw/dz[4]
ad_loc_2 = ad_loc_1 * DERIV_VAL(z[4]);
ad_adj_0 = ad_loc_0 * DERIV_VAL(z[4]);      dw/dz[3]
ad_adj_1 = DERIV_VAL(z[3])*DERIV_VAL(z[4]);
ad_adj_2 = DERIV_VAL(z[1])*ad_adj_1;        dw/dz[2]
ad_adj_3 = DERIV_VAL(z[2])*ad_adj_1;        dw/dz[1]

ad_grad_axpy_4( DERIV_GRAD(*w),
                ad_adj_3,DERIV_GRAD(z[1]),
                ad_adj_2,DERIV_GRAD(z[2]),
                ad_adj_0,DERIV_GRAD(z[3]),
                ad_loc_1,DERIV_GRAD(z[3]))

DERIV_VAL(*w) = ad_loc_2;
```

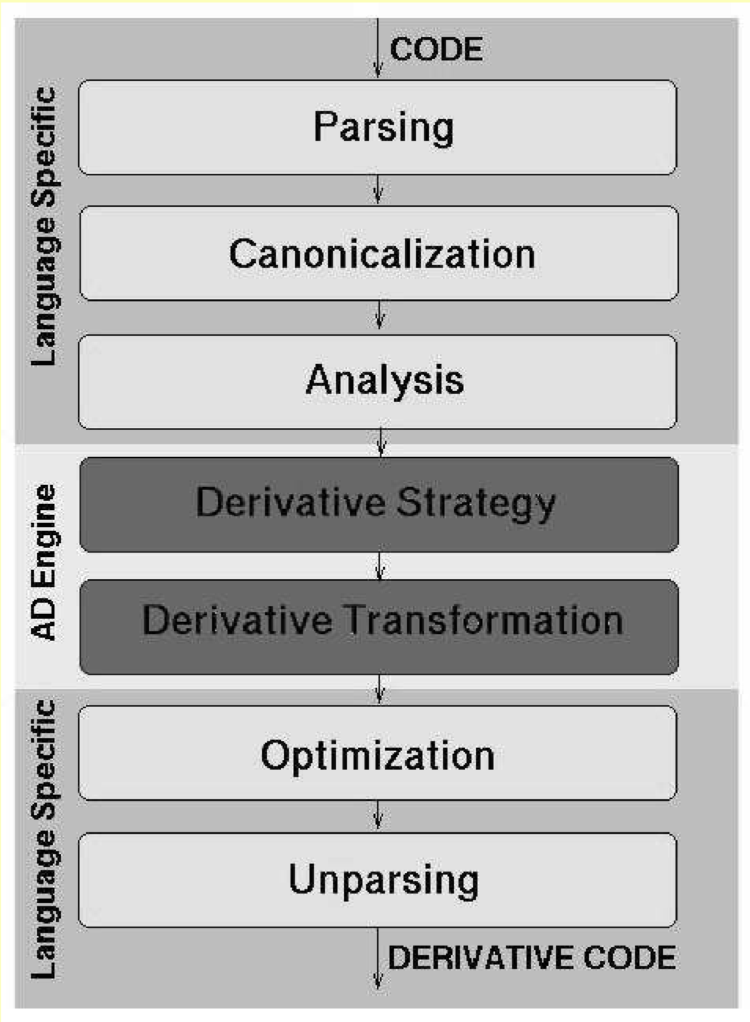
reverse mode

forward mode

function value

```
typedef struct {
  double value;
  double grad[ad_GRAD_MAX];
} DERIV_TYPE
```

AD Tool Architecture



- Shared modules use AIF (AD Interface Format)
  - Unifies "relevant" Fortran and C constructs
  - Extensible annotation system
- Shared modules will be "literate programs"
- Improvement to shared modules benefit all clients

Hessians

- Compute $U^T HV$, diagonals of Hessians and sparse Hessians
- Use shared module with ADIFOR and ADIC
- Best approach is not at all obvious
  - Taylor series vs Hessians, globally and locally
  - Inline code vs procedure library
  - Preaccumulation strategies
- Actual implemented strategy is context-sensitive
  - Based on code-based performance metrics
  - Transparent to frontend
  - AIF transformation written using Sorcerer (T. Parr)
- To be presented at ISSAC 97

Hessians for Nonlinear Structural Analysis

	Time (secs)	Ratio to func eval	Memory (MB)
Function eval	0.6	1	1.6
ADIFOR gradient	7	11.6	22
ADIFOR^2 Hessian	241	401	355
Full Hessian	122	203	188
Diagonal of Hessian	22	36.6	43

15 parameters case

- Joint work w/ Ahmed Noor, U of Virginia, and Larry Green, NASA Langley
- Goal is to use model sensitivities for adaptive evaluation of material properties
- To be presented at USNCCM 97

ADIFOR-MP

- Construct an MPI & PVM aware version of ADIFOR
- Challenges
  - Dependence tracking through send-recv
  - Late binding of message content to send-recv ops
    - derived datatypes
    - absolute addresses and offset arithmetic
  - Communication of derivative data
    - one message or two?
    - prefix receives?
  - Support for reduction operations
    - no opportunity to augment interfaces
- Prototype ADIFOR-MP used for AD + CFD work.

ADJIFOR/ADJIC

- Develop tool to efficiently compute large gradients
- Approach - use code analysis to optimize, and properly place, reverse mode logging and playback
- AIF transformation written in Scheme
- To be presented at 3rd Workshop on Adjoint Applications in Dynamic Meteorology.

	Gradient Evaluation Time (seconds)	Gradient to Function Ratio
Gradient of length 205 by finite differences	1133	206
Gradient of length 205 using ADJIFOR	116	21

Gradient Evaluation Times and Gradient to Function Time Ratios for a 1-D Cloud Model (Provided by Seon Ki Park, CAPS/SOM Univ. of Oklahoma)