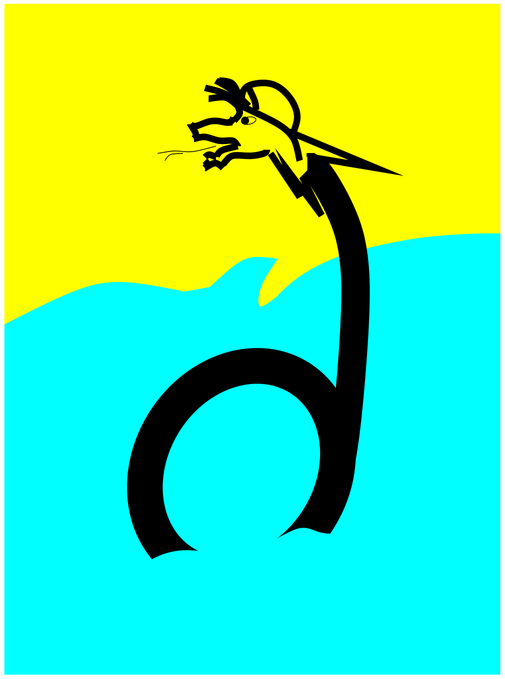
Automatic Differentiation

George F. Corliss, Marquette University, Milwaukee

Collaborators:
Bischof, Carle, Griewank, Hovland, Rall, Chang, ...



Outline of the talk:

- Need derivatives?
- Functionality of automatic differentiation
- Challenges - mathematical
- Challenges - computer science
- Survey field, not tool user manual

A compilation of available automatic differentiation software: `www.mcs.anl.gov/Projects/autodiff/AD_Tools/`

Fields Industrial Optimization Seminar,
University of Toronto, December 7, 2004

Abstract

Automatic differentiation is a technique for providing fast, accurate values of derivative objects (gradients, Hessians, Taylor series) required by modern tools for optimization, nonlinear systems, differential equations, or sensitivity analysis. We outline some needs and applications for derivatives, survey the functionality of AD in forward and reverse modes, and discuss some of the mathematical and computer science challenges of AD.

This talk should be accessible after first semester calculus and a programming course in data structures. It should be interesting to researchers in symbolic computation or in scientific or engineering applications requiring almost any numerical analysis technique requiring derivatives.

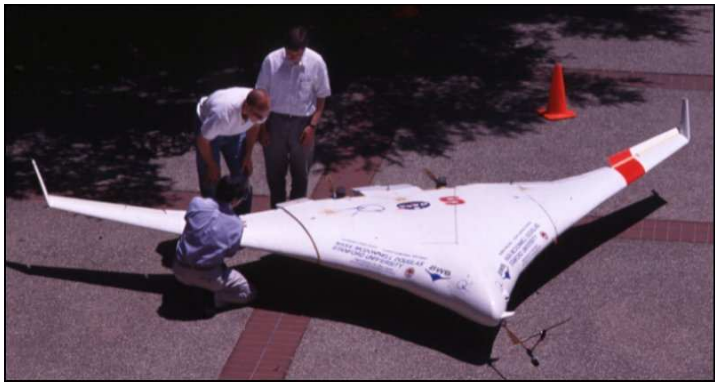
Thunderstorm image: `www.theweatherclub1.homestead.com/tpics.html`

Need derivatives?

Top 10 indicators that AD may be for you:

1. Code requires derivatives
2. Using finite differences
3. Newton's method
4. Interval remainder terms
5. Nonlinear equations
6. Sensitivity
7. Adjoints
8. Optimization
9. Stiff ODE's
10. DAE's

If any of these sound familiar, you are in danger of finding automatic differentiation useful



- Microsoft Excel (via Frontline Systems)
- AMPL (`www.ampl.com`)
- NEOS (`www-neos.mcs.anl.gov`)

all use AD

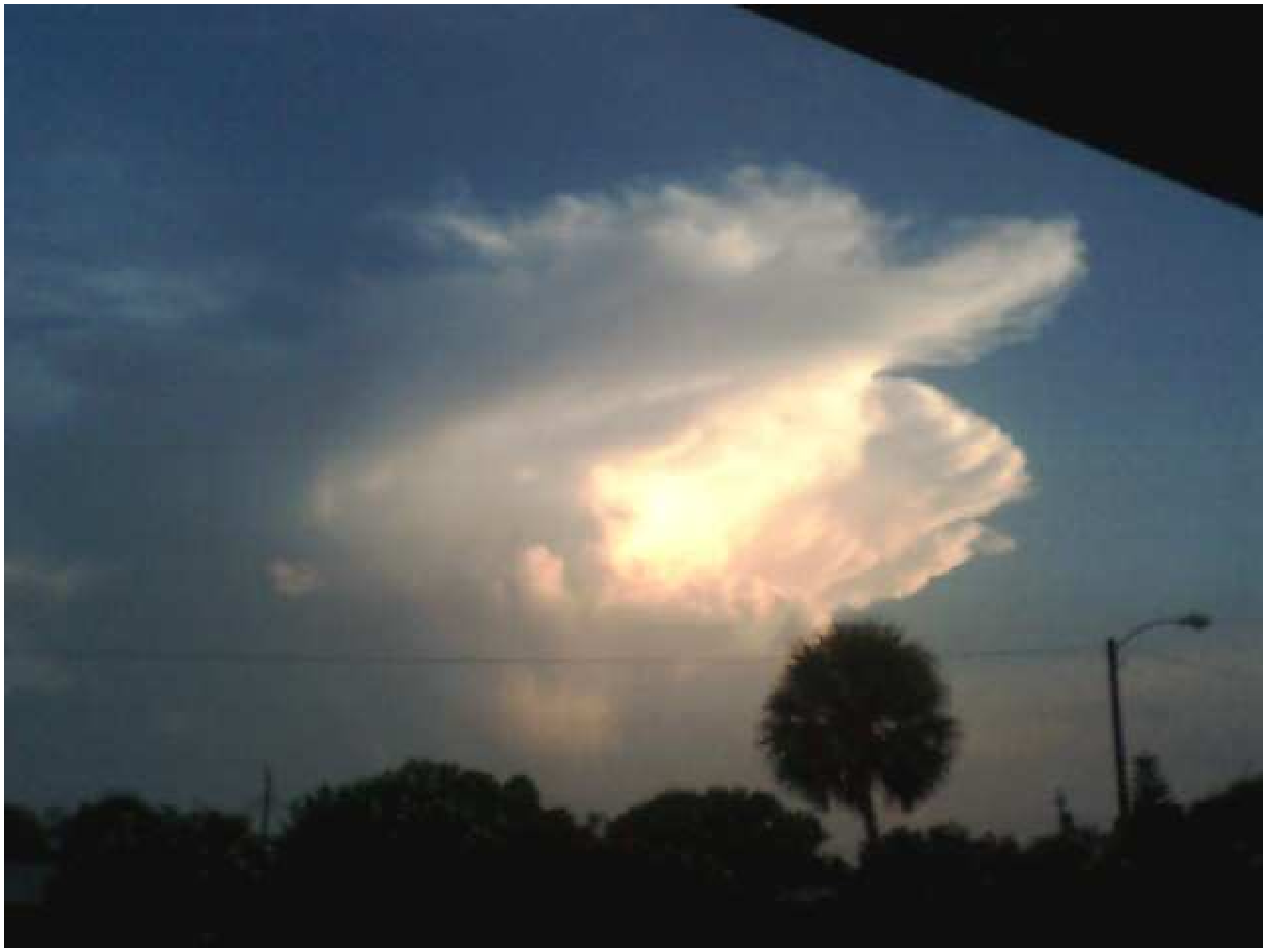
See IMSL library

Photography by Joaquim Martins, `mdolab.utias.utoronto.ca/cgi-bin/ids/index.cgi?mode=album&album=./Aeronautics`

Sensitivities?

Cloud near Melbourn, FL. From `www.theweatherclub1.homestead.com/tpics.html`

Ten minutes later



Differentiation

Automatic/Computational:

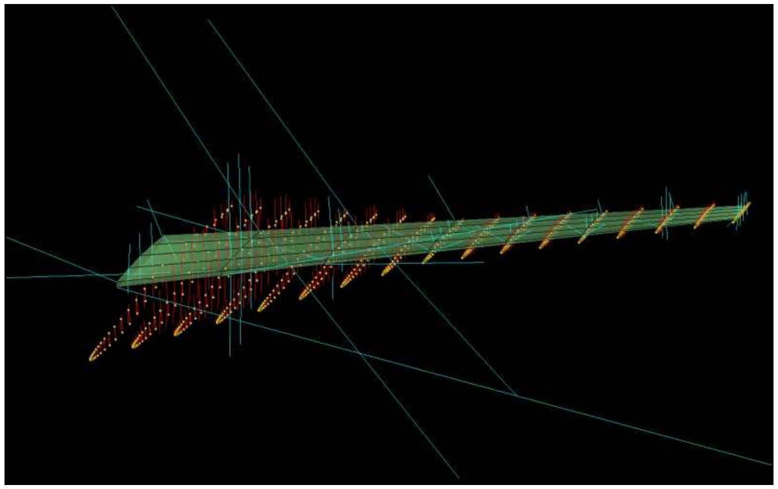
- Propagates values
- Accurate up to round-off

Symbolic:

- Propagates expressions
- Branches?
- Large code?

Numerical:

- Stable?
- $h = ?$



Photography by Joaquim Martins,
`mdolab.utias.utoronto.ca/cgi-bin/ids/index.cgi?mode=album&album=./Aeronautics`

AD Functionality

Given a program for the computation of a function, AD produces a program for evaluating the derivative *exactly* by applying the chain rule to the elementary instructions

$$\begin{tikzcd}
\boxed{\text{Program for } f} \arrow[d, "\text{AD}"] \\
\boxed{\text{Program for } f'}
\end{tikzcd}$$

Source code transformation: ADIFOR, TAMC, Odyssée, Padre2, ...

Operator overloading: ADOL–C, ADOL–F, Lohner's AWA, ...

AD Functionality

AD implements the chain rule

For $f = f(x_1, \ldots, x_m)$, its gradient vector

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_m} \right],$$

is required, for example, by most algorithms for optimization.

Let u and v be functions whose gradients ∇u and ∇v are known or are previously computed, we compute ∇f using the rules

$$\begin{aligned} \nabla(u \pm v) &= \nabla u \pm \nabla v, \\ \nabla(uv) &= u \nabla v + v \nabla u, \\ \nabla(u/v) &= (\nabla u - (u/v)\nabla v)/v, \quad v \neq 0, \end{aligned}$$

for the arithmetic operations and the chain rule

$$\nabla \phi(u) = \phi'(u) \nabla u,$$

for differentiable functions ϕ (such as the standard functions) with known derivatives

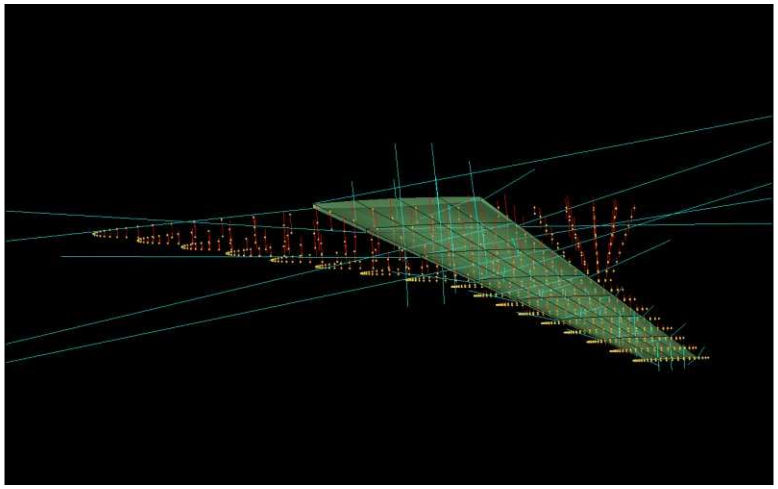
AD Functionality

Assignment statement

$$f(x,y) = (xy + \sin x + 4)(3y^2 + 6),$$

executed as unary & binary operators (code list, computational graph):

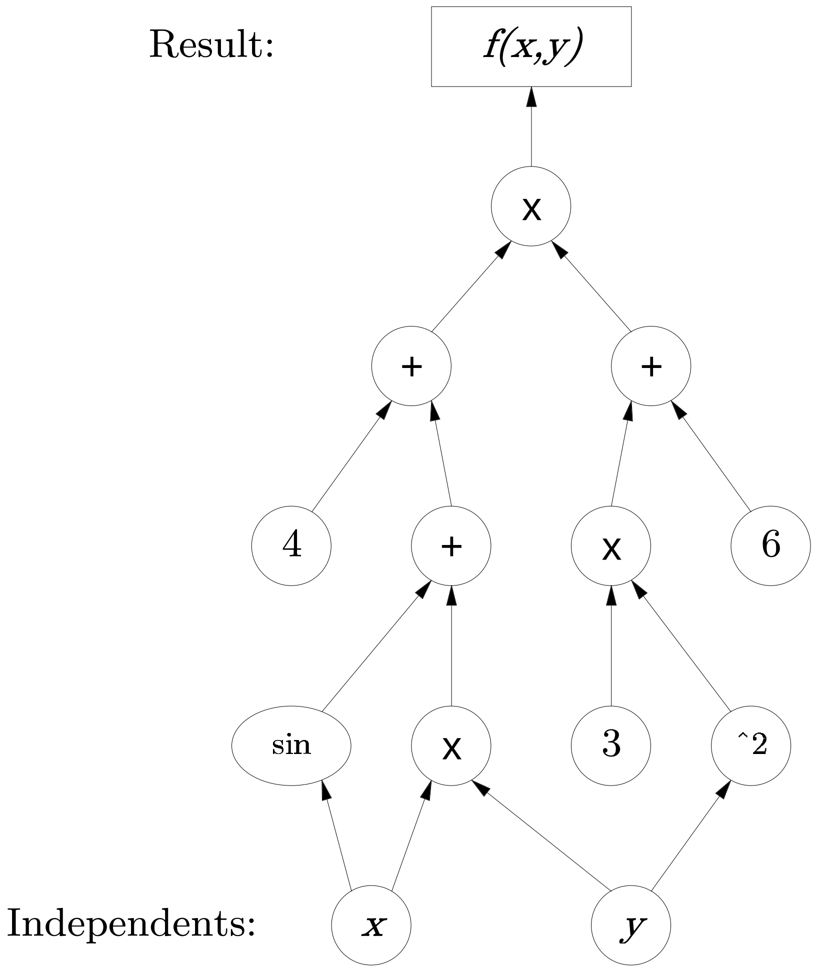
$$\begin{aligned} t_1 &= x & t_6 &= t_5 + 4 \\ t_2 &= y & t_7 &= t_2^2 \\ t_3 &= t_1 t_2 & t_8 &= 3t_7 \\ t_4 &= \sin t_1 & t_9 &= t_8 + 6 \\ t_5 &= t_3 + t_4 & t_{10} &= t_6 t_9 \end{aligned}$$



Derivatives may be evaluated in

- Forward mode
- Reverse mode

Computational Graph: $f(x,y) = (xy + \sin x + 4)(3y^2 + 6)$



AD - Forward Mode

Compute $\dfrac{\partial\ \text{intermediate}}{\partial\ \text{independent}}$

With every variable u associate a derivative object $\nabla\phi u$

In our example, start with $\nabla\phi x$ and $\nabla\phi y$ and compute $\nabla\phi f$:

$$\begin{array}{lllllll}
t_1 & = & x & \nabla t_1 & = & [1,\ 0] & \\
t_2 & = & y & \nabla t_2 & = & [0,\ 1] & \\
t_3 & = & t_1 t_2 & \nabla t_3 & = & t_1\nabla t_2 + t_2\nabla t_1 & \text{yields } [t_2,\ t_1] \\
t_4 & = & \sin t_1 & \nabla t_4 & = & (\cos t_1)\nabla t_1 & \text{yields } [\cos t_1,\ 0] \\
t_5 & = & t_3 + t_4 & \nabla t_5 & = & \nabla t_3 + \nabla t_4 & \text{yields } [t_2 + \cos t_1,\ t_1] \\
t_6 & = & t_5 + 4 & \nabla t_6 & = & \nabla t_5 & \text{yields } [t_2 + \cos t_1,\ t_1] \\
t_7 & = & t_2^2 & \nabla t_7 & = & 2t_2\nabla t_2 & \text{yields } [0,\ 2t_2] \\
t_8 & = & 3t_7 & \nabla t_8 & = & 3\nabla t_7 & \text{yields } [0,\ 6t_2] \\
t_9 & = & t_8 + 6 & \nabla t_9 & = & \nabla t_8 & \text{yields } [0,\ 6t_2] \\
t_{10} & = & t_6 t_9 & \nabla t_{10} & = & t_6\nabla t_9 + t_9\nabla t_6 & \text{yields } [t_9(t_2 + \cos t_1), 6t_2t_6 + t_1t_9]
\end{array}$$

Code for F : scalar operations

Code for $\dfrac{\partial F}{\partial x}$: vector operations

– Parallelize/vectorize

AD - Reverse Mode

Compute $\dfrac{\partial \text{ dependent}}{\partial \text{ intermediate}}$

With every variable u associate an ADJOINT object $\texttt{ubar} = \dfrac{\partial w}{\partial u}$

Key rule for the reverse mode:

$$\boxed{s = f(t,u) \quad \Rightarrow \quad \begin{array}{l} \texttt{tbar += sbar * } \dfrac{\partial f}{\partial t} \\[2ex] \texttt{ubar += sbar * } \dfrac{\partial f}{\partial u} \end{array}}$$

Reverse mode requires the ability to reverse the flow of the computation

AD - Reverse Mode

Assignment: $f(x,y) = (xy + \sin x + 4)(3y^2 + 6)$

In our example, start with $\frac{\partial f}{\partial f} = 1$ and compute $\nabla \phi f$:

$$
\begin{aligned}
\frac{\partial t_{10}}{\partial t_{10}} &= 1 \\
\frac{\partial t_{10}}{\partial t_9} &= t_6 \\
\frac{\partial t_{10}}{\partial t_8} &= \frac{\partial t_{10}}{\partial t_9}\frac{\partial t_9}{\partial t_8} \text{ yields } t_6 \cdot 1 = t_6 \\
\frac{\partial t_{10}}{\partial t_7} &= \frac{\partial t_{10}}{\partial t_8}\frac{\partial t_8}{\partial t_7} \text{ yields } t_6 \cdot 3 = 3t_6 \\
\frac{\partial t_{10}}{\partial t_6} &= t_9 \\
\frac{\partial t_{10}}{\partial t_5} &= \frac{\partial t_{10}}{\partial t_6}\frac{\partial t_6}{\partial t_5} \text{ yields } t_9 \cdot 1 = t_9
\end{aligned}
$$

AD - Reverse Mode (cont.)

$$\frac{\partial t_{10}}{\partial t_4} = \frac{\partial t_{10}}{\partial t_5}\frac{\partial t_5}{\partial t_4} \text{ yields } t_9 \cdot 1 = t_9$$

$$\frac{\partial t_{10}}{\partial t_3} = \frac{\partial t_{10}}{\partial t_5}\frac{\partial t_5}{\partial t_3} \text{ yields } t_9 \cdot 1 = t_9$$

$$\frac{\partial t_{10}}{\partial t_2} = \frac{\partial t_{10}}{\partial t_7}\frac{\partial t_7}{\partial t_2} + \frac{\partial t_{10}}{\partial t_3}\frac{\partial t_3}{\partial t_2} \text{ yields } (3t_6) \cdot (2t_2) + t_9 \cdot t_1 = 6t_2t_6 + t_1t_9$$

$$\frac{\partial t_{10}}{\partial t_1} = \frac{\partial t_{10}}{\partial t_4}\frac{\partial t_4}{\partial t_1} + \frac{\partial t_{10}}{\partial t_3}\frac{\partial t_3}{\partial t_1} \text{ yields } t_9 \cdot \cos t_1 + t_9 \cdot t_2 = t_9(\cos t_1 + t_2)$$

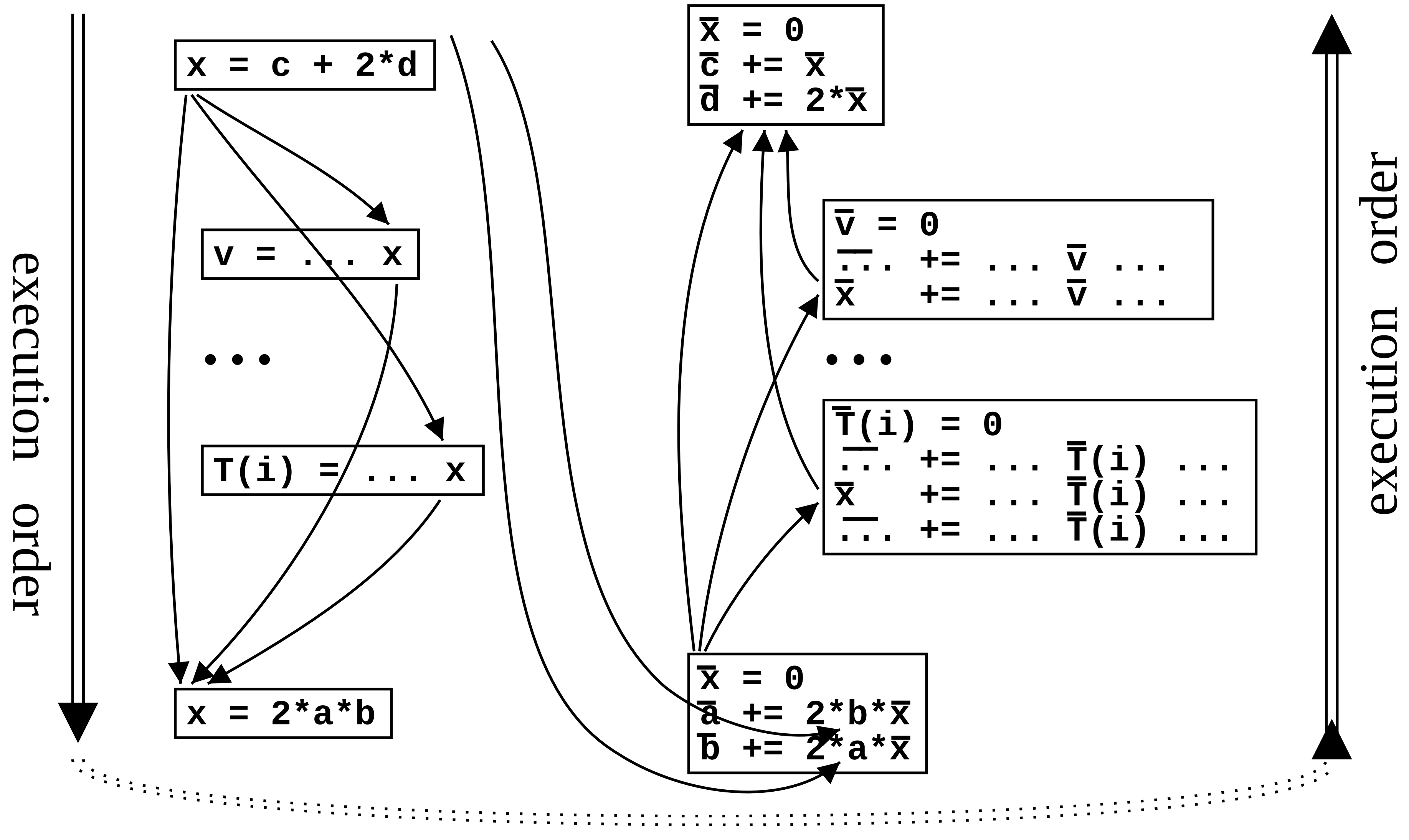
$$\nabla f = \left[\frac{\partial t_{10}}{\partial t_1}, \ \frac{\partial t_{10}}{\partial t_2}\right] = [t_9(\cos t_1 + t_2), \ 6t_2t_6 + t_1t_9]$$

$\nabla\phi$ `f = xbar *` $\nabla\phi$`x + ybar *` $\nabla\phi$`y`

– mostly scalar operations

– one vector parallel loop

Hascoët: Adjoining a Data-Dependence Graph



AD - Hybrid Mode

"Preaccumulation of Local Derivatives"

If $w = f(y, z)$,

$$\nabla\phi w = \frac{\partial w}{\partial y} \cdot \nabla\phi y + \frac{\partial w}{\partial z} \cdot \nabla\phi z$$

Compute "local" derivatives $\frac{\partial w}{\partial y}$ and $\frac{\partial w}{\partial z}$ in the reverse mode

Propagate "global" derivatives $\nabla\phi w$ in the forward mode

Generalizes to second derivatives

Navajo Bridge, Marble Canyon, Arizona, National Information Service for Earthquake Engineering `nisee.berkeley.edu/images/servlet/BrowseGodden?group=GoddenD27.1-3`

Implementation - Source Transformation

E.g., Adifor 2.0 for Fortran [Bischof, Carle, et al.]

Identify top level subroutine:

```
C  File:  squareroot.f
C  Reference: ADIFOR 2.0 User's Guide, section 2.6
C  Figure 2.4.  A Very Simple Subroutine.

      subroutine squareroot (x, y)
      real x, y
      y = sqrt (x)
      end
```

Implementation - ADIFOR-generated code:

```
subroutine g_squareroot(g_p_, x, g_x, ldg_x, y, g_y, ldg_y)
  real x, y
  integer g_pmax_
  parameter (g_pmax_ = 1)
  integer g_i_, g_p_, ldg_y, ldg_x
  real r1_p, r2_v, g_y(ldg_y), g_x(ldg_x)
  integer g_ehfid
  data g_ehfid /0/
  call ehsfid(g_ehfid, 'squareroot','g_squareroot.f')
  if (g_p_ .gt. g_pmax_) then
    print *, 'Parameter g_p_ is greater than g_pmax_'
    stop
  endif
  r2_v = sqrt(x)
  if ( x .gt. 0.0e0 ) then
     r1_p = 1.0e0 / (2.0e0 *  r2_v)
  else
     call ehufSO (9,x, r2_v, r1_p, g_ehfid, 42)
  endif
  do g_i_ = 1, g_p_
    g_y(g_i_) = r1_p * g_x(g_i_)
  enddo
  y = r2_v
end
```

Implementation - Source Transformation

New main program (or modified applications algorithm):

- allocate variable storage
- call ADIFOR-generated code
- use the derivative

```
program driver
real x, y
real g_x(1), g_y(1)
print *, ' Enter x: '
read *, x
g_x(1) = 1.0
call g_squareroot (1, x, g_x, 1, y, g_y, 1)
call ehrpt
print *, y
print *, g_y(1)
end
```

Implementation - Operator Overloading

E.g., ADOL-C [Griewank, et al.]

`class adouble { ... }`

Features include

- Forward or reverse mode
- Partial derivatives or Taylor series
- Record computations on a "tape"
- Jacobians with structure
- Implicit functions
- ODE tools – implicit Taylor series/Padé
- "Checkpointing"

Spiral bevel gears, Emerson Power Transmission Company,

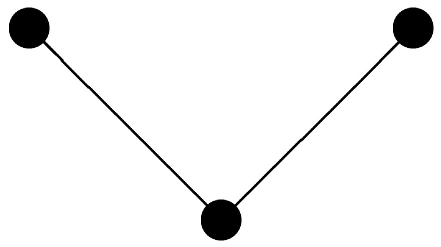
`science.howstuffworks.com/gear4.htm`

Math - Not differentiable?

f is defined, but f' does not exist?

- E.g., $f(x) = \|x\|$; $f(x) = \max\{g(x), h(x)\}$
- Very important – norms, scaling

Automatic differentiation differentiated the code:



if $x < 0$ then $f = -x$; $f' = -x'$
else $f = x$; $f' = x'$

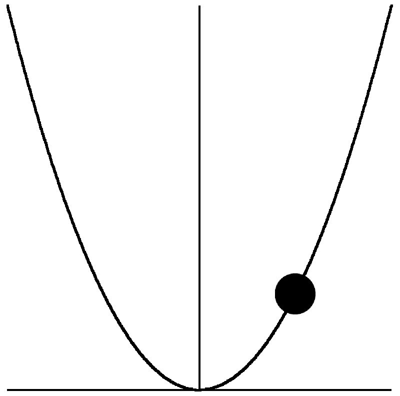
Yields $\|0\|' = 1 * x'$

Elementary functions are supported
Issue here is how to handle user's code

Warning: You should know what you are doing!

Math - Special cases

f is
defined by a special case? E.g., $f(x) = x^2$ [Fischer]



if $x = 1$ then $f = 1$; $f' = 0$
else $f = x^2$; $f' = 2 * x * x'$

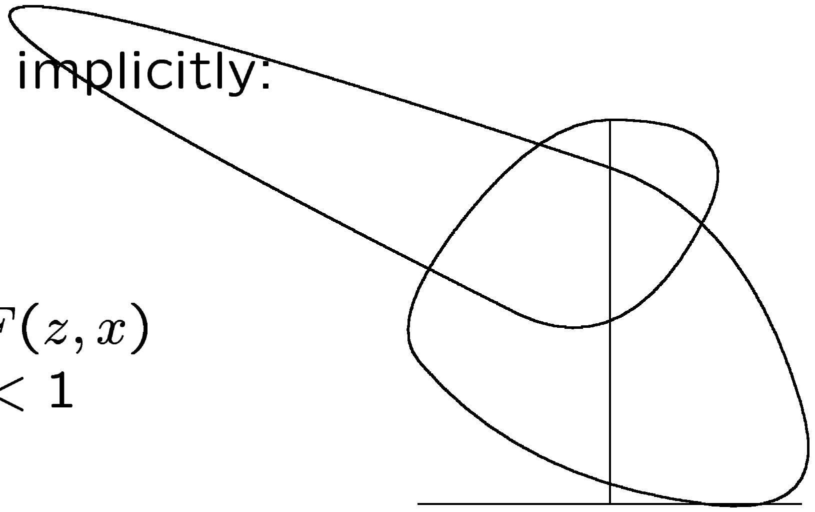
You would *never* code it that way???

Many codes have hard-coded special cases

Automatic differentiation differentiates your code,
NOT your intentions

If Interior (Closure (Domain)) $\neq \emptyset$, then derivatives computed $=$ limits

Math - Implicit functions



Original algorithm: Function is defined implicitly:

$F(z, x) = 0$ defines $z_*(x)$ and $z'_*(x)$

Precondition:	$P_k \cdot F(z, x) = 0$
Iterate:	$z_{k+1} = z_k - P_k \cdot F(z, x)$
Converges when	$\lvert\lvert I - P_k \cdot F_z \rvert\rvert \le \rho < 1$

e.g., Newton's method

Strategy 1: Black box

Apply AD

Usually works

May need 1 or more extra iterations

Can fail

Math - Implicit functions

Strategy 2: White box

Iterate function $z(x)$ to convergence

– Tests based on $F(z, x)$

Then turn on AD and iterate for $z'(x)$

– Tests based on $F_z z' + F_x$

– Often, one iteration suffices

Do not need to differentiate the entire iteration

Example: NASA

– Input: 6 - 100 wing shape parameters

– Generate initial 3-D finite element grid

– Iterate to steady-state pressure distribution solving PDE with multi-grid

– Vary parameters to optimize design (uses Newton)

– f' cost only a few % additional!

Math - Implicit functions

Strategy 3: Use some math - Derivative z'_*

Differentiate to get:	$F_z z' + F_x = 0$
Precondition:	$P_k \cdot (F_z z' + F_x) = 0$
Iterate:	$z'_{k+1} = z'_k - P_k \cdot (F_z z' + F_x)$
Converges when	$\|\|I - P_k \cdot F_z\|\| \leq \rho < 1$

Same condition!

A little math beats a lot of computing

Work of Gilbert, Griewank, Christianson, et al.

Computer Science - Compiler technologies

Source transformation and operator overloading stretch compilers

Adifor was built on Ken Kennedy, et al. (Rice) Parascope tools for High Performance Fortran

- Analysis and annotation of computational graph
- Interprocedural dependency analysis
- Common subexpression removal
- Code pruning (unused code)
- Parallel scheduling?

Sun and NAG have let it be known that their Fortran compilers will soon have an -AD option

Computer Science - Checkpointing

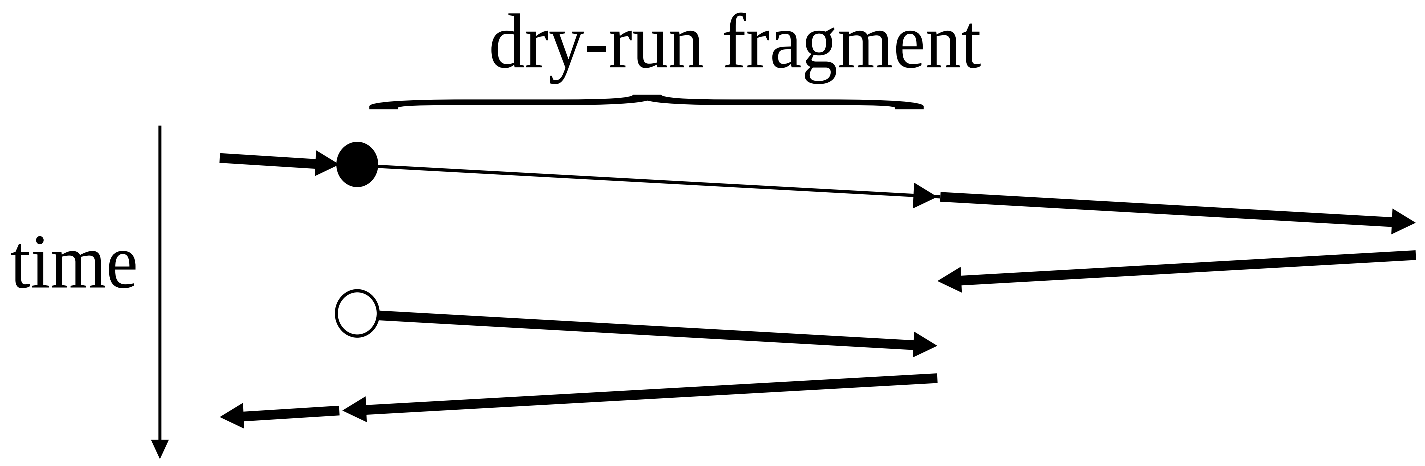
Reverse mode (or multiple passes of forward) records execution trace (potentially enormous)

- Record: Operation, operands [result]
- Implies storage proportional to run time

Solution: Checkpointing, as used for execution recovery

Requires memory to store 1 checkpoints (this example)
Requires 2 forward passes of same code (this example)
Requires 1/2 as much tape storage

Vs. "Dry-run" (Hascoët)



Computer Science - Complexity

Finite difference approximation to $\nabla \phi f$ requires $n+1$ evaluations of f

Forward mode AD costs about 2-3 $n \times$ Cost(f)

- Why? $f = u * v \Rightarrow f' = u * v' + u' * v$ has two $*$

Reverse mode AD costs about 5 $\times$ Cost(f) **independent** of n

$f : R^n \longrightarrow R^m$ rules of thumb:

- $n >> m$ (e.g., $m = 1$) prefer reverse
- $n << m$ (e.g., $n = 1$) prefer forward

See computational graph node elimination

Computer Science - Complexity

Tadjoudine, Forth, and Pryce:

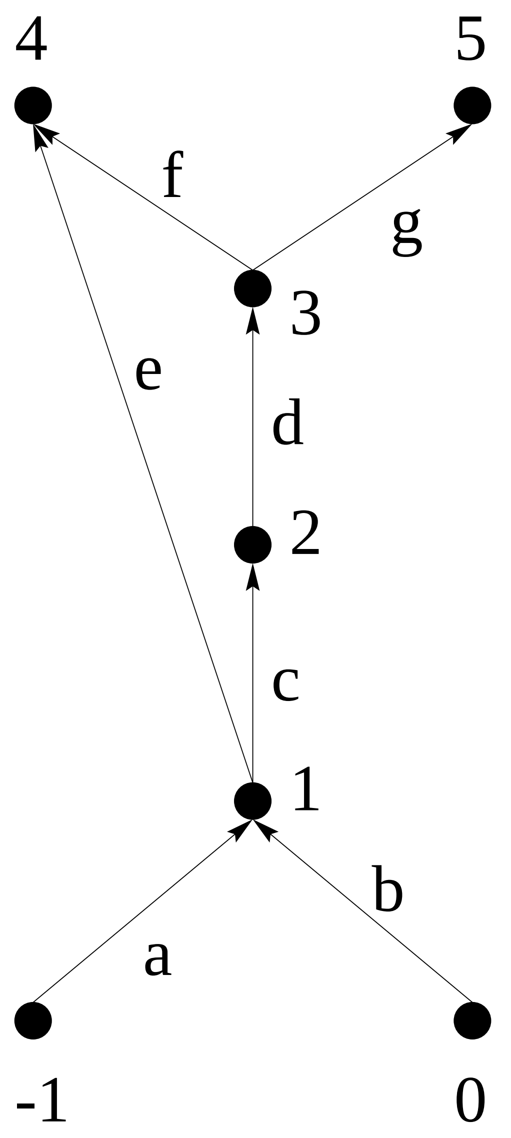
Roe's numerical flux (flow fields with moderate to strong shocks)

Method	time (∇F)	$\frac{\text{time } (\nabla F)}{\text{time } (F)}$	Deviation vs ADIFOR
Finite difference, 1-sided	0.12	10.64	4.34E-07
ADIFOR	0.15	13.37	—
TAMC (Forward)	0.13	11.94	4.66E-15
TAMC (Reverse)	0.11	10.28	5.77E-15
AD01 (Forward)	1.55	134.68	7.99E-15
AD01 (Reverse)	0.95	82.90	4.88E-15

CPU timings (in seconds) on SGI IRIX64 IP27

Computer Science - Graph node elimination

$$\begin{aligned} h_1 &= x_1 \cdot x_2 \\ h_2 &= \exp(\sin(h_1)) \\ y_1 &= h_1 \cdot h_2 \\ y_2 &= \cos(h_2) \end{aligned}$$



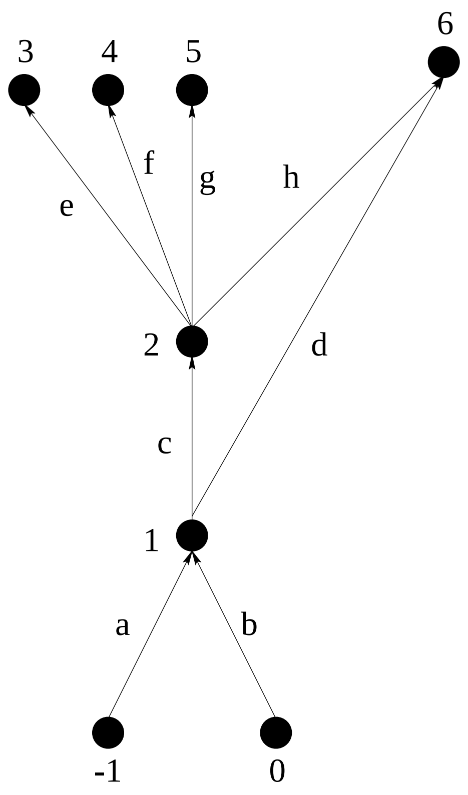
Forward

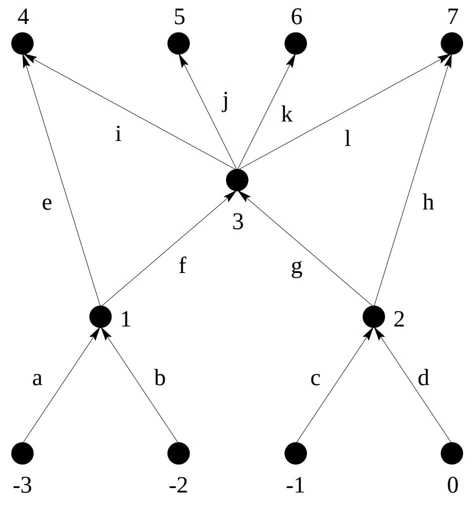
Reverse

Hybrid? Minimize number of *

Markowitz [Griewank, Naumann]

Computer Science - Graph node elimination



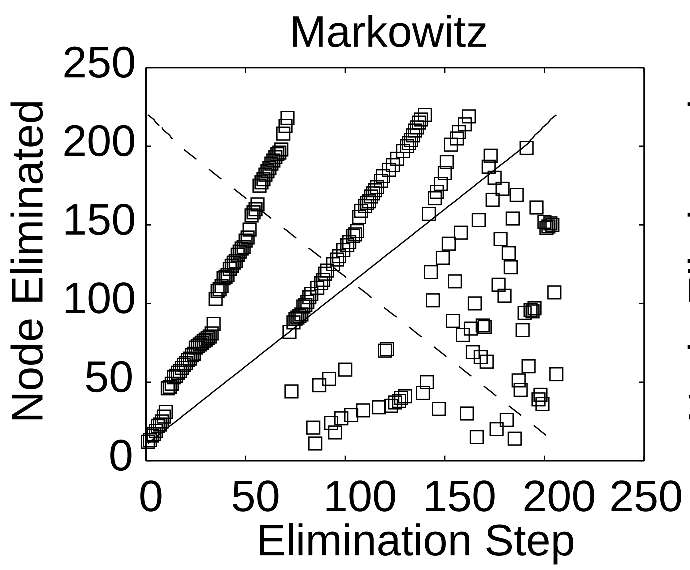


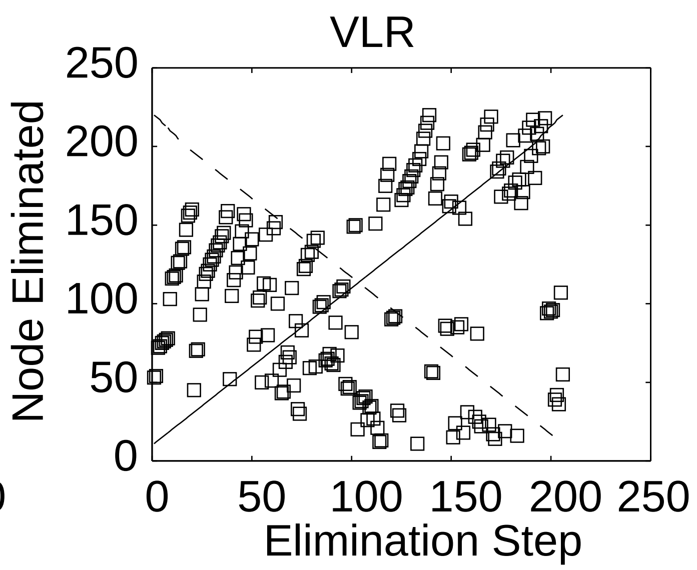
Naumann: "Lion" and "bat" graphs

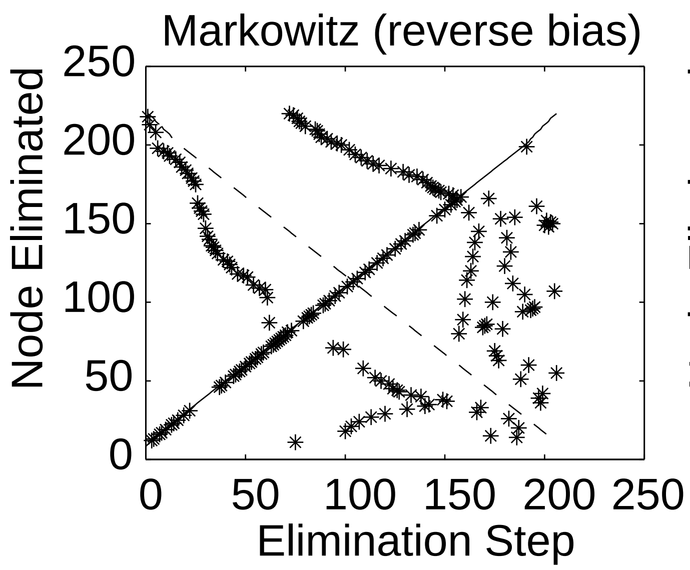
Neither vertex nor edge elimination solves the general "Optimal Jacobian Accumulation" problem

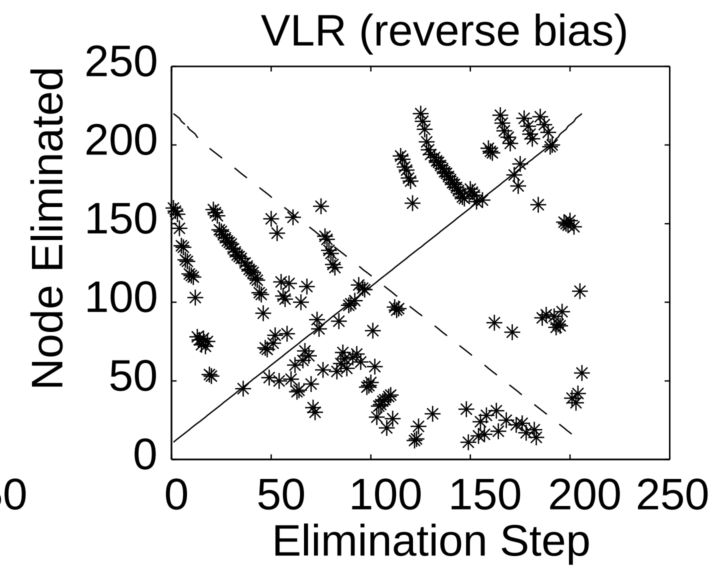
Elimination sequence

Different strategies applied to Roe









Applications

See *Computational Differentiation*, SIAM, 1996; and *Automatic Differentiation*, Springer, 2001

acoustic wave equation	aerofoil optimization	chemical reaction
circuit simulation	computational aerodynamics	computational fluid dynamics
computational quantum chemistry	cooling system	coupled pendula
data assimilation	design optimization	disordered condensed matter
diurnal kinetics advection-diffusion	elastic-plastic torsion	electromagnetic problem
electron paramagnetic resonance	Euler equations	gear tooth contact
hypoid bevel gears	low thrust orbit transfer	material modelling
minimum time orbit transfer	model fitting	model structure validation
MOS transistor model	multibody dynamics	Navier-Stokes solver
near-earth asteroids	neutron scattering	nonlinear least squares
nonlinear regression	nonlinear solver	North Sea herring
observer design	ODE solver	optimal control
parameter identification	parametric uncertainty analysis	partial differential equations
periodic functions	power plant model	race car performance
radiation diffusion	rational approximation	sea ice
sensitivity analysis	structural analysis	structural mechanics
terrain modelling	thermal-hydraulic	vehicle dynamics

Conclusions

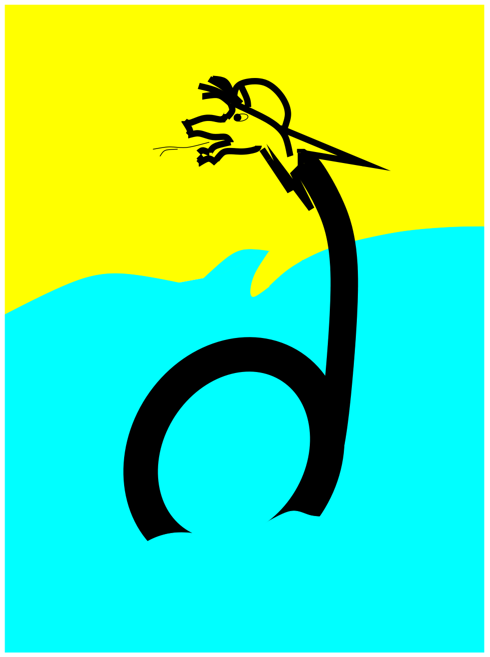
Derivatives can be computed
– efficiently
– accurately
– easily

Users can be expected to provide for library codes

Algorithm designers need not avoid derivatives

Black box users
– Minimize programmer and user effort
– Gets the right answer competitively with FD

Wizards
– Willing to apply knowledge of problem
– Get the right answer FAST



George Corliss, George.Corliss@Marquette.edu

AD dragon - logo for AD2000, Automatic Differentiation: From Simulation to Optimization, Nice, France, June 2000