

# Enhancing Efficiency Through User Assistance

Short Course at SIAM Optimization Conference

Andreas Griewank  
Technical University Dresden

SIAM Optimization, Toronto 2002

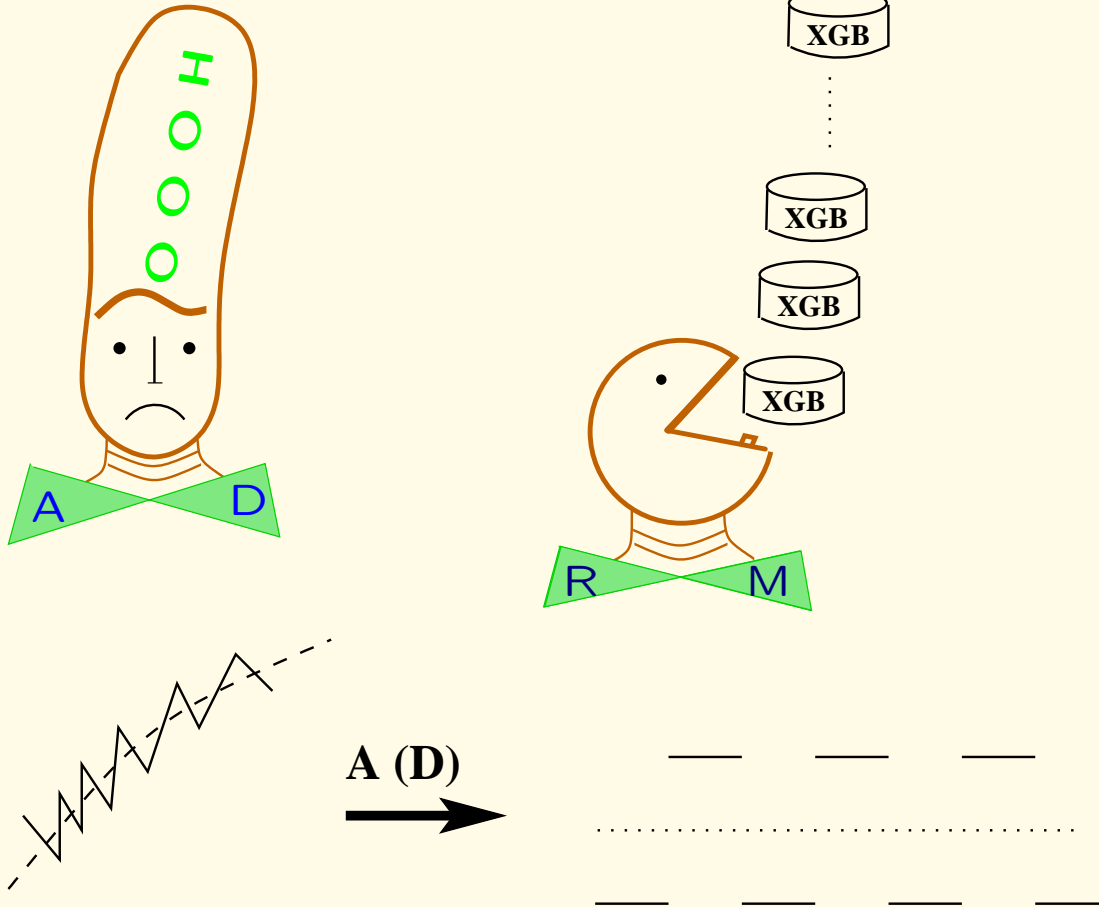
## Table of Contents

- Potential Trouble Spots.
- General Advice on Code Preparation.
- Matrix Compression by CPR or Newsam/Ramsdell.
- Checkpointing for proper Evolutions.
- Piggyback Differentiation for Fixed Point Iterations.

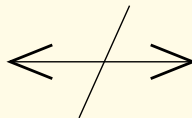
# Piggyback Differentiation and Optimization

## Contents:

- Motivation / History
- Basic Forward and Reverse
- Typical Design Szenario
- Piggy Back Differentiation
- Conclusion w.r.t. Optimization



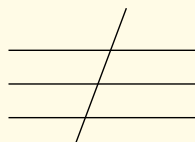
COMMON  $A(7, 5)$



COMMON  $I(35)$

$$y = \sin(x)$$

$$z = \sin(x)$$



$$y = \sin(x)$$

$$z = y$$

## Warning 1

1. In contrast to CA systems current AD tools DO NOT perform algebraic simplifications.
2. Peephole optimization and vectorization MAY suffer due to insertion of derivative statements (calls).
3. Compiler optimization WILL suffer on account of the introduction of user defined types with global side effects.

## Remedy 1

- a) Write tight efficient source code.
- b) Evaluate several or complex derivatives.
- c) Finance an integrated SC-AD tool.

## Potential Trouble Spots

- General Branches:

if  $a > 0$  then ... else ...

- Conditional Assignment:

$v = a ? b : c$

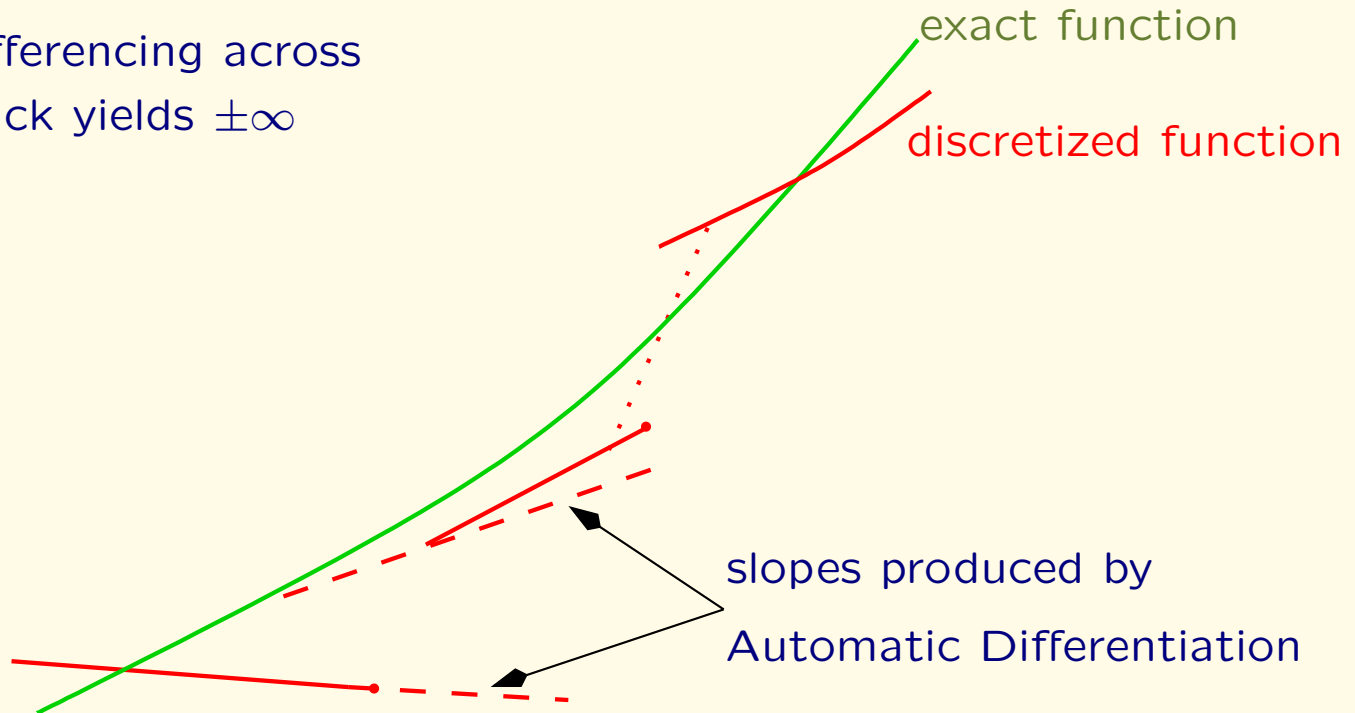
- Absolute values:

$g(x) = f(x) + \mu \max\{0, c(x)\}$

- Scaling by  $l_1$  or  $l_\infty$  norms.

## Piecewise smooth Case

Differencing across  
crack yields  $\pm\infty$



Generally, Automatic Differentiation yields the derivatives of the smooth piece containing the current evaluation point.

## Exception

$$y = 0$$

$$\text{IF } (x.\text{NE}.0.0) \ y = (1 - \text{DCOS}(x))/x$$

$$\Rightarrow \left. \frac{\partial y}{\partial x} \right|_0 = 0 \neq \frac{1}{2} = \text{Correct answer.}$$

**Moral:** Avoid tests for exact equality.

## Warning 2

Don't assume the results of your code are continuous functions of the inputs.

## Remedy 2

- Provide code with preferred direction to differentiate in.
- Avoid composition of singularities

$$\begin{array}{ccc}
 \text{😊} & |u| = \sqrt{u \cdot u} & \text{😡}
 \end{array}$$

- Avoid unnecessary branches

$$\begin{array}{ccc}
 \text{😊} & \max(u, w) = u > w : u \text{ ? } v & \text{😡}
 \end{array}$$

- Make differentiation and discretization (nearly) commutative!!!

### Warning 3

AD-Tools don't/can't yet exploit problem structure automatically

### Remedy 3

- a) Separate active from passive code and data as much as possible.
- b) Differentiate all active code together and utilize all common intermediates.
- c) Exploit Sparsity by Matrix Compression.

## Row-Compression/Reconstruction

$$\begin{array}{c}
 m \left\{ \begin{array}{c} \left[ \begin{array}{c} \dots \\ B \\ \times \times \times \dots \times \end{array} \right] \\ \underbrace{\hspace{10em}}_p \end{array} \right. \leftarrow \equiv \left[ \begin{array}{c} \dots \\ F'(x) \\ 0 \times 00 \dots 00 \times \times 0 \end{array} \right] \underbrace{\hspace{10em}}_n \left. \begin{array}{c} \left[ \begin{array}{c} \dots \\ S \\ \dots \\ \dots \end{array} \right] \right\}_p \left. \vphantom{\left[ \begin{array}{c} \dots \\ F'(x) \\ 0 \times 00 \dots 00 \times \times 0 \end{array} \right]} \right\}_n
 \end{array}$$

CM: NR:

$$\begin{bmatrix} a & b & 0 & 0 \\ c & 0 & d & 0 \\ e & 0 & 0 & f \\ 0 & 0 & g & h \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & | & 1 & -1 \\ 0 & 1 & 0 & | & 1 & 0 \\ 0 & 1 & 0 & | & 1 & 0 \\ 0 & 0 & 1 & | & 1 & 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 & | & a+b & -a \\ c & d & 0 & | & c+d & -c \\ e & 0 & f & | & e+f & f-e \\ 0 & g & h & | & g+h & h \end{bmatrix}$$

- **Reconstruction**

$$B \mapsto F'(x) \quad \text{need} \quad p \geq \hat{n} \equiv \max_{1 \leq i \leq m} \{ \# \text{nonzeros}(\nabla F_i) \}$$

which is sufficient for generic  $S \in \mathbb{R}^{n \times p}$

- **Transpose = Column/Reconstruction**

$$C = W F'(x) \in \mathbb{R}^{q \times n} \mapsto F'(x)$$

possible for generic  $W \in \mathbb{R}^{q \times n}$  with

$$q \geq \hat{m} \equiv \max_{1 \leq j \leq n} \{ \# \text{nonzeros}(\partial F / \partial x_j) \}$$

- **Twosided Compression:**

$$B = F' S, \quad C = W F'(x) \mapsto F'$$

**Jacobian with  $\bar{n} \leq \hat{n} \leq \chi(\mathcal{G}_c) \leq n$  and  $\bar{m} \leq \hat{m} \leq \chi(\mathcal{G}_r) \leq m$**

| Sparse            |                  | NR compressed     |                  |                               | CPR compressed    |                  |                       |
|-------------------|------------------|-------------------|------------------|-------------------------------|-------------------|------------------|-----------------------|
| $\longrightarrow$ | $\longleftarrow$ | $\longrightarrow$ | $\longleftarrow$ | $\longleftrightarrow$         | $\longrightarrow$ | $\longleftarrow$ | $\longleftrightarrow$ |
| $\bar{n}$         | $\bar{m}$        | $\hat{n}$         | $\hat{m}$        | $\leq \min(\hat{n}, \hat{m})$ | $\chi(A)$         | $\chi(A^T)$      | ??                    |

## Combined Column and Row Compression

$$F'(x) = A \equiv \left[ \begin{array}{c|ccc} \delta_1 & \alpha_2 & & \alpha_{n-1} & \alpha_n \\ \hline \beta_2 & \delta_2 & & 0 & 0 \\ & & \ddots & & \\ \beta_{n-1} & 0 & & \delta_{n-1} & 0 \\ \beta_n & 0 & & 0 & \delta_n \end{array} \right]$$

$$A \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} \sum_{i=2}^n \alpha_i & \delta_1 \\ \delta_2 & \beta_2 \\ \vdots & \vdots \\ \delta_{n-1} & \beta_{n-1} \\ \delta_n & \beta_n \end{bmatrix} \quad \text{and} \quad A^T \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \alpha_2 \\ \vdots \\ \alpha_{n-1} \\ \alpha_n \end{bmatrix}$$

A combination of two forward Jacobian-vector products and one reverse vector-Jacobian product yields the complete Jacobian, whose reconstruction would require  $n$  derivative vectors if either pure column or pure row compression was applied.



## Remedy 3 continued

- d) Choose **forward**  $\longleftrightarrow$  **reverse** by comparing the maximal number of nonzeros per row and column rather than the full dimensions.
- e) Evaluate only what you really need !!!
- f) Exploit Partial Separability by splitting off into separate (in)dependents
- linear combinations of independents,
  - linear contributions to dependents.

## Rank 1 Example

$$F(x) = x + b \sin(a^T x) \quad \text{for } b, a \in \mathbb{R}^n$$

$$\implies OPS(F'(x)) = OPS(I + b \cos(a^T x) a^T) = n^2$$

$$\implies \# \text{nonzeros per row and column} = n$$

but with  $\xi \equiv a^T x$  independent and  $\sin(\xi)$  dependent.

$$\tilde{F}(x, \xi) = (x^T, \sin(\xi))^T : \mathbb{R}^{n+1} \rightarrow \mathbb{R}^{n+1}$$

$$\implies OPS(\tilde{F}') = OPS(\cos(\xi)) = \mathcal{O}(n^0)$$

Newton-step by Sherman-Morrison cheap.

## Warning 4

$$z = g(x)$$

for  $t = 1, 2 \dots l$

$$z = G_t(z)$$

$$y = Qz$$

implies in basic reverse

$$MEM(x, \bar{y} \rightarrow \bar{x}) \sim l \sim OPS(x \rightarrow y)$$

## Remedy 4

- a) Use checkpointing if trajectory of  $z$  values matters.
- b) Reverse instantaneously if loop commutes (at least asymptotically).



## Theoretical Results

Let  $l$  be the total number of time steps to reverse with up to  $c$  checkpoints at any time and  $r$  the unique integer satisfying

$$\beta(c, r - 1) < l \leq \beta(c, r) \equiv \binom{c+r}{c}.$$

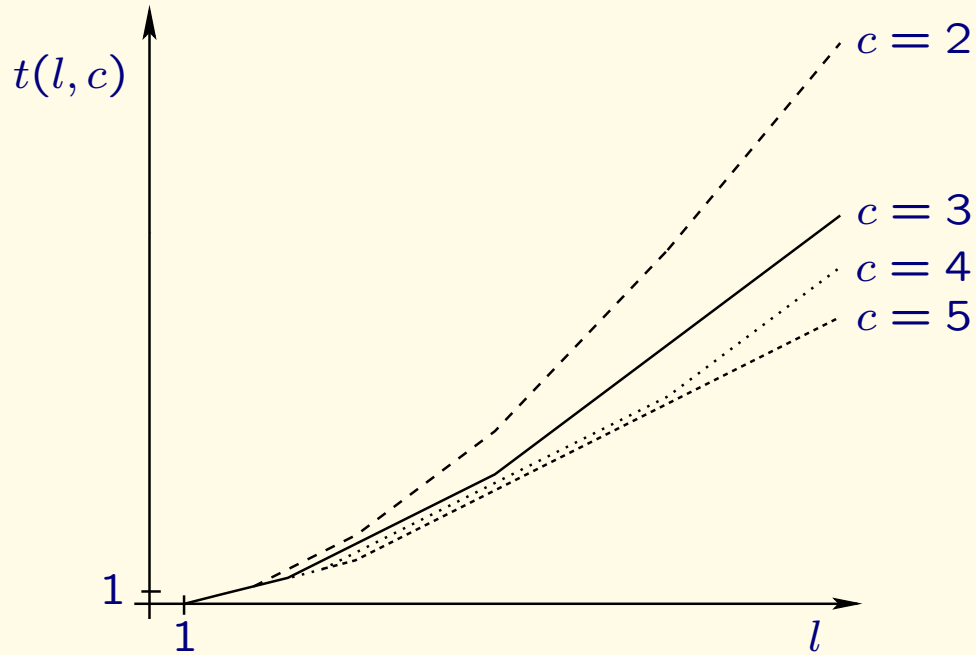
Then the minimal number of forward steps is given by

$$t(l, c) = rl - \beta(c + 1, r - 1).$$

and the minimal number of checkpoint settings equals

$$q(l, c) = \begin{cases} \beta(c - 1, r - 1) & l \leq \beta(c, r - 1) + \beta(c - 1, r - 1) \\ l - \beta(c, r - 1) & l \geq \beta(c, r - 1) + \beta(c - 1, r - 1) \end{cases}$$

## Development of Additional Cost



Minimal number  $t(l, c)$  of time steps

## Source code example

```
capo = 0,   fine = capo + steps,   check = -1
do
  oldcapo = capo
  whatodo = revolve(check, capo, fine, snaps, info)
  switch(whatodo)
    case advance: for oldcapo ≤ i < capo
                  forward(y)
    case takeshot: store(y, ystor, check)
    case firsturn: forwardrec(y)
                  init(bz, by)
                  reverse(by, bz)
    case youturn:  forwardrec(y)
                  reverse(by, bz)
    case restore:  restore(y, ystor, check)
    case error:    printf( "schedule error!" )
  end switch
while((whatodo <> terminate) && (whatodo <> error))
```

## Numerical example: Flow simulation

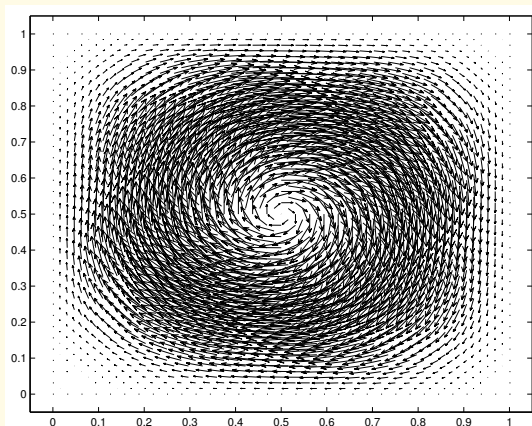
(joint work with Michael Hinze)

### Model problem: Cavity flow

Cost function of tracking type, initial state:

$$y_0(x) = e \begin{bmatrix} (\cos 2\pi x_1 - 1) \sin 2\pi x_2 \\ -(\cos 2\pi x_2 - 1) \sin 2\pi x_1 \end{bmatrix},$$

$\Omega := (0, 1) \times (0, 1)$ ,  $T = 1$ ,  $\text{Re} = 10$ ,  $e = \text{Euler number}$



Desired state:

$$z(x, t) = \begin{bmatrix} \varphi_{x_2}(x, t) \\ -\varphi_{x_1}(x, t) \end{bmatrix},$$

$$\varphi(x, t) = \theta(x_1, t)\theta(x_2, t),$$

$$\theta(y, t) = (1 - y)^2(1 - \cos kt),$$

$$y \in [0, 1].$$

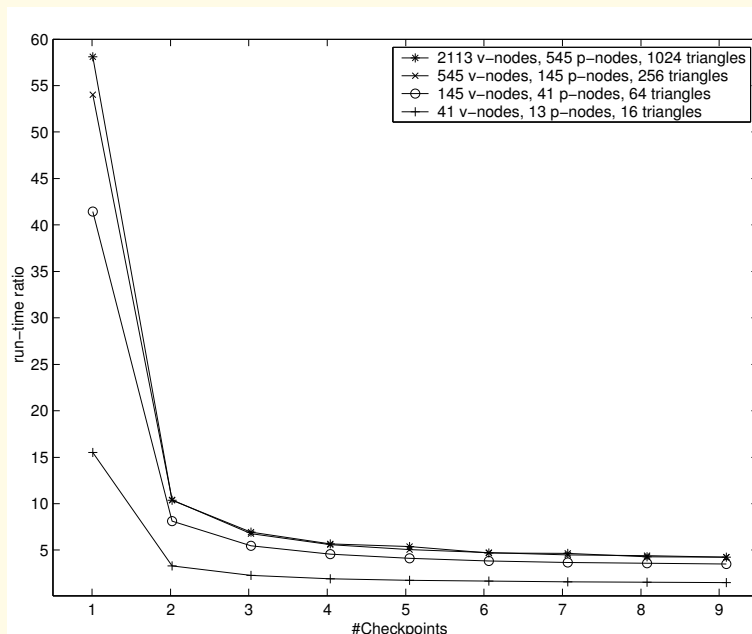
## Memory Requirement

- Storage of full state information:  
2113 velocity nodes, 545 pressure nodes, and  
100 time steps  $\approx$  3.8 MByte  
 $\Rightarrow$  380 MByte for 10.000 time steps
- Checkpointing: 38 kByte / Checkpoint

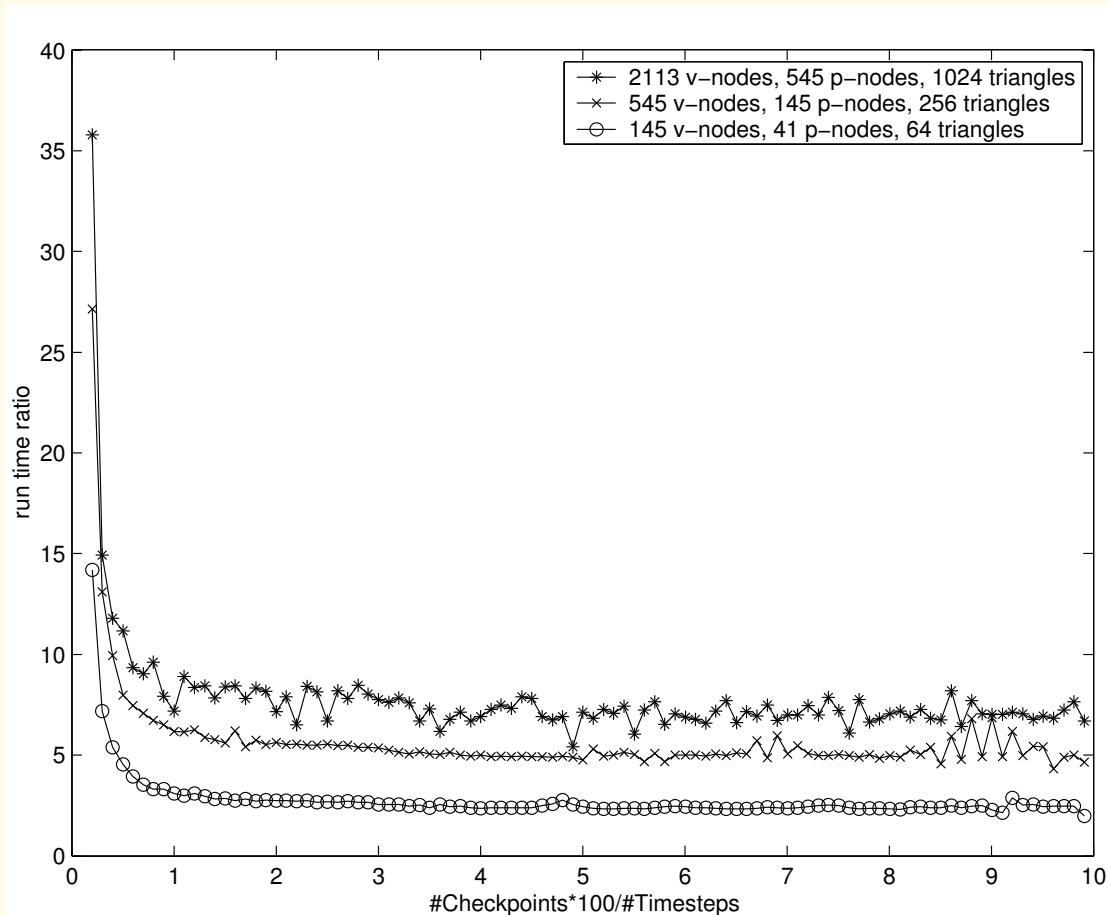
Requirements for 2D model!

For 3D model:  
Storage of full state information becomes difficult!

## Run-times, 100 Time Steps



- Run-time ratio between 2 and 5
- Slow down factor up to 2 compared to full state storage
- But: Only a very small number of checkpoints required  
 5 Checkpoints = 190 kByte  $\iff$  3.8 MByte !!



## Typical Design Scenario

### State Equation:

$$w \equiv F(z, x) = 0 \quad \text{with} \quad F : \mathbb{R}^l \times \mathbb{R}^n \mapsto \mathbb{R}^l$$

state variables

design variables

### Response Function:

$$y = f(z, x) \quad \text{with} \quad f : \mathbb{R}^l \times \mathbb{R}^n \mapsto \mathbb{R}^m$$

where possibly  $l \sim h^{-3}$ ,  $n \sim h^{-2}$  or  $h^{-1}$ ,  $m \sim h^0$ .

## Typical

# Computation of Coleman–Verma Partition

For Given  $p$ :

Read  $p$  and initialize  $q$

For  $s = 1, 2, \dots$

1. Combine some rows of  $A$  with no more than  $p$  nonzero entries into the matrix  $A_s^r$  and remove it from  $A$ .
2. Stop if  $A$  is empty.
3. If necessary, increment  $q$  until at least one column of  $A$  has no more than  $q$  entries.
4. Combine some columns of  $A$  with no more than  $q$  nonzero entries into the matrix  $A_s^c$  and remove it from  $A$ .

## Consequences of the Implicit Function Theorem

Implicit Derivative for fixed  $w = F(z_*(x), x) = 0$ :

$$\frac{dz}{dx} \equiv - \left( \frac{\partial w}{\partial z} \right)^{-1} \frac{\partial w}{\partial x} = -F'(z_*, x)^{-1} F_x(z_*, x)$$

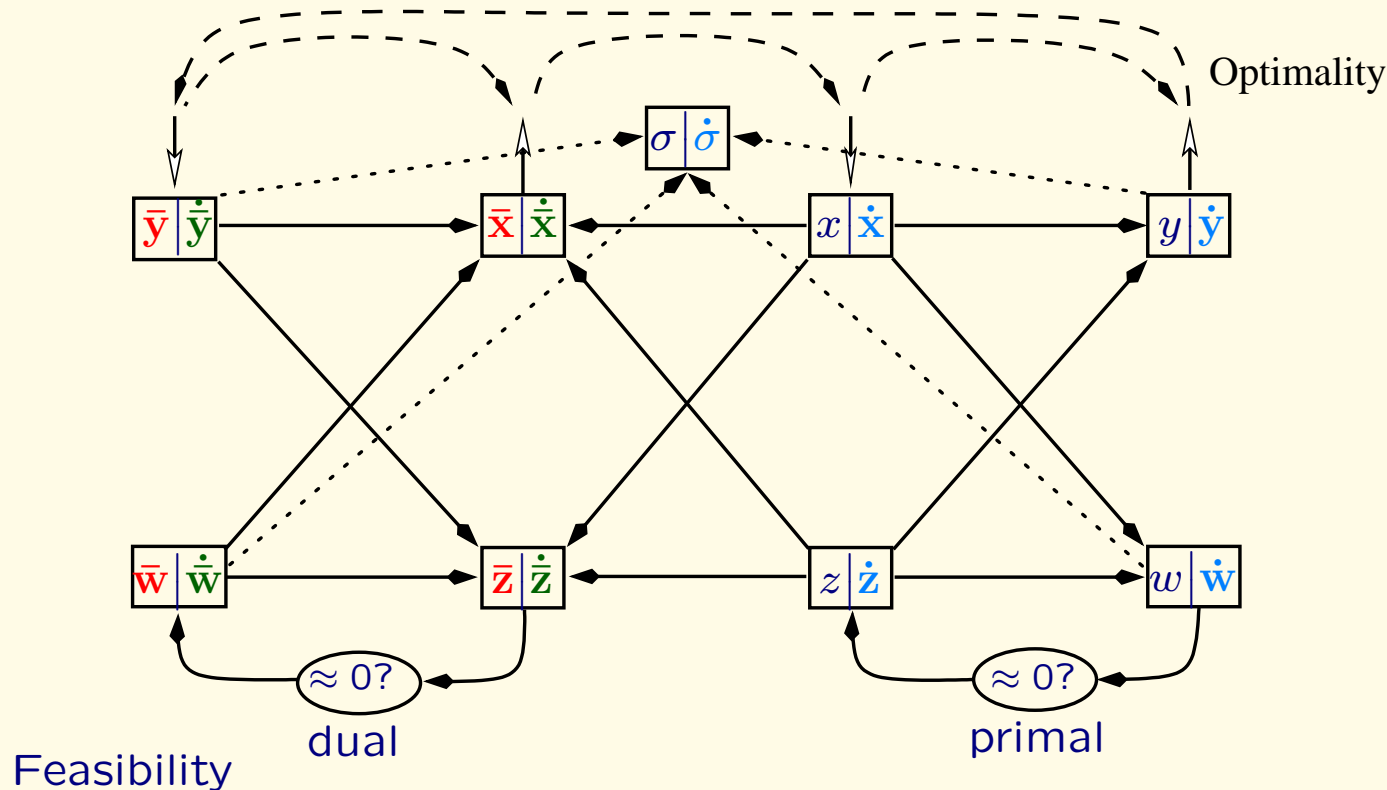
Reduced Jacobian of  $\varphi(x) \equiv f(z_*(x), x)$

$$\begin{aligned} \frac{dy}{dx} &= \frac{\partial y}{\partial x} + \frac{\partial y}{\partial z} \frac{\partial z}{\partial x} \\ &= Z_* \quad (\text{forward}) \\ \equiv \varphi'(x) &= f_x(z_*, x) - \underbrace{f_z(z_*, x) F_z(z_*, x)^{-1}}_{= \bar{W}_* \quad (\text{reverse})} F_x(z_*, x) \end{aligned}$$

## Cost factors for derivative estimates with single and double accuracy

| Symbol                      | Degree | Components            | Single       | Double | Name                 |
|-----------------------------|--------|-----------------------|--------------|--------|----------------------|
| $\varphi$                   | 0      | $m$                   | 1            | 2      | Reduced Function     |
| $\bar{y} \varphi'$          | 1      | $n$                   | 2            | $n$    | Reduced Gradient     |
| $\varphi' \dot{x}$          | 1      | $m$                   | 2            | $m$    | Reduced Tangent      |
| $\varphi'$                  | 1      | $m \times n$          | $\min(m, n)$ | $m n$  | Reduced Jacobian     |
| $\bar{y} \varphi'' \dot{x}$ | 2      | $n$                   | 3            | --     | Second Order Adjoint |
| $\varphi''$                 | 2      | $m \times n \times n$ | $m + n$      | --     | Reduced Tensor       |

# Combination of Direct and Adjoint Fixed Point Iteration with Directional Derivatives



## Pseudo Evolution = Fixed Point Iteration

$$\dot{x} \in \mathbb{R}^n, \quad \dot{z} \in \mathbb{R}^l$$

$$x \in \mathbb{R}^n, \quad z \in \mathbb{R}^l$$

$$\bar{y} \in \mathbb{R}^m, \quad \bar{w} \in \mathbb{R}^l$$

for  $k = 1, 2, \dots$

$$\dot{w} = F'(z, x)\dot{z} + \dot{F}(z, x)\dot{x}$$

$$w = F(z, x), \quad y = f(z, x)$$

$$\bar{z} = \bar{w} F'(z, x) + \bar{y} f'(z, x)$$

exit if  $w \approx 0$  and  $\dot{w} \approx 0$  and  $\bar{z} \approx 0$

$$\dot{z} \leftarrow P_k \dot{w}$$

$$z \leftarrow P_k w \quad \text{with} \quad P_k \in \mathbb{R}^{l \times l} \quad \text{preconditionierer}$$

$$\bar{w} \leftarrow \bar{z} P_k$$

$$\dot{y} = f'(z, x)\dot{z} + \dot{f}(z, x)\dot{x}$$

$$y = f(z, x)$$

$$\bar{x} = \bar{w} \dot{F}(z, x) + \bar{y} \dot{f}(z, x)$$

## Results for Contractive Iteration

$$\|I - P_k F'(z, x)\| \leq \rho < 1$$

$\implies$

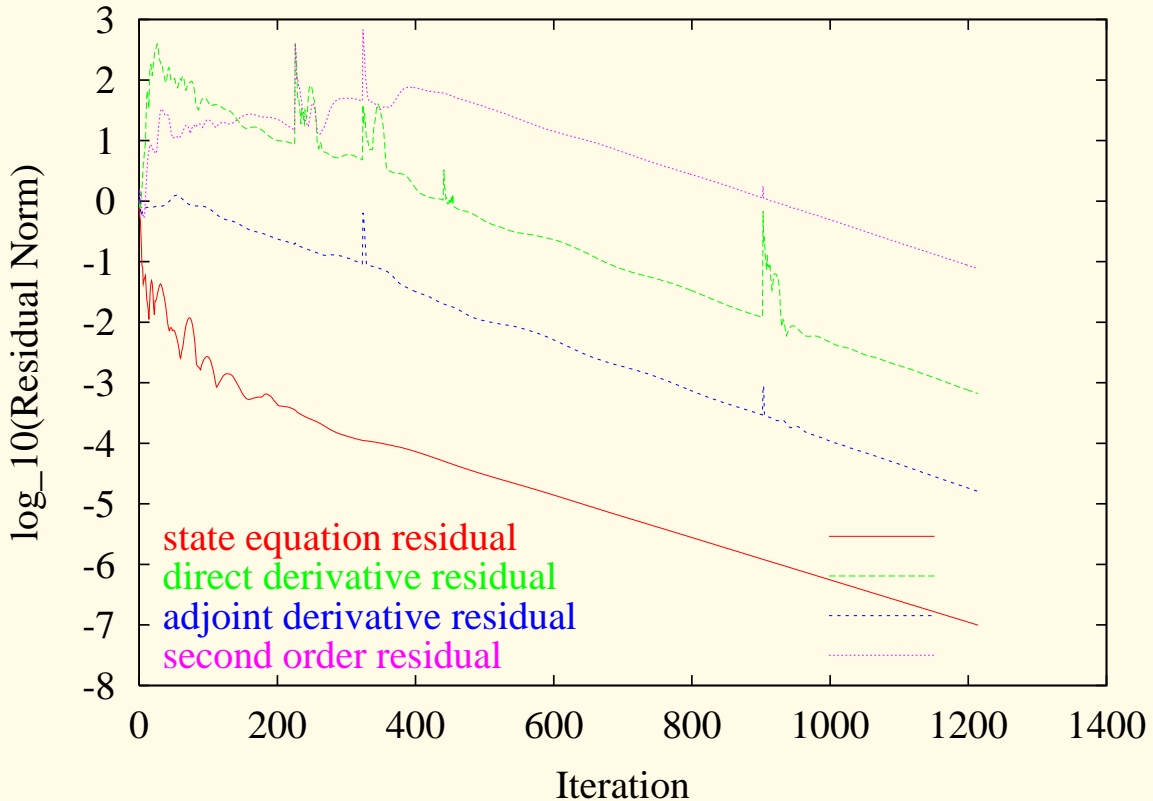
$$\|z - z_*\| = O(\|w\|) = O(\rho^k) \quad \text{Q- linear}$$

$$\|\dot{z} - \dot{z}_*\| = O(\|w\| + \|\dot{w}\|) = O(\rho^k) \quad \text{R- linear}$$

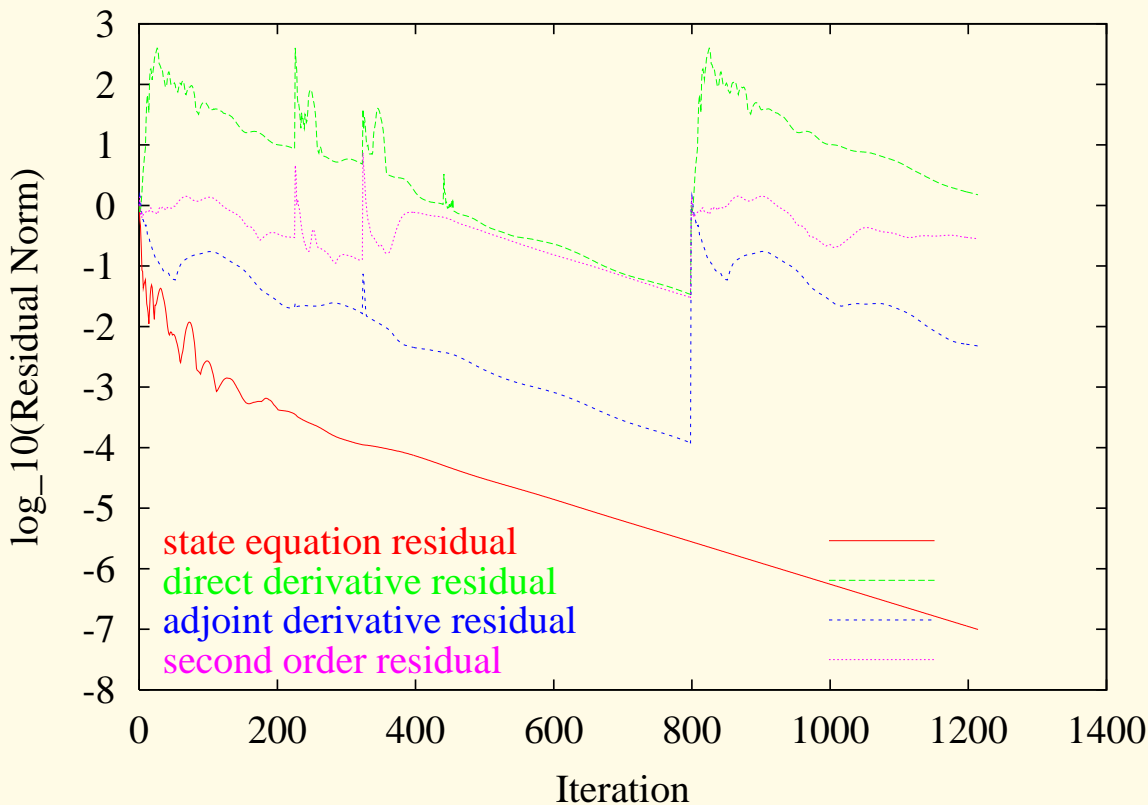
$$\|\bar{w} - \bar{w}_*\| = O(\|w\| + \|\bar{z}\|) = O(\rho^k) \quad \text{R- linear}$$

$\implies$

$$\begin{aligned} & \left. \frac{\partial}{\partial \alpha} \bar{y} y_*(x + \alpha \dot{x}) \right|_{\alpha=0} \\ &= \bar{y} \dot{y} + O(\|w\| + \|\dot{w}\|) \\ &= \bar{x} \dot{x} + O(\|w\| + \|\bar{z}\|) \\ &= \bar{y} \dot{y} + \bar{x} \dot{x} + \bar{w} F'(z, x) \dot{z} + O(\|w\| + \|\dot{w}\| \|\bar{z}\|) \end{aligned}$$



History of Residual Norms lift coefficient and Mach number.



History of Residual Norms for drag coefficient and angle of attack.

## Lessons and Advice

- Nothing to Worry except for Large Problems
- Good Code is good for Differentiation
- Evaluate needed Derivatives selectively
- Exploit Sparsity by Matrix Compression
- Use Checkpointing on True Evolutions
- Avoid Reversal of Fixed-Point Iterations

## Further Topics/Techniques

- Higher Order Taylor and Tensor Coefficients
- Beyond Forward/Reverse  $\Rightarrow$  Cross Country
- Onesided Derivatives/Generalized Gradients
- Sparsity/Structure Detection and Exploitation
- Hierarchical and Parallel Reversal Schedules
- Direct Calculation of Newton-Steps
- Generic Rank, Multiplicities . . . . .