Introduction to Automatic Differentiation

Thomas Kaminski and Ralf Giering

*Fast*Opt

Web: http://www.FastOpt.com

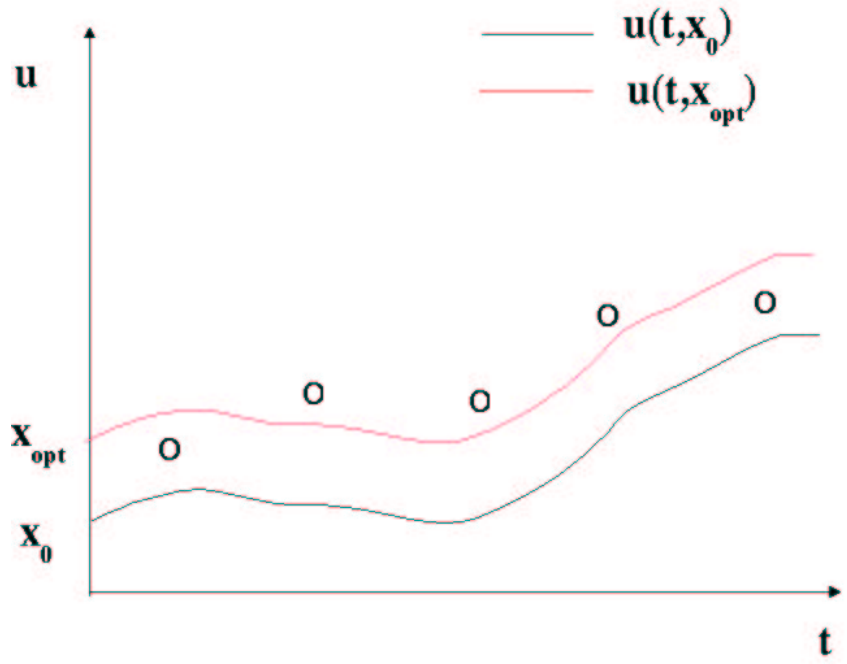
Ecole d' Ete, Versailles, September 2002

Motivation

Solution of an initial value problem:

$\mathbf{u} : \mathbf{R} \times \mathbf{R}^m \rightarrow \mathbf{R}^m$

$: \langle t,x \rangle \rightarrow u(t,x)$



Observations:

$d_1, d_2, \ldots d_n$ at $t_1, t_2, \ldots t_n$

Variational Data Assimilation:

vary x to yield best possible match to observations for $x = x_{opt}$,

i.e. x_{opt} minimises some measure of the misfit, e.g.:

$$J(x) = 1/2 \sum (u(t_i, x) - d_i)^2$$

First Derivative (Gradient) of J(x) w.r.t. x :

DJ(x) or dJ(x)/dx

yields direction of steepest ascent

Second Derivative (Hessian) of J(x): $d^2J(x)/dx^2$

yields curvature of J,

provides estimate of uncertainty in x_{opt}

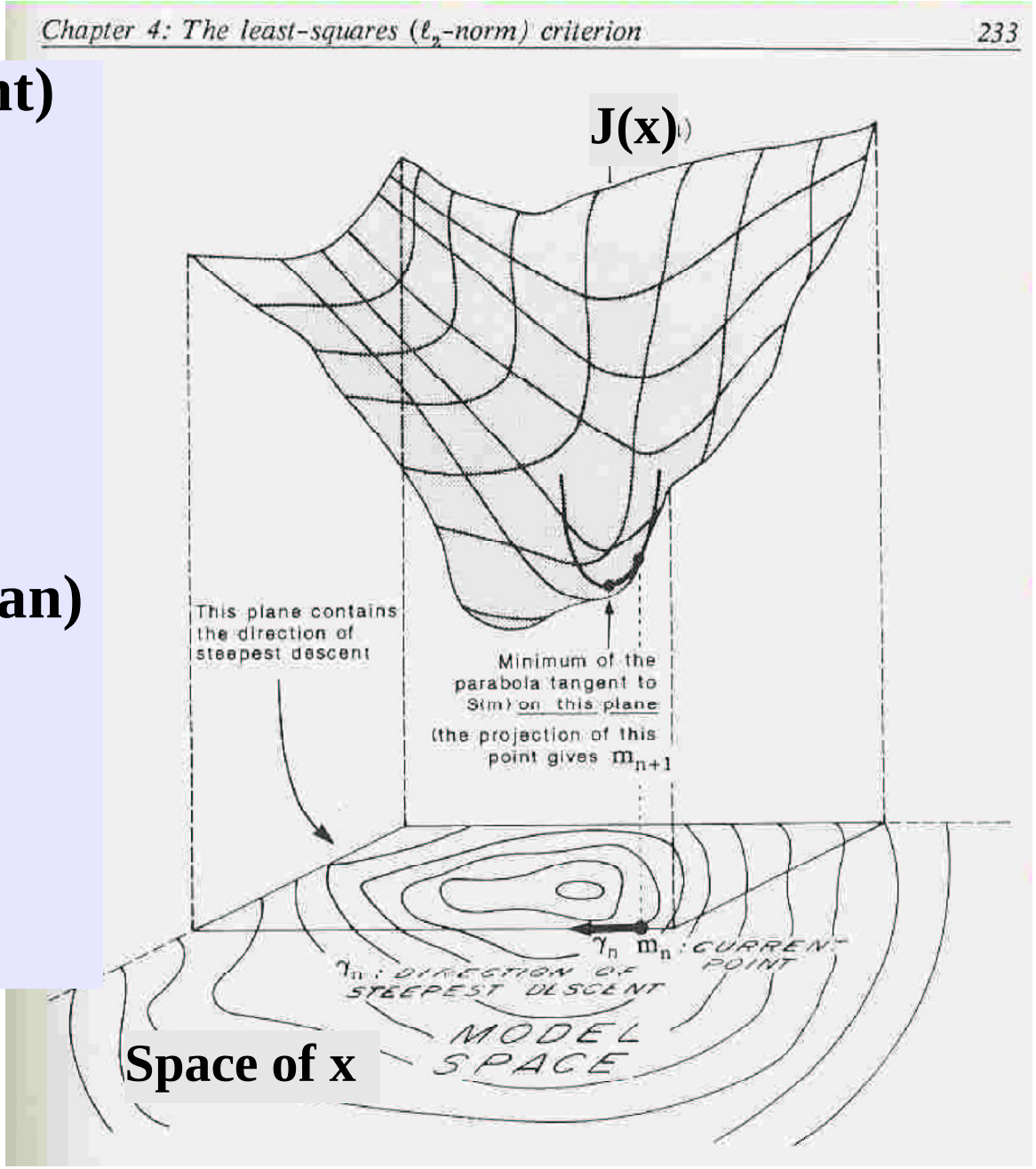


Figure taken from **Tarantola** ’87

Overview

- Motivation
- Intro to AD
- AD Tools Overview
- AD Tool TAF
- Examples

Intro to AD

Example:

$F : R^5 \rightarrow R^1$

$: x \rightarrow y$

$F(x) = f_4 \circ f_3 \circ f_2 \circ f_1(x)$, let any f_i be differentiable

Apply the chain rule for Differentiation!

$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$

Evaluation of Derivative in Forward Mode:

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$$

$$\begin{pmatrix} x & x & x \end{pmatrix} \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x \end{pmatrix} \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x & x & x \end{pmatrix}$$

Intermediate Matrices are large:

- Much space
- Many operations

Evaluation of Derivative in Reverse Mode:

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$$

$$\begin{pmatrix} x & x & x \end{pmatrix} \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix} \begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x \end{pmatrix} \begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x \end{pmatrix} \begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

$$\begin{pmatrix} x & x & x & x & x \end{pmatrix}$$

Intermediate Matrices small:

- Less space
- Fewer operations

Reverse and Adjoint

$$DF = \overrightarrow{Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1} \qquad DF^T = \overleftarrow{Df_1^T \cdot Df_2^T \cdot Df_3^T \cdot Df_4^T}$$

Propagation of Derivatives

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Forward:

$$x' = z'_0 \xrightarrow{Df_1} z'_1 \xrightarrow{Df_2} z'_2 \xrightarrow{Df_3} z'_3 \xrightarrow{Df_4} z'_4 = y'$$

$z'_0 \ldots z'_4$ are called tangent linear variables

Reverse:

$$\bar{x} = \bar{z}_0 \xleftarrow{Df_1^T} \bar{z}_1 \xleftarrow{Df_2^T} \bar{z}_2 \xleftarrow{Df_3^T} \bar{z}_3 \xleftarrow{Df_4^T} \bar{z}_4 = \bar{y}$$

$\bar{z}_0 \ldots \bar{z}_4$ are called adjoint variables

Forward Mode

Interpretation of tangent linear variables

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Forward:

$$x' = z'_0 \xrightarrow{Df_1} z'_1 \xrightarrow{Df_2} z'_2 \xrightarrow{Df_3} z'_3 \xrightarrow{Df_4} z'_4 = y'$$

$x'=Id$

$$z'_2 = Df_2 \cdot z'_1 = Df_2 \cdot Df_1 \cdot x' = Df_2 \cdot Df_1$$

tangent linear variable z'_2 holds derivative of z_2 w.r.t. x: dz_2/dx

Function y=F(x) defined by Fortran code:

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Task: Evaluate **DF = dy/dx** in forward mode!

Problem: Identify $\mathbf{f}_1$ $\mathbf{f}_2$ $\mathbf{f}_3$ $\mathbf{f}_4$ $\mathbf{z}_1$ $\mathbf{z}_2$ $\mathbf{z}_3$

Observation: $\mathbf{f}_3$: `w = sin(u)` can't work, dimensions don't match!

Instead:
Just take all variables

$$f_3 : z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \longrightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

A step in forward mode

$$f_3 \quad : \quad z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \quad \longrightarrow \quad z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

```
w       = sin(u)
```

$$z'_3 = Df_3 \, z'_2$$

$$\begin{pmatrix} x'(1) \\ x'(2) \\ x'(3) \\ u' \\ v' \\ w' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x'(1) \\ x'(2) \\ x'(3) \\ u' \\ v' \\ w' \\ y' \end{pmatrix}$$

```
gx(1) = gx(1)
gx(2) = gx(2)
gx(3) = gx(3)
gu    = gu
gv    = gv
gw    = gu*cos(u)
gy    = gy
```

Entire Function + required variables

Function code

```
u = 3*x(1)+2*x(2)+x(3)

v = cos(u)

w = sin(u)

y = v * w
```

Forward mode/ tangent linear code

```
gu = 3*gx(1)+2*gx(2)+gx(3)
 u = 3* x(1)+2* x(2)+ x(3)
gv = -gu*sin(u)
 v = cos(u)
gw = gu*cos(u)
 w = sin(u)
gy = gv*w + v*gw
 y = v * w
```

u v w are *required* variables, their values need to be provided to the derivative statements

Active and passive variables

Consider slight modification of code for y = F(x):

```
u   = 3*x(1)+2*x(2)+x(3)
pi  = 3.14
v   = pi*cos(u)
w   = pi*sin(u)
sum = v + u
y   = v * w
```

Observation: Variable **sum** (diagnostic) does not influce the function value y
Variable **pi** (constant) does not depend on the independent variables x

Variables that do influence y and are influenced by x are called *active variables*.
The remaining variables are called *passive variables*

Active and passive variables

Function code

```
u   = 3*x(1)+2*x(2)+x(3)

pi  = 3.14

v   = pi*cos(u)

w   = pi*sin(u)

sum = v + u

y   = v * w
```

Forward mode/ tangent linear code

```
gu = 3*gx(1)+2*gx(2)+gx(3)
u  = 3* x(1)+2* x(2)+ x(3)

pi = 3.14
gv = -gu*pi*sin(u)
v  = pi*cos(u)
gw = gu*pi*cos(u)
w  = pi*sin(u)


gy = gv*w + v*gw
```

For passive variables
- no tangent linear variables needed
- no tangent linear statements for their assignments needed

Reverse Mode

Function:

$$x = z_0 \xrightarrow{f_1} z_1 \xrightarrow{f_2} z_2 \xrightarrow{f_3} z_3 \xrightarrow{f_4} z_4 = y$$

Forward:

$$x' = z'_0 \xrightarrow{Df_1} z'_1 \xrightarrow{Df_2} z'_2 \xrightarrow{Df_3} z'_3 \xrightarrow{Df_4} z'_4 = y'$$

$x' = Id$ $\qquad z'_2 = Df_2 \cdot z'_1 = Df_2 \cdot Df_1 \cdot x' = Df_2 \cdot Df_1$

tangent linear variable z'_2 holds derivative of z_2 w.r.t. x : dz_2/dx

Reverse:

$$\bar{x} = \bar{z}_0 \xleftarrow{Df_1^T} \bar{z}_1 \xleftarrow{Df_2^T} \bar{z}_2 \xleftarrow{Df_3^T} \bar{z}_3 \xleftarrow{Df_4^T} \bar{z}_4 = \bar{y}$$

$\bar{y} = Id$ $\qquad \bar{z}_2 = Df_3^T \cdot \bar{z}_3 = Df_3^T \cdot Df_4^T \cdot \bar{y} = (Df_4 \cdot Df_3)^T$

adjoint variable $\bar{z}_2$ holds derivative of y w.r.t. z_2 : dy/dz_2

Function F defined by Fortran code:

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Task: Evaluate DF in reverse mode!

Again:
Take all Variables

$$f_3 : z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \longrightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

A step in reverse mode

$$f_3 \quad : \quad z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \quad \longrightarrow \quad z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

$$Df_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```
w = sin(u)
```

$$\overline{z_2} = Df_3^T \, \overline{z_3}$$

$$\begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{w} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cos(u) & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{w} \\ \overline{y} \end{pmatrix}$$

```
adx(1) = adx(1)
adx(2) = adx(2)
adx(3) = adx(3)
adu = adu+adw*cos(u)
adv = adv
adw = 0.
ady = ady
```

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Adjoint code

```
   u = 3*x(1)+2*x(2)+x(3)
   v = cos(u)
   w = sin(u)

adv = adv+ady*w
adw = adw+ady*v
ady = 0.

adu = adu+adw*cos(u)
adw = 0.

adu = adu-adv*sin(u)
adv = 0.

adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Function F defined by Fortran code:

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y = v * w
```

Typically, to save memory, variables are used more than once!

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

$$f_3 : z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ y \end{pmatrix} \rightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ \sin(u) \\ v \\ y \end{pmatrix}$$

How does the adjoint code change?

```
u = sin(u)
```

$$Df_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\overline{z_2} = Df_3^T \overline{z_3}$$

$$\begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{y} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \overline{x}(1) \\ \overline{x}(2) \\ \overline{x}(3) \\ \overline{u} \\ \overline{v} \\ \overline{y} \end{pmatrix}$$

```
adu = adu*cos(u)
```

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

Adjoint code

```
  u = 3*x(1)+2*x(2)+x(3)
  v = cos(u)
  u = sin(u)
adv = adv+ady*u
adu = adu+ady*v
ady = 0.
  u = 3*x(1)+2*x(2)+x(3)
adu = adu*cos(u)
adu = adu-adv*sin(u)
adv = 0.
adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Store and Retrieve values

Function code

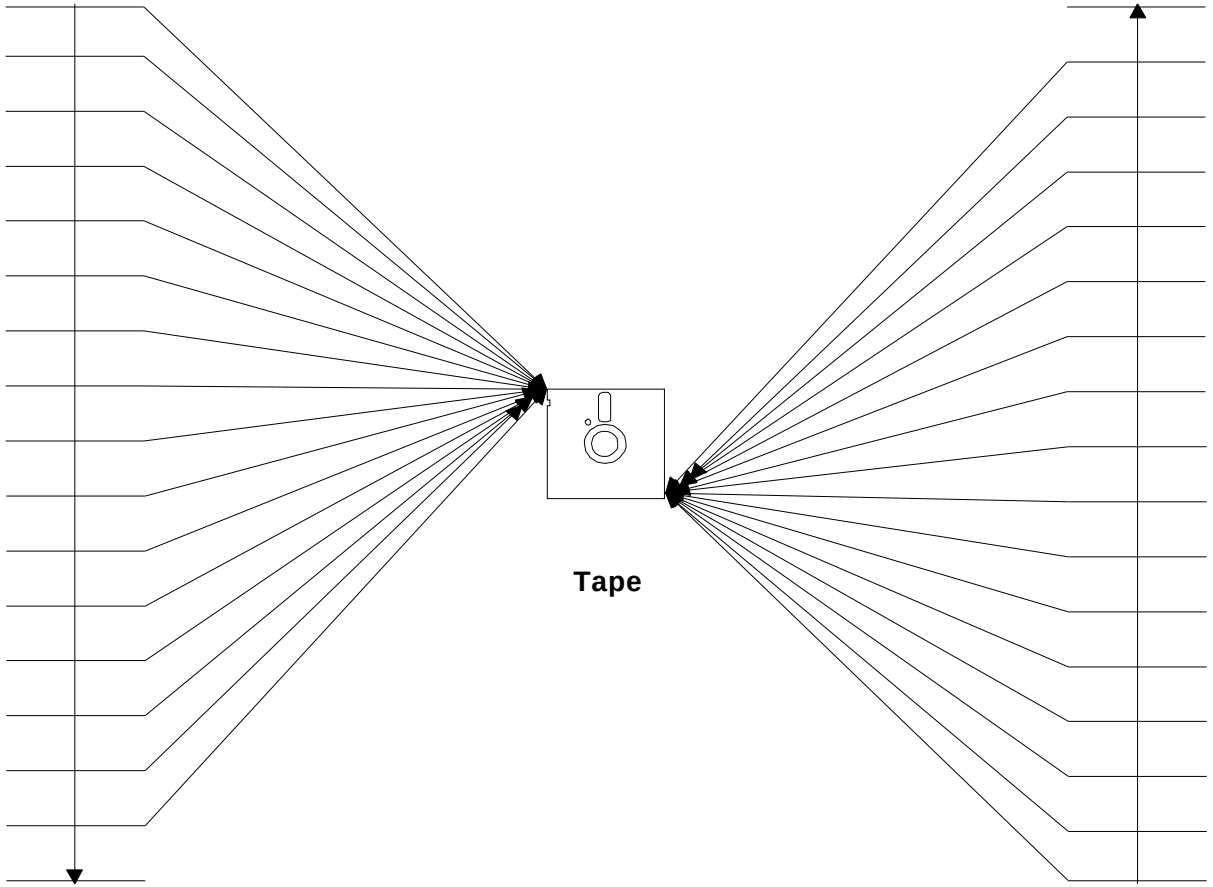
```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
u = sin(u)
y = v * u
```

Bookkeeping must be arranged
(store / retrieve)
Values can be saved
- **on disc or**
- **in memory**

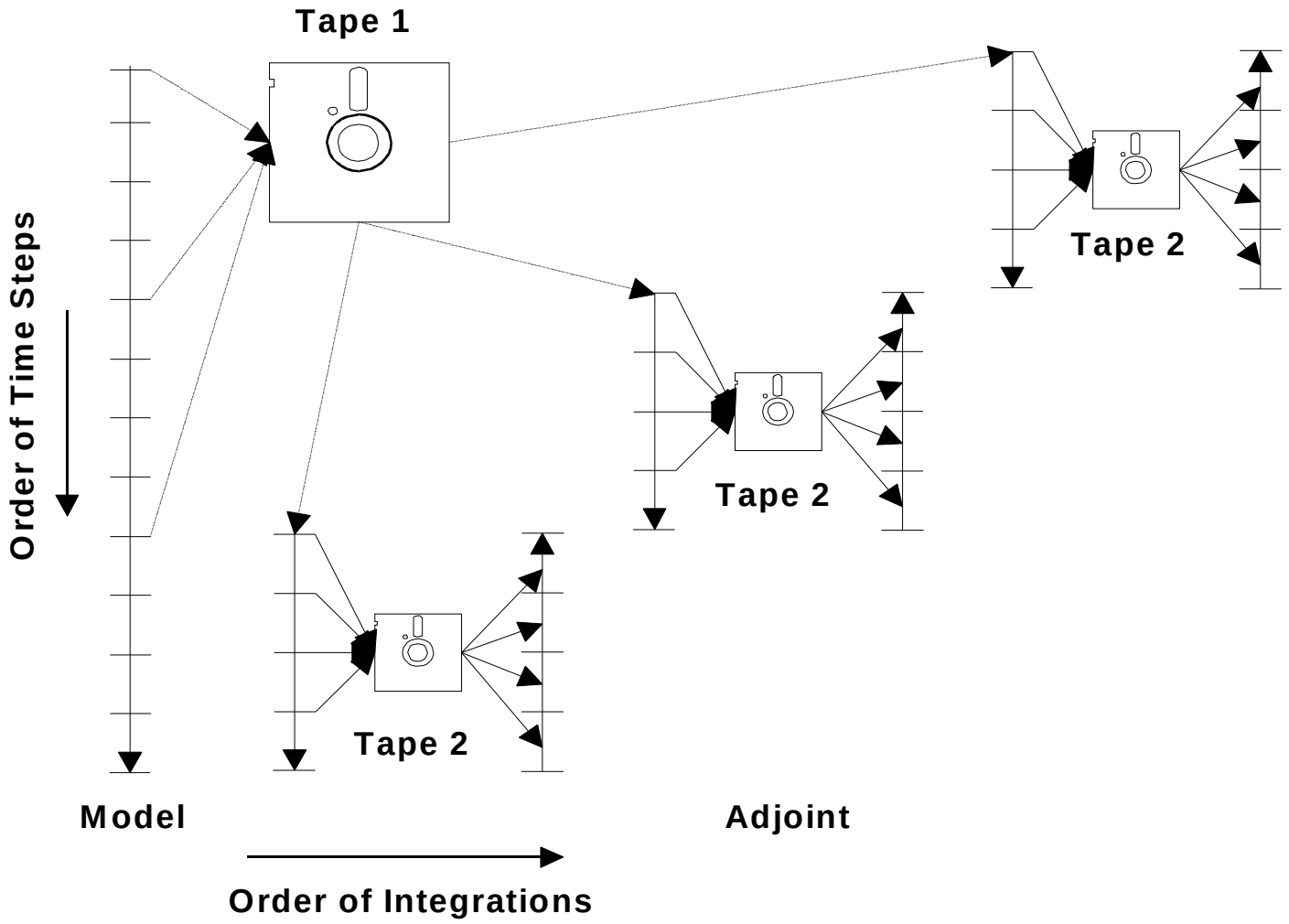
Adjoint code

```
u = 3*x(1)+2*x(2)+x(3)
   store (u)
v = cos(u)
u = sin(u)

adv = adv+ady*u
adu = adu+ady*v
ady = 0.

   retrieve (u)
adu = adu*cos(u)

adu = adu-adv*sin(u)
adv = 0.

adx(3) = adx(3)+3*adu
adx(2) = adx(2)+2*adu
adx(1) = adx(1)+adu
adu    = 0.
```

Storing of required variables



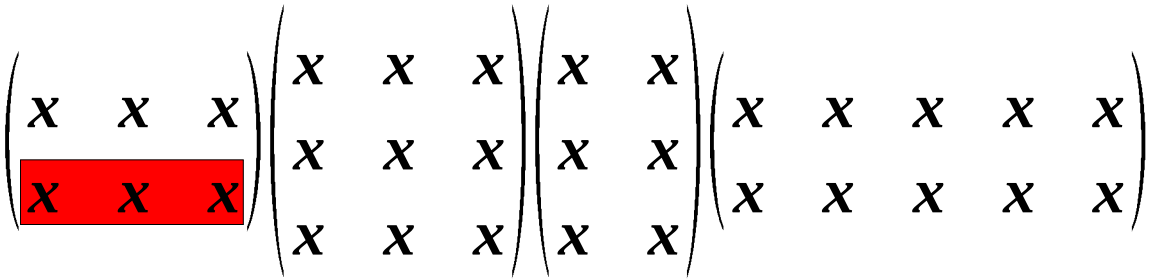
Checkpointing

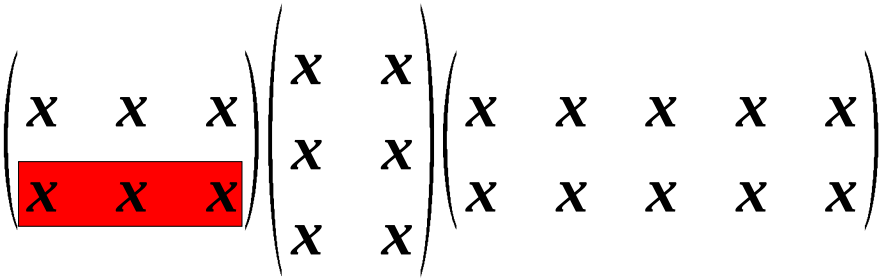


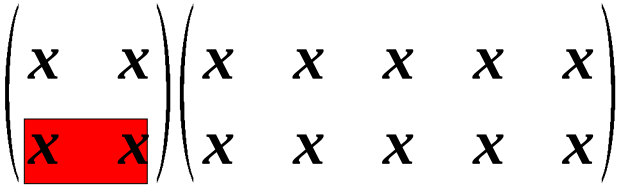
Vector valued function in Reverse Mode

$$F : R^5 \rightarrow R^2$$
$$: x \rightarrow y$$

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1$$



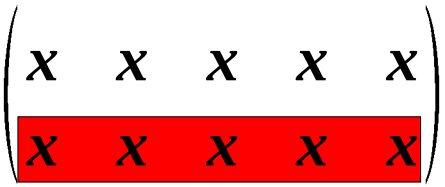




Intermediate Results (adjoint Variables)

$\bar{z}_1, \bar{z}_2, \ldots$

are Matrices now rather than vectors!



$$f_3 : z_2 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ w \\ y(1) \\ y(2) \end{pmatrix} \rightarrow z_3 = \begin{pmatrix} x(1) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y(1) \\ y(2) \end{pmatrix}$$

$$Df_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \cos(u) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\overline{z_2} = Df_3^T \overline{z_3}$$

$$\begin{pmatrix} \overline{u}(1) & \overline{u}(2) \\ \overline{w}(2) & \overline{w}(2) \end{pmatrix} = \begin{pmatrix} 1 & \cos(u) \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \overline{u}(1) & \overline{u}(2) \\ \overline{w}(2) & \overline{w}(2) \end{pmatrix}$$

(only relevant part shown)

A step in reverse mode for a vector valued function:

```
w = sin(u)
```

```
do i=1,2
   adu(i) = adu(i)+adw(i)*cos(i,u)
   adw(i) = 0.
enddo
```

Adjoint code for vector valued function

Function code

```
u = 3*x(1)+2*x(2)+x(3)
v = cos(u)
w = sin(u)
y(1) = v * w
y(2) = v + w
```

Adjoint code

```
      u = 3*x(1)+2*x(2)+x(3)
      v = cos(u)
      w = sin(u)
do i=1,2
   adv(i) = ady(i,2)
   adw(i) = ady(i,2)
   ady(i,2) = 0.
enddo

do i=1,2
   adv(i) = adv(i)+ady(i,1)*w
   adw(i) = adw(i)+ady(i,1)*v
   ady(i,1) = 0.
enddo

do i=1,2
   adu(i) = adu(i)+adw(i)*cos(u)
   adw(i) = 0.
enddo

do i=1,2
   adu(i) = adu(i)-adv(i)*sin(u)
   adv(i) = 0.
enddo

and so on...
```

Summary

- **AD exploits chain rule**
- **Forward and reverse modes**
- **Active/Passive variables**
- **Required variables:**
  Recomputation vs. Store/Reading
- **Scalar and vector modes**

Further Reading

- AD Book of **Andreas Griewank**: *Evaluating Derivatives: Principles of Algorithmic Differentiation, SIAM, 2000*
- Books on **AD Workshops:**
  Nice 2000: *Corliss et al. (Eds.), Springer*
  Santa Fe 1996: *Berz et al. (Eds.), SIAM*
  Beckenridge 1991: *Griewank and Corliss (Eds.), SIAM*
- **Olivier Talagrand's** overview article in Santa Fe Book
- RG/TK article: *Recipes of Adjoint Code Construction, TOMS, 1998*

Overview

- Motivation
- Intro to AD
- AD Tools Overview
- AD Tool TAF
- Examples

Operator Overloading

- **E.g. Automatic Differentiaton by OverLoading in C++ / F (ADOLC / ADOLF)**
  or Integrated Modeling and Analysis System (IMAS)
- **Overloading as capability of programming languages like Fortran 90 or C++**
- **Use new class definition for active variables, e.g. „adouble“, which includes tangent linear/adjoint variables**
- **Operators are overloaded, so that they also operate on the tangent linear/adjoint variables**
- **The reverse mode interprets a tape that is recorded during an initial function evaluation.**
  The tape contains each operation plus the values of the operands

Fortran source to source AD tools

see www.autodiff.org for more information

- ADIFOR−2 (F77): TLM
  - Future version 3 will support ADM
- AUTO_DERIV (F77 to F95): TLM + ADM
  - ?
- TAF (F77 to F95) −> ...
- TAMC (F77): TLM + ADM
  - remote access to server
  - no more maintainance
- TAPENADE (F77 to F95): TLM + ADM
  - successor of ODYSSEE
  - accessible through web or download

Overview

- Motivation
- Intro to AD
- AD Tools Overview
- AD Tool TAF
- Examples

A few facts about FastOpt

- Founded in February 2000 at Hamburg
- By Ralf Giering and Thomas Kaminski
- Two kinds of business:
  - Develop and provide tools (TAF) for Automatic Differentiation (AD)
  - Carry out Consulting Projects with focus on AD, Inverse Modelling, Optimisation...
- 13 years of Experience in AD

TAF: Transformation of Algorithms in Fortran

- Source−to−source translator for Fortran−77 to 95
- Commercial successor of TAMC
- Forward and reverse mode (1st derivatives): Tangent linear and adjoint models
- Efficient Hessian (2nd derivative) code by applying TAF twice
- Command line program with many options
- TAF−Directives are Fortran comments
- Extensive and complex code analyses (similar to optimising compilers)
- Generated code is structured and well readable

TAF: Ongoing Development

- TAF is constantly being adapted to new Fortran standards
  - Fortran 2000
  - OpenMP
- TAF code analyses are constantly being extended
- TAF algorithms are constantly being improved and adapted to the needs of the users
- FastOpt is giving support for TAF users
- FastOpt is offering consulting for AD and further projects

TAF Derivatives of Large Models

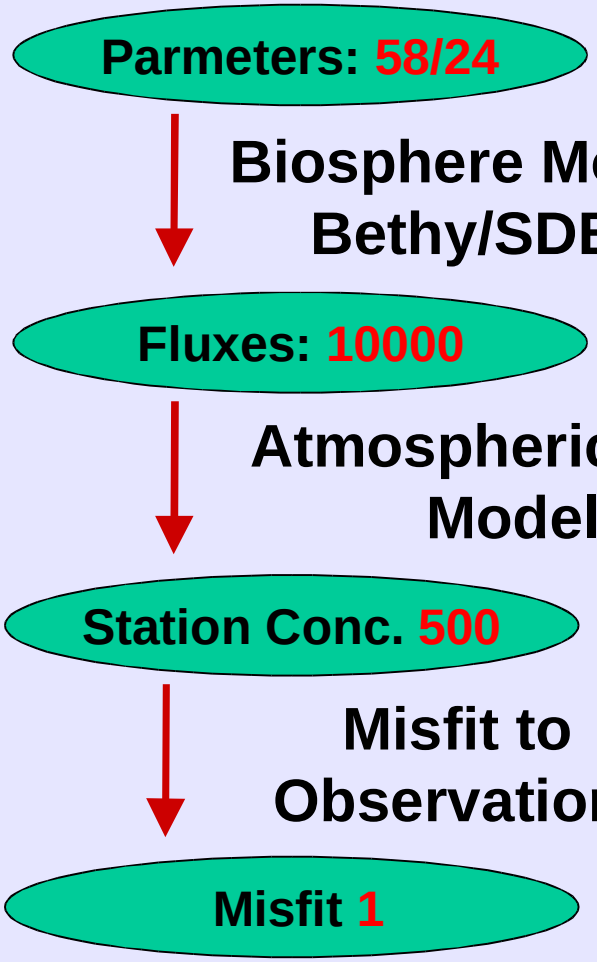
Model (Who)	Lines	Lang	TLM	ADM	Ckp	HES
NASA/NCAR (w. Todling & Lin)	87'000	F90	2.7	8.4	2 lev	
MOM3 (Galanti & Tziperman)	50'000	F77	Yes	4.0	2 lev	
MITGCM (ECCO Consortium)	100'000	F77	1.8	4.5	2 lev	11,0
BETHY (w. Knorr, Rayner, Scholze)	5'000	F90	1.5	3.6	2 lev	12.7
Nav.-Stokes-Solver (Hinze-Slawig)	1'000	F77		1.3	steady	
NSC2KE	3'400	F77		5.0	steady	

- Lines: total number of Fortran lines without comments
- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- Number for HES gives CPU time for Hessian * vector
- 2 level checkpointing costs 1 additional model run

Carbon Cycle Data Assimilation

Adjoint / Tangent Linear / Hessian

(with Heimann, Knorr, Rayner, Scholze)



1. **Parameter Optimisation:**

   Adjoint of Parameters –> Misfit

2. **Parameter Uncertainty:**

   Hessian of Parameters –> Misfit

3. **Uncertainty of Predictions/Diagnostics:**

   Tangent Linear / Adjoint of Parameters –> Diagnostics

Ocean Data Assimilation
ECCO Consortium

- **MIT–GCM: Primitive Equation Model of General Oceanic Circulation**
- **Various Configurations ~ 100’000 lines of Fortran 77 (without comments)**
- **Uses message passing interface (MPI) for Parallelisation**
- **Tangent Linear, Adjoint, Hessian code generated by TAF Only hand written code for MPI wrappers**
- **Is used for Variational Data Assimilation, Uncertainty Analysis, Kalman Filter and Sensitivity Studies**

Atmospheric Data Assimilation
at Data Assimilation Office (Todling, Lin...)

- DAO finite Volume GCM: Model of General Atmospheric Circulation
- Hydrodynamics kernel by Lin, Lin/Rood
- ~ 87’000 lines of Fortran 90 (without comments)
- Uses message passing interface (MPI)
  as well as OpenMP for Parallelisation
- Tangent Linear and Adjoint generated by TAF
  Only hand written code for MPI wrappers
  OpenMP handled by TAF
- To be used by DAO for Data Assimilation (Retrospective Analysis System), Sensitivity Studies, Singular Vector Detection ...

Performance compared to hand coded adjoints

Code	Hand	TAF/TAMC	relativ
EPT (MINPACK–2)	1.5	1.9	26%
GL1 (MINPACK–2)	1.5	1.7	13%
GL2 (MINPACK–2)	2.1	1.3	–40%
MSA (MINPACK–2)	1.75	1.6	–9%
PJB (MINPACK–2)	1.75	2.2	+25%
SSC (MINPACK–2)	1.18	1.13	–5%
Nav.–Stokes (H & S)	1.9	1.3	–30%

=> Performance of TAF generated adjoints is comparable to that of hand written adjoints

TAMC Adjoints of Large Models

Model (Who)	TLM	ADM	Checkp
PUMA (Blessing)	2.0	2.8	
MM5 (Nehrkorn et al.)	Yes	Yes	?
TM2 (Kaminski et al.)		3.4	2 level
TM3 (Roedenbeck et al.)		Yes	?
HCM (Eckert)	Yes	3.7	2 level
MOM3 (Galanti & Tziperman)		4.0	2 level
MITGCM (ECCO Consortium)	1.8	4.5	2 level
HOPE (Oldenborg et al.)		5-6	3 level
WAM (Hersbach)		3.7	2 level

- Numbers for TLM and ADM give CPU time for (function + gradient) relative to forward model
- 2 (3) level checkpointing costs 1 (2) additional model runs

Overview

- Motivation
- Intro to AD
- AD Tools Overview
- AD Tool TAF
- Examples