Intro to Automatic Differentiation

Thomas Kaminski (http://FastOpt.com)

Thanks to:

Simon Blessing (FastOpt), Ralf Giering (FastOpt), Nadine Gobron (JRC),

Wolfgang Knorr (QUEST), Thomas Lavergne (Met.No), Bernard Pinty (JRC),

Peter Rayner (LSCE), Marko Scholze (QUEST), Michael Voßbeck (FastOpt)

4th Earth Observation Summer School, Frascati, August 2008

*Fast*Opt

Recap

- Derivative information useful for solving inverse problems
  - 1st derivative of cost function for minimisation with gradient algorithm (mean value of posterior PDF)
  - 2nd derivative of cost function for approximation of uncertainties (covariance of posterior PDF)
- This lecture: Construction of efficient derivative code

Outline

- Chain Rule
- Basic Concepts: Active and Required Variables
- Tangent linear and adjoint code
- Verification of derivative code

Intro AD

Example:

$$F \quad : \mathbf{R}^5 \rightarrow \mathbf{R}^1$$

$$\quad : x \rightarrow y$$

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

Apply the chain rule for Differentiation!

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

AD: Forward vs. Reverse

Example function:
N=5 inputs and M=1 output

Forward mode

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

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

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

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

Reverse mode

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

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

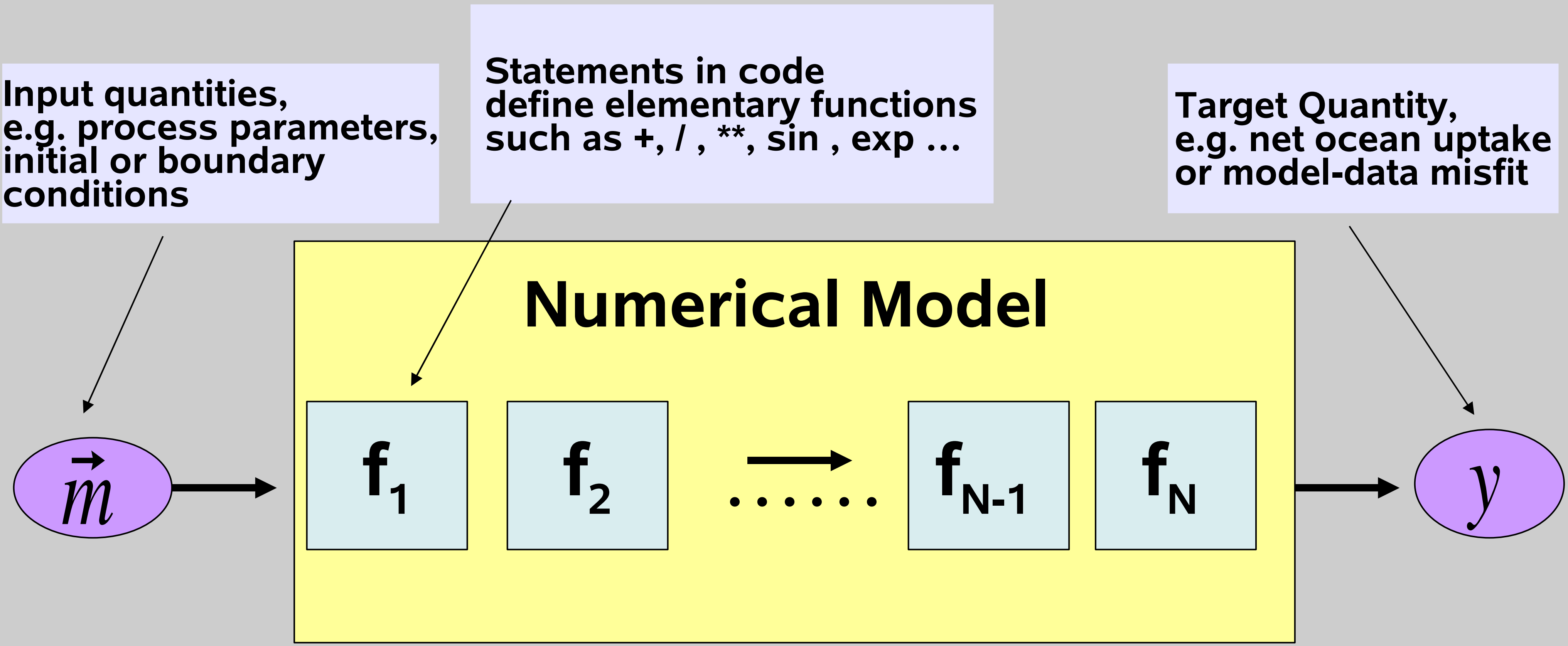
$$= \begin{pmatrix} x & x \end{pmatrix}\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}$$

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

Intermediate Results

- **Forward and reverse mode yield the same result.**
- **Reverse mode: fewer operations (time) and less space for intermediates (memory)**
- **Cost for forward mode grows with N**
- **Cost for reverse mode grows with M**

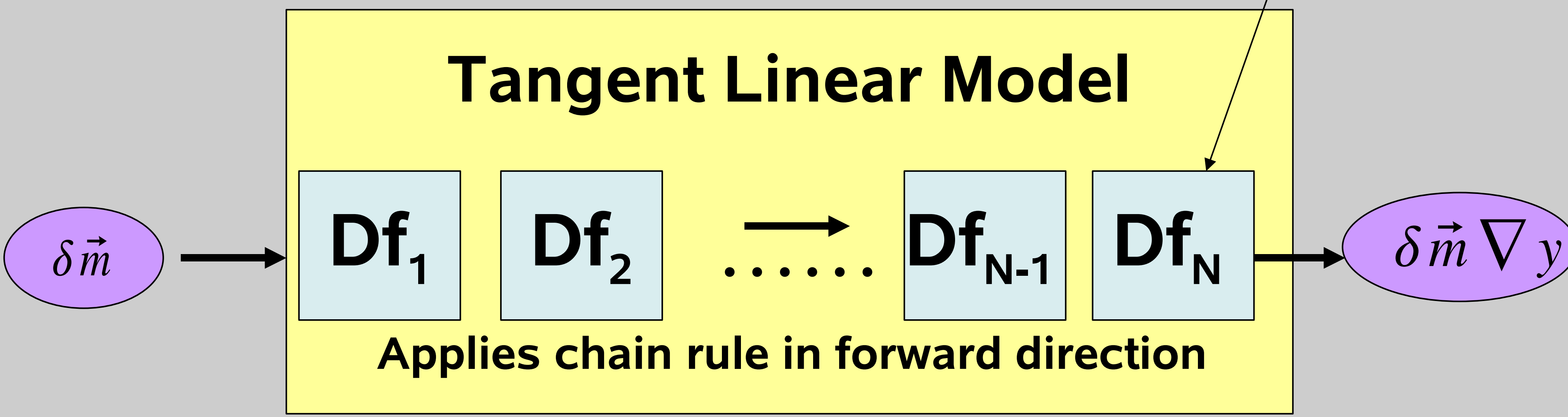
Sensitivities via AD



Sensitivities via AD

Cost of gradient evaluation proportional to # of parameters

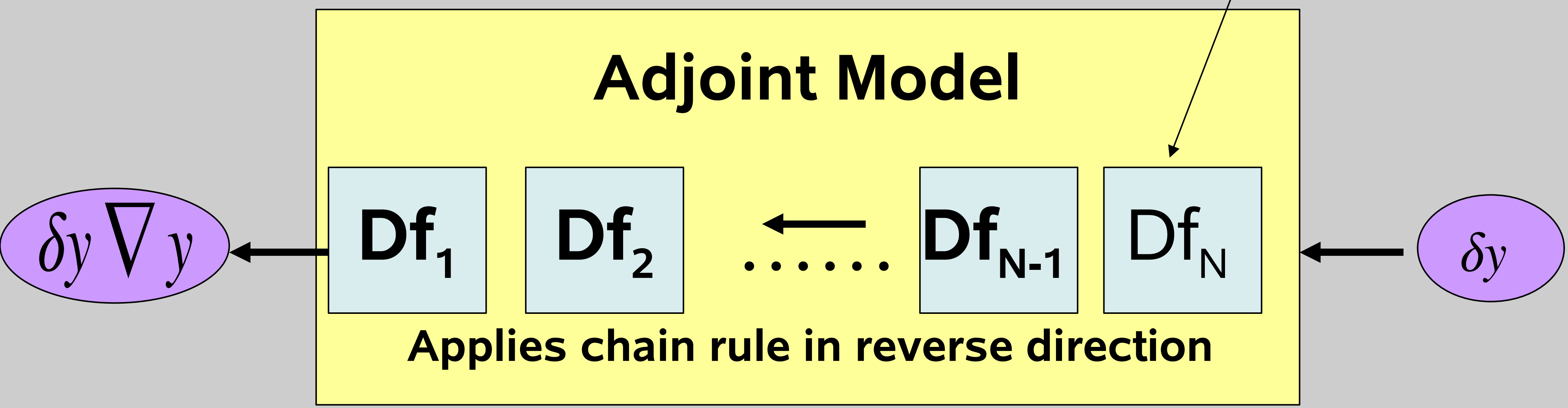
Derivatives of elementary functions are simple, they define local Jacobians



Sensitivities via AD

Cost of gradient evaluation independent of # of inputs

Derivatives of elementary functions are simple, they define local Jacobians



Reverse and Adjoint

$$DF = Df_4 \cdot Df_3 \cdot Df_2 \cdot Df_1 \qquad DF^T = Df_1^T \cdot Df_2^T \cdot Df_3^T \cdot Df_4^T$$

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

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

x'=v: $z'_2 = Df_2 \cdot Df_1 \cdot v$

tangent linear variable z'_2 holds directional derivative of z_2 w.r.t. x in direction of v

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: 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

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

```
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:

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

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

Adjoint variable $\bar{z}_2$ holds (transposed) derivative of **y** w.r.t. z_2: dy/dz_2

For example: y scalar, i.e. $\bar{y}=1$

A step in reverse mode

$$f_3 \quad : \quad z_2 = \begin{pmatrix} x(1) \\ x(\text{٢}) \\ x(3) \\ u \\ v \\ w \\ y \end{pmatrix} \rightarrow \quad z_3 = \begin{pmatrix} x(\text{١}) \\ x(2) \\ x(3) \\ u \\ v \\ \sin(u) \\ y \end{pmatrix}$$

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

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

$$\overline{z_2} = Df_3^T \, \overline{z_{\text{٣}}}$$

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

```
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
```

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
```

Adjoint code

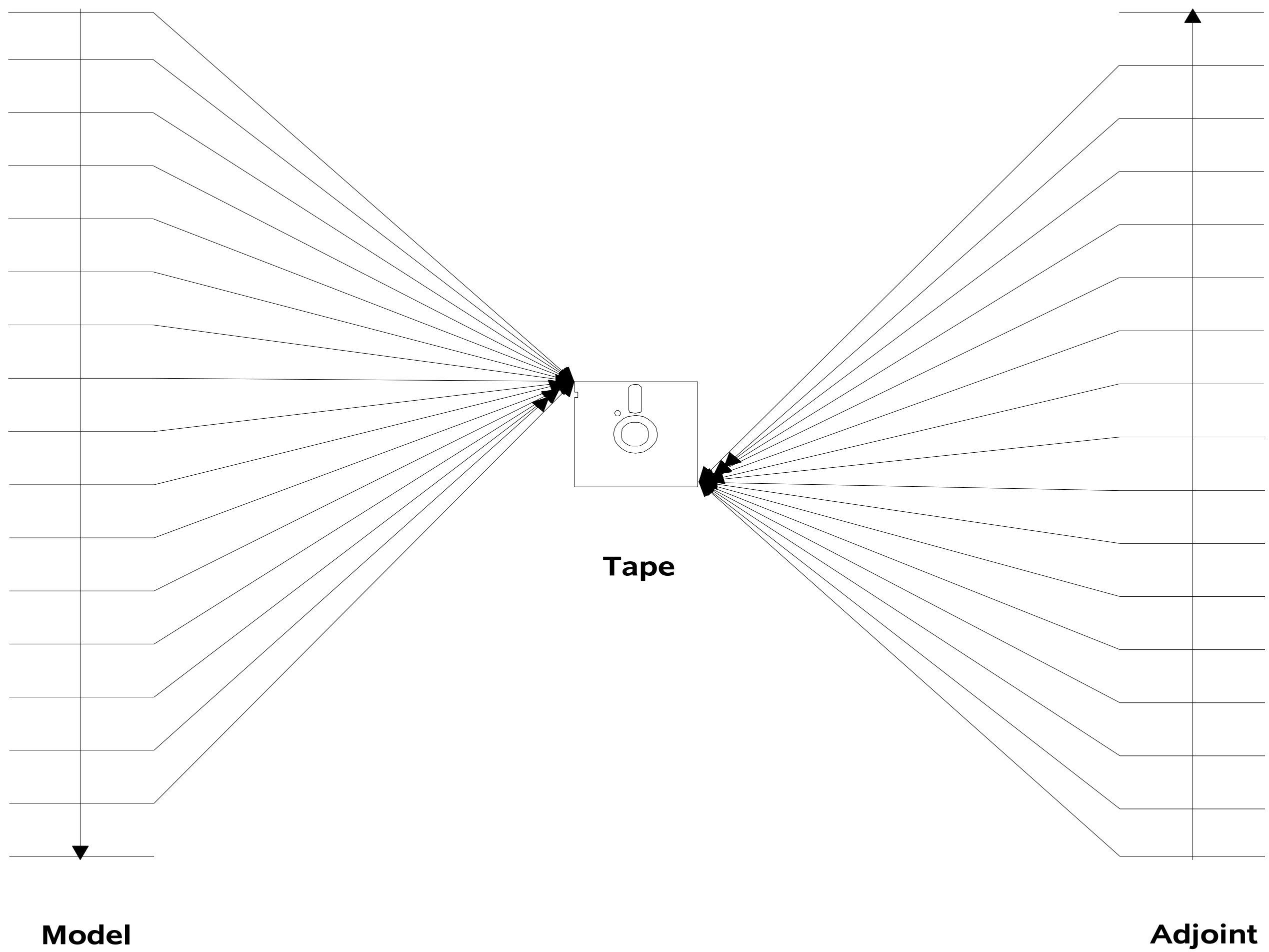
```
u = 3*x(1)+2*x(2)+x(3)
   store (u)
v = cos(u)
u = sin(u)

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.
```

Bookkeeping must be arranged (store / retrieve)

Values can be saved

- **on disc or**
- **in memory**

Storing of required variables



AD Summary

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

Further Reading

- AD Book of **Andreas Griewank**: *Evaluating Derivatives: Principles of Algorithmic Differentiation, SIAM, 2000*
- Books on **AD Workshops:**
  Chicago 2004: *Buecker et al. (Eds.), Springer*
  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*

AD Tools

- Specific to programming language
- Source-to-source / Operator overloading
- For details check http://www.autodiff.org !

Selected Fortran tools (source to source):

- ADIFOR (M. Fagan, Rice, Houston)
- Odyssee (C. Faure) -> TAPENADE (L. Hascoet, INRIA, Sophia- Antipolis, France)
- TAMC (R. Giering) -> TAF (FastOpt)

Selected C/C++ tools:

- ADOLC (A. Walther, TU-Dresden, Operator Overloading)
- ADIC (P. Hovland, Argonne, Chicago)
- TAC++ (FastOpt)

very simple example

ex1.f90

```
subroutine ex1( x, u, y )
  implicit none
  real x, u, y

  y = 4*x + sin(u)
end
```

drv1tlm.f90

```
program driver
  implicit none
  real x, u, y

  x = 1.2
  u = 0.5
  call ex1( x, u, y )
  print *,' y = ',y
end
```

command line:

taf -f90 -v2 **-forward** -toplevel ex1 -input x,u -ouput y ex1.f90

generated tangent linear code (ex1_tl.f90)

```
subroutine ex1_tl( x, x_tl, u, u_tl, y, y_tl )

implicit none

!===============================================
! declare arguments
!===============================================
real u
real u_tl
real x
real x_tl
real y
real y_tl

!-----------------------------------------------
! TANGENT LINEAR AND FUNCTION STATEMENTS
!-----------------------------------------------
y_tl = u_tl*cos(u)+4*x_tl
y = 4*x+sin(u)

end subroutine ex1_tl
```

driver of tangent linear code

```
program drivertlm
  implicit none
  real x_tl, u_tl, y_tl
  real x, u, y

  x = 1.2     ! initial x
  u = 0.5     ! initial u
  x_tl = 0.0 ! define direction in input space
  u_tl = 1.0 !

  call ex1_tl( x, x_tl, u, u_tl, y, y_tl )

  print *,' y = ',y
  print *,' y_tl = ',y_tl
end
```

```
subroutine ex1_tl( x, x_tl, u, u_tl, y, y_tl )
...
end
```

very simple example

ex1.f90

```
subroutine ex1( x, u, y )
  implicit none
  real x, u, y
  y = 4*x + sin(u)
end
```

command line:

taf -f90 -v2 **-reverse** -toplevel ex1 -input x,u -ouput y ex1.f90

generated adjoint code (ex1_ad.f90)

```
subroutine ex1_ad( x, x_ad, u, u_ad, y, y_ad )

implicit none

!==============================================
! declare arguments
!==============================================
real u
real u_ad
real x
real x_ad
real y
real y_ad

!----------------------------------------------
! FUNCTION AND TAPE COMPUTATIONS
!----------------------------------------------
y = 4*x+sin(u)

!----------------------------------------------
! ADJOINT COMPUTATIONS
!----------------------------------------------
u_ad = u_ad+y_ad*cos(u)
x_ad = x_ad+4*y_ad
y_ad = 0.

end subroutine ex1_ad
```

FastOpt

driver of adjoint code

```
program driveradm
  implicit none
  real x_ad, u_ad, y_ad
  real x, u, y

  x = 1.2     ! initial x
  u = 0.5     ! initial u
  x_ad = 0.   ! no other influence
  u_ad = 0.   ! no other influence
  y_ad = 1.   ! just some sensitivity

  call ex1_ad( x, x_ad, u, u_ad, y, y_ad )

  print *,' x_ad = ', x_ad
  print *,' u_ad = ', u_ad

end
```

```
subroutine ex1_ad( x, x_ad, u, u_ad, y, y_ad )
...
end
```

Verification of Derivative code

Compare to finite difference approximation:

FD = (f(x+eps)-f(x))/eps

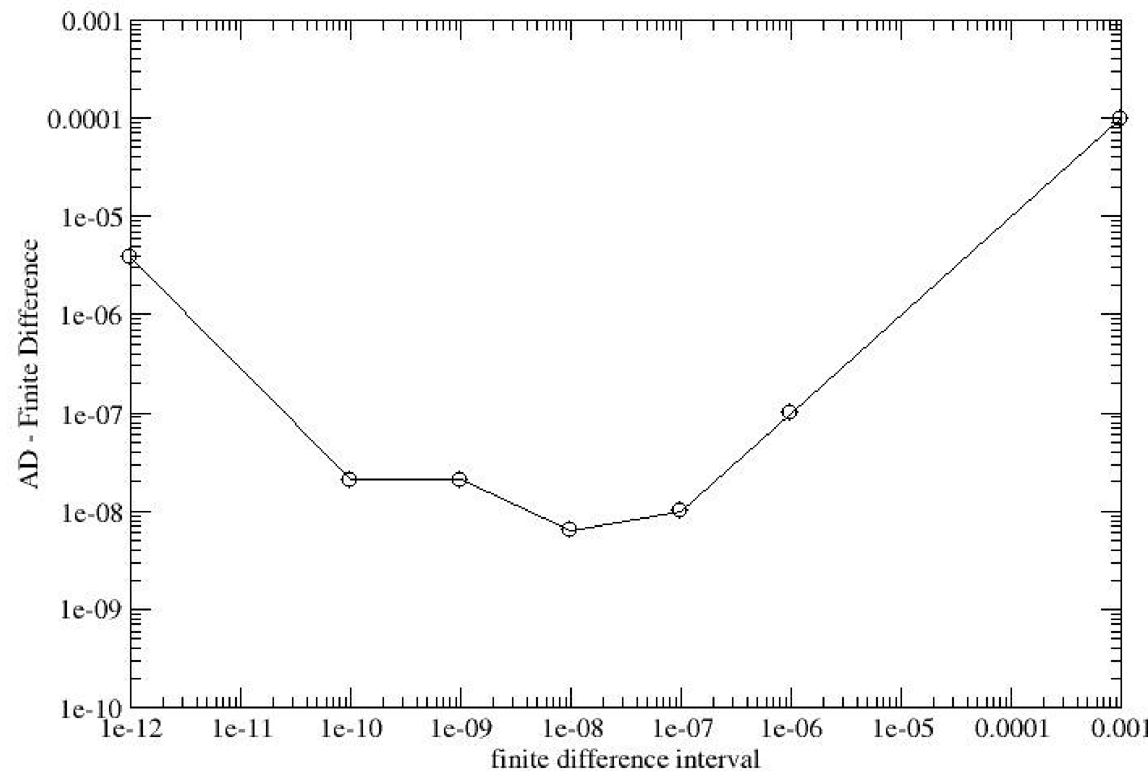
Result depends on eps:

•Too large eps: Non linear terms spoil result

•Too small eps: Rounding error problems:

f(x+eps) = f(x) -> FD = 0.

Example --------------------------->



Exercise

Hand code tangents and adjoints and the respective drivers for:

```
subroutine func( n, x, m, y )
  implicit none
  integer :: n, m
  real    :: x(n), y(m), u, v
  u = cos(x(1))
  v = sqrt(x(2)) + x(1)
  y(1) =  u + v
end subroutine func

program main
  implicit none
  ! dimensions
  integer, parameter :: n = 3
  integer, parameter :: m = 1
  real    :: x(n), y(m)
  ! initialisation
  x(1) = 0.
  x(2) = 1.
  x(3) = 2.
  ! function evaluation
  call func( n, x, m, y )
  ! postprocessing
  print '(a2,3(x,f6.2),a4,f6.2)', 'y(',x,') = ',y
end program main
```