

Automatic Differentiation in Theory and Practice

The slides are available from WWW at
<<http://www.imm.dtu.dk/documents/users/os/fadbad.html>>

Ole Stauning <os@imm.dtu.dk>
Department of Mathematical Modelling
Technical University of Denmark

24 oct 1996

MOTIVATION

- Why use derivatives??
 - It is the most important tool in mathematics.
 - Solving nonlinear equations (Newtons method)
 - Optimization
 - Sensitivity analysis

HOW DO WE OBTAIN DERIVATIVES??

- **Symbolic differentiation:** On expressions. By hand or by a computer algebra system (Maple, Axiom or Mathematica).

PROBLEMS: Some “simple” expressions will lead to complicated derivatives.

- **Numerical differentiation (divided differences):** On a “black box” which evaluates the function.

PROBLEMS: Truncation errors, inaccurate.

HOW DO WE OBTAIN DERIVATIVES??

- If we have an expression or a program which computes function values we can use:

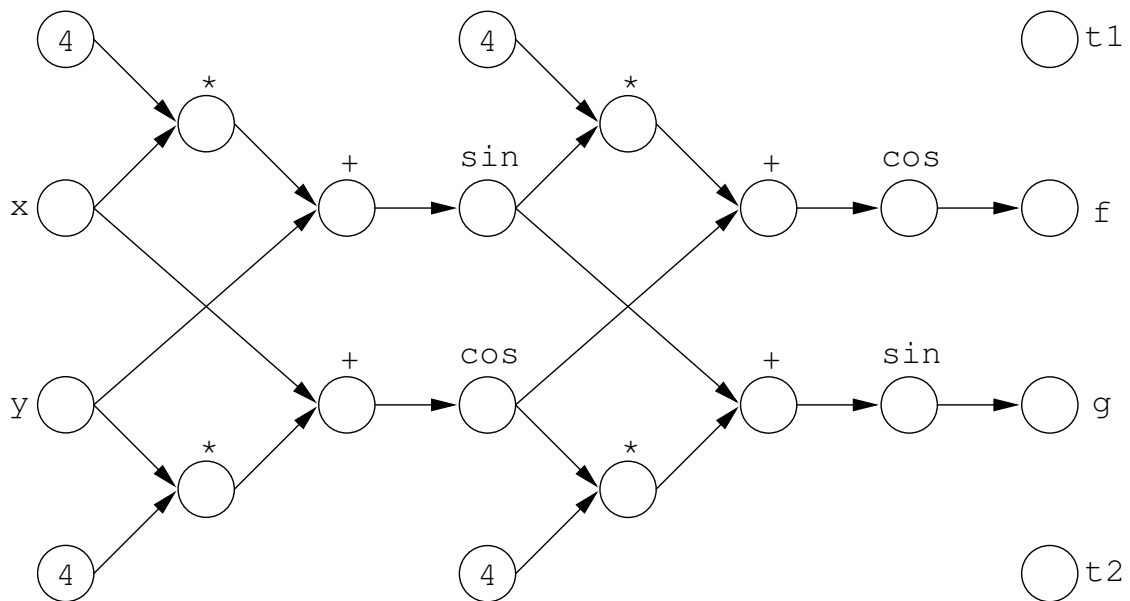
Automatic Differentiation

PROBLEMS: ??????????????????????????????????

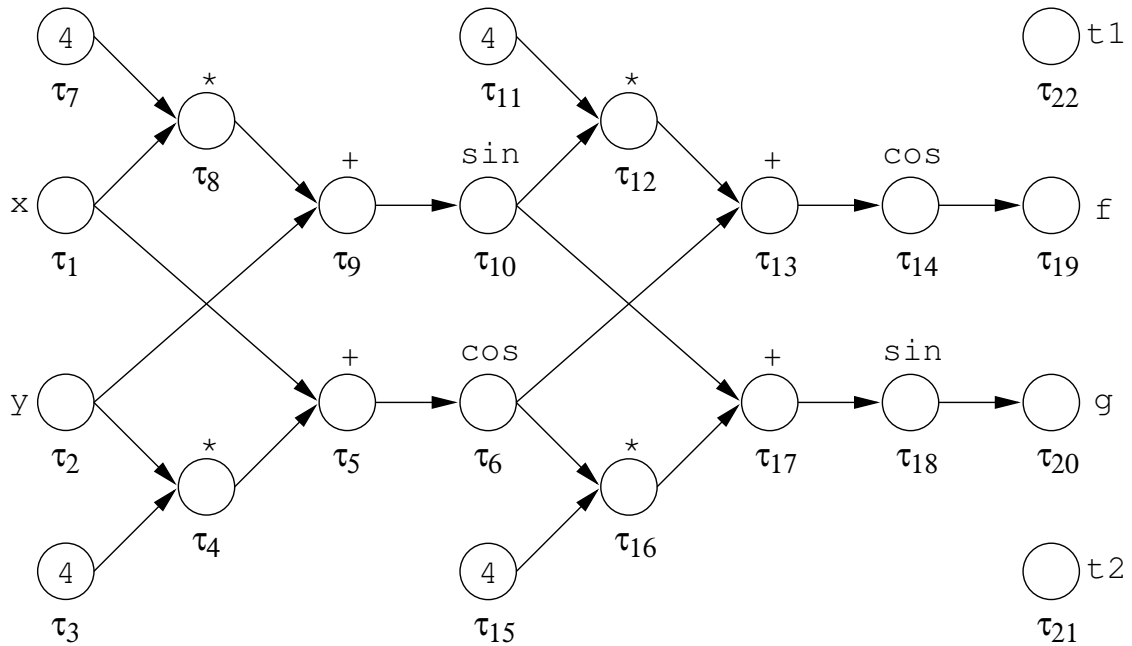
PROBLEM FORMULATION

Consider:

```
f=x; g=y;  
for (int i=1;i<=2;i++) {  
    t1=cos(f+4*g);  
    t2=sin(4*f+g);  
    f=t1; g=t2;  
}  
t1=t2=0;
```



PROBLEM FORMULATION



$\tau_1 = x$	$\tau_{11} = 4$
$\tau_2 = y$	$\tau_{12} = \tau_{11} \cdot \tau_{10}$
$\tau_3 = 4$	$\tau_{13} = \tau_6 + \tau_{12}$
$\tau_4 = \tau_3 \cdot \tau_2$	$\tau_{14} = \cos(\tau_{13})$
$\tau_5 = \tau_1 + \tau_4$	$\tau_{15} = 4$
$\tau_6 = \cos(\tau_5)$	$\tau_{16} = \tau_{15} \cdot \tau_6$
$\tau_7 = 4$	$\tau_{17} = \tau_{16} + \tau_{10}$
$\tau_8 = \tau_7 \cdot \tau_1$	$\tau_{18} = \sin(\tau_{17})$
$\tau_9 = \tau_8 + \tau_2$	$\tau_{19} = \tau_{14}$
$\tau_{10} = \sin(\tau_9)$	$\tau_{20} = \tau_{18}$

PROBLEM FORMULATION

In general:

initialize the values:

$$\tau_i = f_i = x_i, \quad \text{for } i = 1, \dots, m.$$

compute:

for $i = m + 1$ to n ,

$$\tau_i = f_i(\tau_1, \dots, \tau_{i-1}).$$

The “chain rule” for composite functions:

$$\frac{\partial \tau_i}{\partial \tau_j} = \delta_{ij} + \sum_{k=j}^{i-1} \frac{\partial f_i}{\partial \tau_k} \frac{\partial \tau_k}{\partial \tau_j}, \quad \text{where } \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$$

Or in matrix formulation:

$$\mathbf{D}\tau = \mathbf{I} + \mathbf{Df}\mathbf{D}\tau,$$

where

$$\mathbf{Df} = \left\{ \frac{\partial f_i}{\partial \tau_j} \right\}_{i,j=1,\dots,n} = \begin{pmatrix} 0 & \dots & \dots & \dots \\ \frac{\partial f_2}{\partial \tau_1} & 0 & \dots & \dots \\ \frac{\partial f_3}{\partial \tau_1} & \frac{\partial f_3}{\partial \tau_2} & 0 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix},$$

$$\mathbf{D}\tau = \left\{ \frac{\partial \tau_i}{\partial \tau_j} \right\}_{i,j=1,\dots,n} = \begin{pmatrix} 1 & 0 & \dots & \dots \\ \frac{\partial \tau_2}{\partial \tau_1} & 1 & \dots & \dots \\ \frac{\partial \tau_3}{\partial \tau_1} & \frac{\partial \tau_3}{\partial \tau_2} & 1 & \dots \\ \dots & \dots & \dots & \dots \end{pmatrix}.$$

OBTAINING THE MATRIX $D\tau$

$$\begin{aligned}
 D\tau &= I + DfD\tau && \Leftrightarrow \\
 (I - Df)D\tau &= I && \Leftrightarrow && (1) \\
 D\tau &= (I - Df)^{-1} && \Leftrightarrow \\
 D\tau(I - Df) &= I && \Leftrightarrow \\
 (I - Df)^T D\tau^T &= I && (2)
 \end{aligned}$$

From (1) we get: $I =$

$$\begin{pmatrix} 1 & 0 & \dots & & 0 \\ -\frac{\partial f_2}{\partial \tau_1} & 1 & \ddots & & 0 \\ -\frac{\partial f_3}{\partial \tau_1} & -\frac{\partial f_3}{\partial \tau_2} & 1 & \ddots & 0 \\ \dots & \ddots & \ddots & \ddots & \vdots \\ -\frac{\partial f_n}{\partial \tau_1} & -\frac{\partial f_n}{\partial \tau_2} & \dots & -\frac{\partial f_n}{\partial \tau_{n-1}} & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \dots & & 0 \\ \frac{\partial \tau_2}{\partial \tau_1} & 1 & \ddots & & 0 \\ \frac{\partial \tau_3}{\partial \tau_1} & \frac{\partial \tau_3}{\partial \tau_2} & 1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \frac{\partial \tau_n}{\partial \tau_1} & \frac{\partial \tau_n}{\partial \tau_2} & \dots & \frac{\partial \tau_n}{\partial \tau_{n-1}} & 1 \end{pmatrix}$$

From (2) we get: $I =$

$$\begin{pmatrix} 1 & -\frac{\partial f_2}{\partial \tau_1} & \dots & -\frac{\partial f_{n-1}}{\partial \tau_1} & -\frac{\partial f_n}{\partial \tau_1} \\ \vdots & \ddots & \ddots & \ddots & \dots \\ 0 & \ddots & 1 & -\frac{\partial f_{n-1}}{\partial \tau_{n-2}} & -\frac{\partial f_n}{\partial \tau_{n-2}} \\ 0 & \dots & \dots & 1 & -\frac{\partial f_n}{\partial \tau_{n-1}} \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \frac{\partial \tau_2}{\partial \tau_1} & \dots & \frac{\partial \tau_{n-1}}{\partial \tau_1} & \frac{\partial \tau_n}{\partial \tau_1} \\ \vdots & \ddots & \ddots & \ddots & \dots \\ 0 & \dots & 1 & \frac{\partial \tau_{n-1}}{\partial \tau_{n-2}} & \frac{\partial \tau_n}{\partial \tau_{n-2}} \\ 0 & \dots & \dots & 1 & \frac{\partial \tau_n}{\partial \tau_{n-1}} \\ 0 & \dots & 0 & 0 & 1 \end{pmatrix}$$

THE TWO METHODS

Let a_i be the arity of f_i and define the map

$$\kappa_i : \{1, \dots, a_i\} \mapsto \mathcal{I}_i \subset \{1, \dots, i-1\},$$

so that

$$\tau_i = f_i(\tau_{\kappa_i 1}, \dots, \tau_{\kappa_i a_i}).$$

The **FORWARD** method is given by:

initialize the values:

$$\tau_i = x_i, \hat{\tau}_{ij} = \delta_{ij}, \text{ for } i, j = 1, \dots, m.$$

compute:

for $i = m + 1$ to n ,

$$\tau_i = f_i(\tau_{\kappa_i 1}, \dots, \tau_{\kappa_i a_i}),$$

$$\hat{\tau}_{i,j} = \sum_{k=1}^{a_i} \frac{\partial f_i}{\partial \tau_{\kappa_i k}} \hat{\tau}_{\kappa_i k, j} \text{ for } j = 1, \dots, m.$$

$\hat{\tau}_{i,j} = \frac{\partial \tau_i}{\partial \tau_j}$ for $i = 1, \dots, n, j = 1, \dots, m$ after the algorithm has stopped.

EXAMPLE OF THE FORWARD METHOD

$$\begin{array}{ll}
 \tau_1 = x & \tau_{11} = 4 \\
 \hat{\tau}_{1,1} = 1 & \hat{\tau}_{11,1} = 0 \\
 \tau_2 = y & \tau_{12} = \tau_{11} \cdot \tau_{10} \\
 \hat{\tau}_{2,1} = 0 & \hat{\tau}_{11,1} = \tau_{10} \cdot \hat{\tau}_{11,1} + \tau_{11} \cdot \hat{\tau}_{10,1} \\
 \tau_3 = 4 & \tau_{13} = \tau_6 + \tau_{12} \\
 \hat{\tau}_{3,1} = 0 & \hat{\tau}_{13,1} = 1 \cdot \hat{\tau}_{6,1} + 1 \cdot \hat{\tau}_{12,1} \\
 \tau_4 = \tau_3 \cdot \tau_2 & \tau_{14} = \cos(\tau_{13}) \\
 \hat{\tau}_{4,1} = \tau_2 \cdot \hat{\tau}_{3,1} + \tau_3 \cdot \hat{\tau}_{2,1} & \hat{\tau}_{14,1} = -\sin(\tau_{13}) \cdot \hat{\tau}_{13,1} \\
 \tau_5 = \tau_1 + \tau_4 & \tau_{15} = 4 \\
 \hat{\tau}_{5,1} = 1 \cdot \hat{\tau}_{1,1} + 1 \cdot \hat{\tau}_{4,1} & \hat{\tau}_{15,1} = 0 \\
 \tau_6 = \cos(\tau_5) & \tau_{16} = \tau_{15} \cdot \tau_6 \\
 \hat{\tau}_{6,1} = -\sin(\tau_5) \cdot \hat{\tau}_{5,1} & \hat{\tau}_{16,1} = \tau_6 \cdot \hat{\tau}_{15,1} + \tau_{15} \cdot \hat{\tau}_{6,1} \\
 \tau_7 = 4 & \tau_{17} = \tau_{16} + \tau_{10} \\
 \hat{\tau}_{7,1} = 0 & \hat{\tau}_{17,1} = 1 \cdot \hat{\tau}_{16,1} + 1 \cdot \hat{\tau}_{10,1} \\
 \tau_8 = \tau_7 \cdot \tau_1 & \tau_{18} = \sin(\tau_{17}) \\
 \hat{\tau}_{8,1} = \tau_1 \cdot \hat{\tau}_{7,1} + \tau_7 \cdot \hat{\tau}_{1,1} & \hat{\tau}_{18,1} = \cos(\tau_{17}) \cdot \hat{\tau}_{17,1} \\
 \tau_9 = \tau_8 + \tau_2 & \tau_{19} = \tau_{14} \\
 \hat{\tau}_{9,1} = 1 \cdot \hat{\tau}_{8,1} + 1 \cdot \hat{\tau}_{2,1} & \hat{\tau}_{19,1} = \hat{\tau}_{14,1} \\
 \tau_{10} = \sin(\tau_9) & \tau_{20} = \tau_{18} \\
 \hat{\tau}_{10,1} = \cos(\tau_9) \cdot \hat{\tau}_{9,1} & \hat{\tau}_{20,1} = \hat{\tau}_{18,1}
 \end{array}$$

THE TWO METHODS

Let \mathcal{D} be the set of indices of the dependent variables which is to be differentiated.

The **BACKWARD** method is given by:

initialize the forward sweep:

$$\tau_i = x_i, \text{ for } i = 1, \dots, m.$$

forward sweep (function evaluation):

for $i = m + 1$ to n ,

$$\tau_i = f_i(\tau_{\kappa_i 1}, \dots, \tau_{\kappa_i a_i}),$$

initialize the reverse sweep:

$$\hat{\tau}_{i,j} = \delta_{ij} \text{ for } i \in \mathcal{D}, j = 1, \dots, n.$$

reverse sweep (function differentiation):

for $j = n$ downto $m + 1$,

$$\hat{\tau}_{i,\kappa_j k} = \hat{\tau}_{i,\kappa_j k} + \frac{\partial f_j}{\partial \tau_{\kappa_j k}} \hat{\tau}_{i,j} \text{ for } i \in \mathcal{D}, k = 1, \dots, a_j.$$

$\hat{\tau}_{i,j} = \frac{\partial \tau_i}{\partial \tau_j}$ for $i \in \mathcal{D}, j = 1, \dots, n$ after the algorithm has stopped.

EXAMPLE OF THE BACKWARD METHOD

$\tau_1 = x$	$\hat{\tau}_{20,18} + = 1 \cdot \hat{\tau}_{20,20}$
$\tau_2 = y$	$\hat{\tau}_{20,14} + = 1 \cdot \hat{\tau}_{20,19}$
$\tau_3 = 4$	$\hat{\tau}_{20,17} + = \cos(\tau_{17}) \cdot \hat{\tau}_{20,18}$
$\tau_4 = \tau_3 \cdot \tau_2$	$\hat{\tau}_{20,16} + = 1 \cdot \hat{\tau}_{20,17}$
$\tau_5 = \tau_1 + \tau_4$	$\hat{\tau}_{20,10} + = 1 \cdot \hat{\tau}_{20,17}$
$\tau_6 = \cos(\tau_5)$	$\hat{\tau}_{20,15} + = \tau_6 \cdot \hat{\tau}_{20,16}$
$\tau_7 = 4$	$\hat{\tau}_{20,6} + = \tau_{15} \cdot \hat{\tau}_{20,16}$
$\tau_8 = \tau_7 \cdot \tau_1$	$\hat{\tau}_{20,13} + = -\sin(\tau_{13}) \cdot \hat{\tau}_{20,14}$
$\tau_9 = \tau_8 + \tau_2$	$\hat{\tau}_{20,6} + = 1 \cdot \hat{\tau}_{20,13}$
$\tau_{10} = \sin(\tau_9)$	$\hat{\tau}_{20,12} + = 1 \cdot \hat{\tau}_{20,13}$
$\tau_{11} = 4$	$\hat{\tau}_{20,11} + = \tau_{10} \cdot \hat{\tau}_{20,12}$
$\tau_{12} = \tau_{11} \cdot \tau_{10}$	$\hat{\tau}_{20,10} + = \tau_{11} \cdot \hat{\tau}_{20,12}$
$\tau_{13} = \tau_6 + \tau_{12}$	$\hat{\tau}_{20,9} + = \cos(\tau_9) \cdot \hat{\tau}_{20,10}$
$\tau_{14} = \cos(\tau_{13})$	$\hat{\tau}_{20,8} + = 1 \cdot \hat{\tau}_{20,9}$
$\tau_{15} = 4$	$\hat{\tau}_{20,2} + = 1 \cdot \hat{\tau}_{20,9}$
$\tau_{16} = \tau_{15} \cdot \tau_6$	$\hat{\tau}_{20,7} + = \tau_1 \cdot \hat{\tau}_{20,8}$
$\tau_{17} = \tau_{16} + \tau_{10}$	$\hat{\tau}_{20,1} + = \tau_7 \cdot \hat{\tau}_{20,8}$
$\tau_{18} = \sin(\tau_{17})$	$\hat{\tau}_{20,5} + = -\sin(\tau_5) \cdot \hat{\tau}_{20,6}$
$\tau_{19} = \tau_{14}$	$\hat{\tau}_{20,1} + = 1 \cdot \hat{\tau}_{20,5}$
$\tau_{20} = \tau_{18}$	$\hat{\tau}_{20,4} + = 1 \cdot \hat{\tau}_{20,5}$
$\hat{\tau}_{20,20} = 1$	$\hat{\tau}_{20,3} + = \tau_2 \cdot \hat{\tau}_{20,4}$
$\hat{\tau}_{20,i} = 0, \text{ for } i < 20$	$\hat{\tau}_{20,2} + = \tau_3 \cdot \hat{\tau}_{20,4}$

THE C++ PACKAGE “FADBAD”

FADBAD is available from the WWW-page:

<<http://www.imm.dtu.dk/documents/users/os/fadbad.html>>

Main strategy:

- **Transparency:** Existing programs is easy to differentiate
- **Other arithmetics:** The type of arithmetics can be changed from IEEE double to other arithmetic types, e.g. INTERVAL
- **Flexibility:** Both forward- and backward methods is available and the usage of the two methods has a minimal difference.
- **Higher order derivatives:** By overloading the package itself.

THE C++ PACKAGE “FADBAD”

Simple function $\mathbb{R}^2 \rightarrow \mathbb{R}^2$:

```
#define x in[0]
#define y in[1]
#define f out[0]
#define g out[1]
int p=2;
void func(double *in,double *out){
    double t1,t2;
    f=x;g=y;
    for(int i=1;i<=p;i++){
        t1=cos(f+4*g);
        t2=sin(4*f+g);
        f=t1;g=t2;
    }
}
#undef x
#undef y
#undef f
#undef g
```

THE C++ PACKAGE "FADBAD"

Newton-Raphson's method using forward-double:

```
#include "Fdouble"

void func(Fdouble *in,Fdouble *out){
    // .. function evaluation deleted ..
}

void newton(double *val){
    Fdouble in[2],out[2];
    double det,dfdx,dfdy,dgdx,dgdy;
    do{
        in[0]=val[0];in[1]=val[1];
        in[0].diff(0,2); // call diff on the independent
        in[1].diff(1,2); // variables of func.
        func(in,out);
        dfdx=out[0].d(0);dfdy=out[0].d(1); // df/dx ; df/dy
        dgdx=out[1].d(0);dgdy=out[1].d(1); // dg/dx ; dg/dy

        det=dfdx*dgdy-dgdx*dfdy;
        val[0]-=(dgdy*out[0].x()-dfdy*out[1].x())/det;
        val[1]-=(dfdx*out[1].x()-dgdx*out[0].x())/det;
    }while(out[0]*out[0]+out[1]*out[1]>1e-6);
}
```

THE C++ PACKAGE "FADBAD"

Differentiating Newton-Raphson's method using forward-backward-double:

```
#include "Bdouble"
#include "FBdouble"

void func(FBdouble *in,FBdouble *out){
    // .. function evaluation deleted ..
}
void newton(Bdouble *val){
    // .. Newton iteration deleted ..
}

int main(){
    Bdouble in[2],out[2];
    in[0]=.4;in[1]=.5;           // initial values
    out[0]=in[0];out[1]=in[1]; // independent variables
    newton(out);
    out[0].diff(0,2); // Differentiate the dependent
    out[1].diff(1,2); // variables

    out[0].x(); // x-result of the Newton iterations
    out[1].x(); // y-result of the Newton iterations
    in[0].d(0);in[0].d(1); // derivatives wrt. in[0]
    in[1].d(0);in[1].d(1); // derivatives wrt. in[1]

    return 1;
}
```