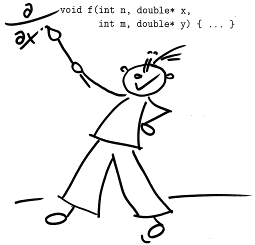
The Art of Differentiating Computer Programs

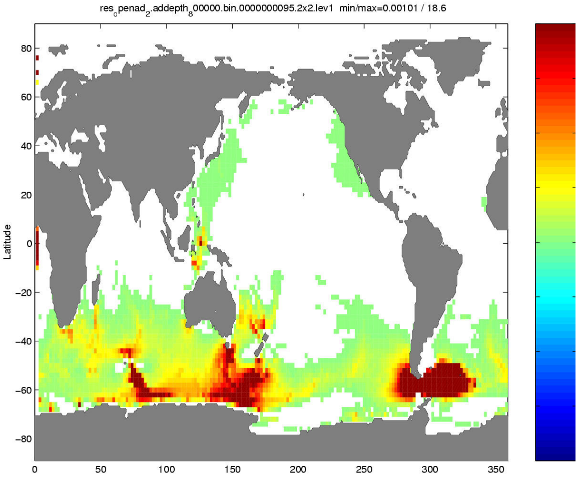
using Algorithmic Differentiation (AD)

Uwe Naumann

LuFG Informatik 12
Software and **T**ools for **C**omputational **E**ngineering
RWTH Aachen
naumann@stce.rwth-aachen.de
www.stce.rwth-aachen.de



Motivation: Numerical Simulation e.g., MITgcm



Formally

Given: Numerical simulation program

$$F : \mathbb{R}^n \to \mathbb{R}^m, \quad \mathbf{y} = F(\mathbf{x})$$

Wanted: Sensitivity of outputs $\mathbf{y} \in \mathbb{R}^m$ on inputs $\mathbf{x} \in \mathbb{R}^n$ (Jacobian $\nabla F(\mathbf{x})$)

$$\nabla F(\mathbf{x}) \equiv \left(\frac{\partial y_j}{\partial x_i} \right)_{i=0,\ldots,n-1}^{j=0,\ldots,m-1}$$

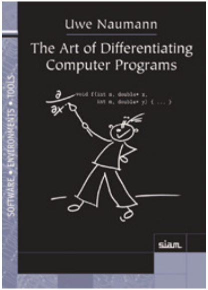
$y = f(x)$; effect of perturbation in x depending on $y' \equiv \frac{\partial y}{\partial x}$

- for large $y' > 0 \rightarrow$ large increase in y (instable)
- for small $y' > 0 \rightarrow$ small increase in y (stable)
- for large $y' < 0 \rightarrow$ large decrease in y (instable)
- for small $y' < 0 \rightarrow$ small decrease in y (stable)
- for $y' = 0 \rightarrow$ invariant

Outline

- 3 lectures
- 2 tutorials
- see tutorial sheet for contents

Literature



U. Naumann:
The Art of Differentiating Computer Programs.
An Introduction to Algorithmic Differentiation
Number 24 of Software, Environments, and Tools Series, SIAM, 2012.

naumann@stce.rwth-aachen.de

More ...

- Computational Differentiation (V3Ü1, Ba, WS)
- Combinatorial Problems in Scientific Computing (V2Ü2, Ma, SS)

Our Objective

For a given numerical simulation program, e.g.,

```
void f(int n, double *x, int m, double *y) {
  y[0]=x[0]*(x[0]+x[1]);
  y[1]=sin(x[1]);
}
```

build a program that can be used to compute the sensitivites of all outputs with respect to all inputs (Jacobian matrix).

Specific Objective: Tangent-Linear Code

$$\dot{\mathbf{y}} = \nabla F(\mathbf{x}) \cdot \dot{\mathbf{x}}$$
$$\mathbf{y} = F(\mathbf{x})$$

```
void f(int n, double *x, int m, double *y)
```

becomes

```
void df(int n, double *x, double *dx,
        int m, double *y, double *dy)
```

and computes product of the Jacobian with $\dot{\mathbf{x}}$ in $\dot{\mathbf{y}}$; not the Jacobian itself.

Example

```
void f(int n, double *x, int m, double *y) {
  y[0]=x[0]*(x[0]+x[1]);
  y[1]=sin(x[1]);
}
```

- Single Assignment Code (SAC)
- Directed Acyclic Graph (AST, DAG)
- Linearized SAC
- Linearized AST, DAG
- Local Tangent-Linear Code
- Chain Rule

Jacobian
Computation and Validation

```
void f(int n, double *x, int m, double *y) {
  y[0]=x[0]*(x[0]+x[1]);
  y[1]=sin(x[1]);
}
```

- driver for tangent-linear code
- validation by finited differences

Validation through Approximation
Forward Finite Differences

Let $D \subseteq \mathbb{R}^n$ be an open domain and $F : D \rightarrow \mathbb{R}^m$ such that

$$F = \begin{pmatrix} F_0 \\ \vdots \\ F_{m-1} \end{pmatrix} \quad .$$

A *forward finite difference* approximation of the ith column of the Jacobian ∇F at point $\mathbf{x}^0$ is computed as

$$\frac{\partial F}{\partial x_i}(\mathbf{x}^0) \equiv \begin{pmatrix} \frac{\partial F_0}{\partial x_i}(\mathbf{x}^0) \\ \vdots \\ \frac{\partial F_{m-1}}{\partial x_i}(\mathbf{x}^0) \end{pmatrix} \approx_1 \frac{F(\mathbf{x}^0 + \mathbf{e}_i \cdot h) - F(\mathbf{x})}{h} \tag{1}$$

The *ith Cartesian basis vector in* $\mathbb{R}^n$ is denoted by $\mathbf{e}_i$.

However ...

Consider the approximation of the first derivative of $y = f(x) = x$ in single precision IEEE floating-point arithmetic on a 64 bit machine at $x = 10^6$ by the forward finite quotient

$$\nabla f(x) \approx \frac{f(x+h) - f(x)}{h}$$

with $h = 0.1$. Obviously, $\nabla f(x) = 1$ independent of x. The code

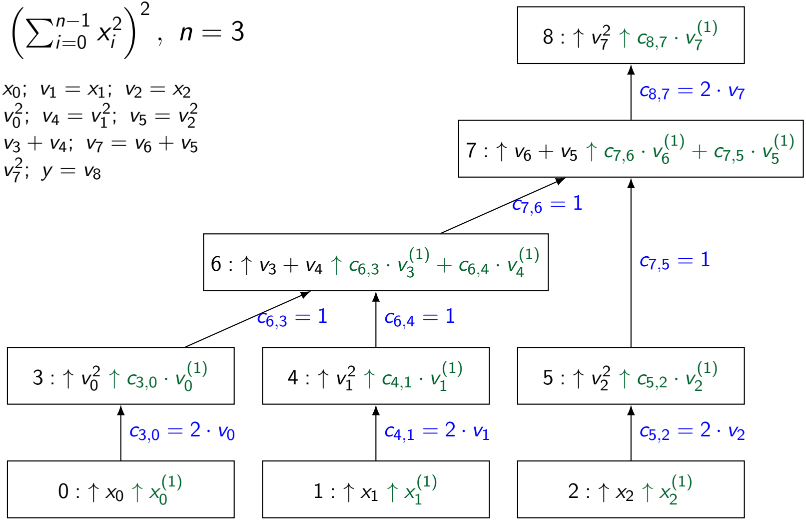
```
...
float x=1e6, h=1e-1;
cout << "(f(x+h)-f(x))/h=" << (x+h-x)/h << endl;
...
```

returns 1.25.

Tangent-Linear Code by Forward Mode AD

$$y = \left(\sum_{i=0}^{n-1} x_i^2\right)^2, \quad n = 3$$

$v_0 = x_0;\ v_1 = x_1;\ v_2 = x_2$
$v_3 = v_0^2;\ v_4 = v_1^2;\ v_5 = v_2^2$
$v_6 = v_3 + v_4;\ v_7 = v_6 + v_5$
$v_8 = v_7^2;\ y = v_8$



Further Motivation
Nonlinear Optimization

Consider the nonlinear programming problem (NLP)

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x})$$

where, for example,

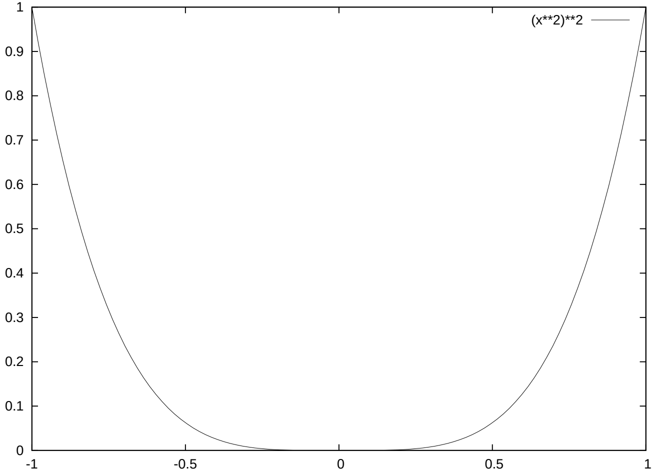
$$f(\mathbf{x}) = \left(\sum_{i=0}^{n-1} x_i^2\right)^2 \tag{2}$$

is implemented as

```
void f(int n, double *x, double &y)
{
  y=0;
  for (int i=0;i<n;i++) y=y+x[i]*x[i];
  y=y*y;
}
```

The function has a global minimum at $\mathbf{x} = 0$.

Nonlinear Optimization



... computes

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \cdot \nabla f(\mathbf{x}^k) \quad .$$

for some suitable starting value $\mathbf{x}^0 = (x_i^0)_{i=0,\ldots,n-1}$ and with a step length $\alpha_k > 0$, where $\nabla f(\mathbf{x}^k)$ denotes the gradient of f at the current iterate.

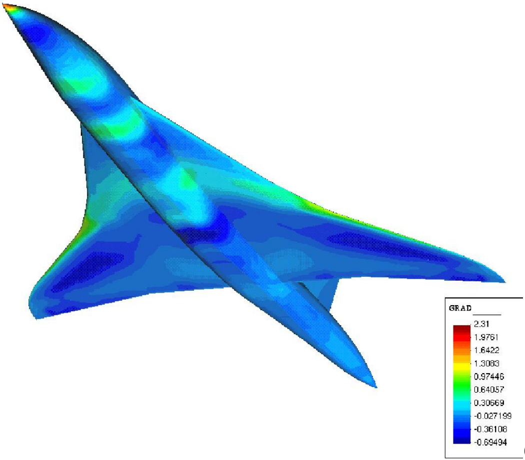
The step length $\alpha_k > 0$ is, for example, chosen by recursive bisection on α_k starting from $\alpha_k = 1$ $(0.5, 0.25, \ldots)$ and such that a decrease in the objective value is ensured

More Applications

...

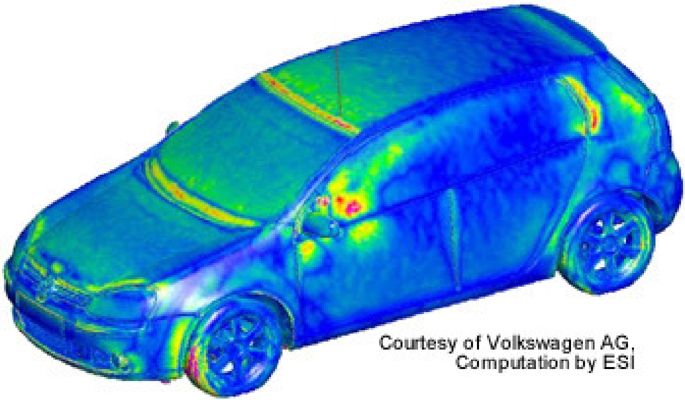

1 Month LIBOR (USD) 0.44 0.0000 (0.00%) 6:24 AM EDT
1d 5d 1m 3m 6m 1y 2y 5y 10y Full
7.35
5.22
3.09
0.96
Nov 2008
Dec
Jan
Feb
wikinvest

©BBA


GRAD
2.31
1.9761
1.6422
1.3083
0.97446
0.64057
0.30669
-0.027199
-0.36108
-0.69494

©INRIA

Automative Engineering



dcc (version 0.9)

See

`www.siam.org/books/se24`

for

- source
- use guide
- Windows version

By The Way ...

It also works for, e.g.

```
void f(int n, double *x, double &y) {
  int i=0;
  while (i<n) {
    if (i==0) { y=x[i]*x[i]; }
    else { y=y+x[i]*x[i]; }
    i=i+1;
  }
  y=y*y;
}
```

as well as for interprocedural code etc.