Introduction to Automatic/Algorithmic Differentiation (AD)

Computational Mathematics Group
CCES, RWTH Aachen University

May 8, 2012

Evaluating Derivatives

Automatic Differentiation (AD)

AD by use of `Tapenade`

Examples

Evaluating Derivatives - Various Approaches

- Hand Differentiation
- Symbolic Differentiation
- Finite Differences
- Complex Taylor Series Expansion (CTSE) Method
- Automatic or Algorithmic Differentiation

Evaluating Derivatives ..

- Hand Differentiation
    - Analytical expression for the derivative is derived manually and then coded into a computer program
    - Any small change in the input function requires complete redoing of all the hand differentiation and coding
    - Laborious, prone to errors and practically infeasible

Evaluating Derivatives ..

- Hand Differentiation
  - Analytical expression for the derivative is derived manually and then coded into a computer program
  - Any small change in the input function requires complete redoing of all the hand differentiation and coding
  - Laborious, prone to errors and practically infeasible
- Symbolic Differentiation
  - Analytical derivatives are obtained using computer algebra systems like Maple or Mathematica
  - Not suitable for complex functions (with loops, branches)

Evaluating Derivatives ..

- Finite Differences
  - The gradient of the objective function I w.r.t. any control/design variable α_i can evaluated by

$$\frac{\partial I}{\partial \alpha_i} = \frac{I\left(\alpha + \Delta\alpha_i\right) - I\left(\alpha - \Delta\alpha_i\right)}{2\Delta\alpha} + O\left(\Delta\alpha_i\right)^2$$

  - Very easy to implement
  - Small step sizes lead to subtractive cancellation errors
  - For example
    - $I\left(\alpha + \Delta\alpha_i\right) = 0.49263278105238\textcolor{red}{9}$
    - $I\left(\alpha - \Delta\alpha_i\right) = 0.492632781052334$
    - $I\left(\alpha + \Delta\alpha_i\right) - I\left(\alpha - \Delta\alpha_i\right) = 0.000000000000055$
  - Lacks robustness
  - Requires $2N$ flow solutions to compute the gradients w.r.t. N control variables. Computationally very expensive if N is large

Evaluating Derivatives ..

- Complex Taylor Series Expansion Method

    - The Taylor series expansion of I with an imaginary perturbation $i\Delta\alpha$ is given by

    $$I\left(\alpha + i\Delta\alpha\right) = I\left(\alpha\right) + i\Delta\alpha\frac{\partial I}{\partial\alpha} - \frac{\left(\Delta\alpha\right)^2}{2}\frac{\partial^2 I}{\partial\alpha^2} + O\left(\Delta\alpha\right)^3$$

    Equating the imaginary parts in the above equation, we get

    $$\frac{\partial I}{\partial\alpha} = \frac{Im\left[I\left(\alpha + i\Delta\alpha\right)\right]}{\Delta\alpha} + O\left(\Delta\alpha\right)^2$$

    - Very robust and has no cancellation errors

    - Requires N flow solutions (with complex arithmetic) for N control variables

Automatic/Algorithmic Differentiation

- Input: A computer program for a given function
- Output: Another program or augments the original program that computes the analytical derivatives along with the original program
- Programming languages: FORTRAN/C/C++
- Interprets the input program as a sequence of simple elementary operations (additions, multiplications, intrinsic functions, etc)
- Assumes that the input program is piecewise differentiable
- The derivative program is obtained by applying the chain rule of differentiation sequentially to each of these elementary operations

Automatic/Algorithmic Differentiation ..

- Does not incur any truncation errors
- Derivative values are accurate to machine precision
- Generates the derivative program with very little effort from the user irrespective of the complexity of the input program
- Many functions that are not analytically differentiable are algorithmically differentiable

Some terminology used in AD

A computer program for a given function consists of the following groups of variables

- Dependent variables: Variables whose derivatives are to be evaluated.
- Independent variables: Variables with respect to which the derivatives are to be computed. One has to supply the values of these variables.
- Intermediate variables: Variables which are computed from independents and whose values are used to find the dependents.

Modes of AD

AD can be performed in two ways

- Forward or Tangent mode
- Reverse or Adjoint mode

These modes are distinguished by the way the chain rule of differentiation is performed.
To explain this, consider a function f, given by

$$f = f_1\left(f_2\left(f_3\left(....\left(f_n\left(x\right)\right)\right)\right)\right)$$

The gradient of f is given by

$$\frac{df}{dx} = \frac{df_1}{dx}\frac{df_2}{dx}\frac{df_3}{dx}......\frac{df_n}{dx}$$

- In forward mode, the chain rule of differentiation propagates from right to left. That is from df_n/dx to df_1/dx
- In reverse mode, the direction of propogation is from left to right

Example 1

Consider the test function $f(x_1, x_2) = x_1 x_2 + sin(x_1) + e^{x_2}$. AD tools read this function as a sequence of elementary operations

$$
\begin{aligned}
t_1 &= x_1 \\
t_2 &= x_2 \\
t_3 &= x_1 * x_2 = t_1 * t_2 \\
t_4 &= sin(x_1) = sin(t_1) \\
t_5 &= e^{x_2} = e^{t_2} \\
t_6 &= t_3 + t_4 \\
t_7 &= t_5 + t_6 = f
\end{aligned}
$$

- Independent variables: $t_1 = x_1$ and $t_2 = x_2$
- Dependent variable: $t_7 = f$
- Intermediate variables: t_3, t_4, t_5 and t_6

Forward mode AD

- Consider a function $f(x) = y$. The derivatives of x and y are denoted by $\dot{x}$ and $\dot{y}$. They are defined as

$$\dot{x} = \frac{\partial x}{\partial x} = 1 \ \textbf{and} \ \dot{y} = \frac{\partial y}{\partial x} = \frac{\partial f}{\partial x}$$

- Another function $f(x_1, x_2) = y$. The derivatives are given by

$$\dot{x_1} = \left(\frac{\partial x_1}{\partial x_1}, \frac{\partial x_1}{\partial x_2}\right) = (1, 0)$$
$$\dot{x_2} = \left(\frac{\partial x_2}{\partial x_1}, \frac{\partial x_2}{\partial x_2}\right) = (0, 1)$$
$$\dot{y} = \left(\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}\right)$$

- In forward mode AD the derivatives are computed in the same direction as the propagation of the primals

Forward mode AD: Example 1

Input: Given function f

$$
\begin{aligned}
t_1 &= x_1 \\
t_2 &= x_2 \\
t_3 &= x_1 * x_2 = t_1 * t_2 \\
t_4 &= sin\,(x_1) = sin\,(t_1) \\
t_5 &= e^{x_2} = e^{t_2} \\
t_6 &= t_3 + t_4 \\
t_7 &= t_5 + t_6 = f
\end{aligned}
$$

Output: $\dot{f}$ (derivative of f)

$$
\begin{aligned}
\dot{t}_1 &= \dot{x}_1 \\
\dot{t}_2 &= \dot{x}_2 \\
\dot{t}_3 &= t_1 * \dot{t}_2 + \dot{t}_1 * t_2 \\
\dot{t}_4 &= cos\,(t_1) * \dot{t}_1 \\
\dot{t}_5 &= e^{t_2} * \dot{t}_2 \\
\dot{t}_6 &= \dot{t}_3 + \dot{t}_4 \\
\dot{t}_7 &= \dot{t}_5 + \dot{t}_6
\end{aligned}
$$

Given the values of x_1 and x_2, the derivatives are obtained by

- Initialising $(\dot{x}_1, \dot{x}_2) = (1, 0)$, we get $\frac{\partial f}{\partial x_1}$
- Initialising $(\dot{x}_1, \dot{x}_2) = (0, 1)$, we get $\frac{\partial f}{\partial x_2}$

Forward mode AD: Example 1

Input: Given function f

$$t_1 = x_1$$
$$t_2 = x_2$$
$$t_3 = x_1 * x_2 = t_1 * t_2$$
$$t_4 = sin(x_1) = sin(t_1)$$
$$t_5 = e^{x_2} = e^{t_2}$$
$$t_6 = t_3 + t_4$$
$$t_7 = t_5 + t_6 = f$$

Output: $\dot{f}$ (derivative of f)

$$\dot{t}_1 = \dot{x}_1$$
$$\dot{t}_2 = \dot{x}_2$$
$$\dot{t}_3 = t_1 * \dot{t}_2 + \dot{t}_1 * t_2$$
$$\dot{t}_4 = cos(t_1) * \dot{t}_1$$
$$\dot{t}_5 = e^{t_2} * \dot{t}_2$$
$$\dot{t}_6 = \dot{t}_3 + \dot{t}_4$$
$$\dot{t}_7 = \dot{t}_5 + \dot{t}_6$$

Given the values of x_1 and x_2, the derivatives are obtained by

- Initialising $(\dot{x}_1, \dot{x}_2) = (1, 0)$, we get $\frac{\partial f}{\partial x_1}$
- Initialising $(\dot{x}_1, \dot{x}_2) = (0, 1)$, we get $\frac{\partial f}{\partial x_2}$

Computational cost of evaluating the gradient vector is linearly dependent on the number of independent variables

Forward mode AD: Example 2

Consider the functions $f_1(x) = sin(x) + log(x)$ and $f_2(x) = cos(x) + e^x$

Input: Functions f_1, f_2

$$
\begin{aligned}
t_1 &= x \\
t_2 &= sin(x) = sin(t_1) \\
t_3 &= log(x) = log(t_1) \\
t_4 &= cos(x) = cos(t_1) \\
t_5 &= e^x = e^{t_1} \\
t_6 &= t_2 + t_3 = f_1 \\
t_7 &= t_4 + t_5 = f_2
\end{aligned}
$$

Forward mode AD: $\dot{f}_1$, $\dot{f}_2$

$$
\begin{aligned}
\dot{t}_1 &= \dot{x} \\
\dot{t}_2 &= cos(t_1)\,\dot{t}_1 \\
\dot{t}_3 &= (1/t_1) * \dot{t}_1 \\
\dot{t}_4 &= -sin(t_1) * \dot{t}_1 \\
\dot{t}_5 &= e^{t_1} * \dot{t}_1 \\
\dot{t}_6 &= \dot{t}_2 + \dot{t}_3 = \dot{f}_1 \\
\dot{t}_7 &= \dot{t}_4 + \dot{t}_5 = \dot{f}_2
\end{aligned}
$$

Forward mode AD: Example 2

Consider the functions $f_1(x) = sin(x) + log(x)$ and $f_2(x) = cos(x) + e^x$

Input: Functions f_1, f_2

$$
\begin{aligned}
t_1 &= x \\
t_2 &= sin(x) = sin(t_1) \\
t_3 &= log(x) = log(t_1) \\
t_4 &= cos(x) = cos(t_1) \\
t_5 &= e^x = e^{t_1} \\
t_6 &= t_2 + t_3 = f_1 \\
t_7 &= t_4 + t_5 = f_2
\end{aligned}
$$

Forward mode AD: $\dot{f}_1$, $\dot{f}_2$

$$
\begin{aligned}
\dot{t}_1 &= \dot{x} \\
\dot{t}_2 &= cos(t_1)\,\dot{t}_1 \\
\dot{t}_3 &= (1/t_1) * \dot{t}_1 \\
\dot{t}_4 &= -sin(t_1) * \dot{t}_1 \\
\dot{t}_5 &= e^{t_1} * \dot{t}_1 \\
\dot{t}_6 &= \dot{t}_2 + \dot{t}_3 = \dot{f}_1 \\
\dot{t}_7 &= \dot{t}_4 + \dot{t}_5 = \dot{f}_2
\end{aligned}
$$

Given x and $\dot{x}$, forward mode AD gives the gradients $\dot{f}_1$ and $\dot{f}_2$ in one sweep

Forward mode AD ..

For a function f with n independent variables t_i $(i = 1 \cdots n)$, r intermediate variables t_{n+i} $(i = 1 \cdots r)$ and m dependent variables t_{n+r+i} $(i = 1 \cdots m)$, the derivatives are given by

$$
\begin{aligned}
\dot{t_k}\,(1 \leq k \leq n) &= e_k \\
\dot{t_k}\,(n+1 \leq k \leq n+r) &= \frac{\partial t_k}{\partial t_j}\ \forall\ j = 1 \cdots n \\
&= \sum_{i \in I_k} \frac{\partial t_k}{\partial t_i}\frac{\partial t_i}{\partial t_j} = \sum_{i \in I_k} \dot{t}_i t_{k,i} \\
\dot{t_k}\,(n+r+1 \leq k \leq n+r+m) &= \frac{\partial t_k}{\partial t_j}\ \forall\ j = 1 \cdots n \\
&= t_{k,i} + \sum_{i \in I_k} \dot{t}_i t_{k,i}
\end{aligned}
$$

$$
\textbf{Here},\ \dot{t_i} = \frac{\partial t_i}{\partial t_j},\ t_{k,i} = \frac{\partial t_k}{\partial t_i}\ \textbf{and}\ I_k = (\forall\ i : n < i \leq k)
$$

$e_k = (0, 0, 0, 1, 0, 0)$ n element vector with k^{th} component as unity

Forward mode AD

In general, for a computer program that represents a given function $F(\mathbf{X}) = \mathbf{Y}$ with $\mathbf{X} \in \mathbb{R}^n$ and $\mathbf{Y} \in \mathbb{R}^m$, the forward mode computes the directional derivative $d\mathbf{Y} = \mathbf{J} \cdot d\mathbf{X}$ for each direction $d\mathbf{X}$

$$d\mathbf{Y} = \dot{\mathbf{Y}} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \cdot \\ \cdot \\ \dot{y}_m \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \cdot \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \cdot \\ \cdot \\ \dot{x}_n \end{pmatrix} = \mathbf{J} \cdot \dot{\mathbf{X}} = \mathbf{J} \cdot d\mathbf{X}$$

Forward mode AD

In general, for a computer program that represents a given function $F(\mathbf{X}) = \mathbf{Y}$ with $\mathbf{X} \in \mathbb{R}^n$ and $\mathbf{Y} \in \mathbb{R}^m$, the forward mode computes the directional derivative $d\mathbf{Y} = \mathbf{J} \cdot d\mathbf{X}$ for each direction $d\mathbf{X}$

$$d\mathbf{Y} = \dot{\mathbf{Y}} = \begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \\ \cdot \\ \cdot \\ \dot{y}_m \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \cdot \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \cdot \\ \cdot \\ \dot{x}_n \end{pmatrix} = \mathbf{J} \cdot \dot{\mathbf{X}} = \mathbf{J} \cdot d\mathbf{X}$$

Consider $m = 1$, the scalar output case

- For a given direction $\dot{x}_1 = 1$ and $\dot{x}_i = 0, \forall i \neq 1$, the forward mode gives the gradient $\frac{\partial y_1}{\partial x_1}$
- To compute the compelete gradient vector, the forward mode has to be called n times

Reverse or adjoint mode AD

- Consider a function $f(x) = y$. The adjoints of x and y are denoted by $\bar{x}$ and $\bar{y}$. They are defined as

$$\bar{x} = \frac{\partial f}{\partial x} = \frac{\partial y}{\partial x} \textbf{ and } \bar{y} = \frac{\partial y}{\partial y} = 1$$

- Another function $f(x) = (y_1, y_2)$. The adjoints are given by

$$\bar{x} = \frac{\partial f}{\partial x} = \frac{\partial}{\partial x}(y_1, y_2) = \left(\frac{\partial y_1}{\partial x}, \frac{\partial y_2}{\partial x}\right) \textbf{ and}$$

$$\bar{y_1} = \frac{\partial f}{\partial y_1} = \left(\frac{\partial y_1}{\partial y_1}, \frac{\partial y_2}{\partial y_1}\right) = (1, 0)$$

$$\bar{y_2} = \frac{\partial f}{\partial y_2} = \left(\frac{\partial y_1}{\partial y_2}, \frac{\partial y_2}{\partial y_2}\right) = (0, 1)$$

Adjoint mode AD ..

- Adjoint mode AD $\Rightarrow$ First forward sweep and then reverse sweep
- Forward sweep: The function value is evaluated
- Reverse sweep: The derivatives are computed in reverse direction
- There may exist variables whose values are over-written during forward sweep and are then used in reverse sweep during the computation of adjoints. One has to store all the values of such variables during the forward sweep

Adjoint mode AD - Example 1

Once again, consider the test function $f(x_1, x_2) = x_1 x_2 + sin(x_1) + e^{x_2}$.

Forward sweep

$$
\begin{aligned}
t_1 &= x_1 \\
t_2 &= x_2 \\
t_3 &= x_1 * x_2 = t_1 * t_2 \\
t_4 &= sin(x_1) = sin(t_1) \\
t_5 &= e^{x_2} = e^{t_2} \\
t_6 &= t_3 + t_4 \\
t_7 &= t_5 + t_6 = f
\end{aligned}
$$

Now define

$$\bar{t_7} = \bar{f} = \frac{\partial t_7}{\partial t_7} = 1$$

$$\bar{t_k}(k < 7) = \frac{\partial t_7}{\partial t_k} = \sum_{i \in I_k} \frac{\partial t_7}{\partial t_i} \frac{\partial t_i}{\partial t_k} = \sum_{i \in I_k} \bar{t}_i t_{i,k} \textbf{ and } I_k = \{\forall\ i : k < i \leq 7\}$$

Adjoint mode AD - Example 1 ..

Reverse sweep

$$\bar{t}_7 = \frac{\partial f}{\partial t_7} = \frac{\partial t_7}{\partial t_7} = 1$$

$$\bar{t}_6 = \frac{\partial t_7}{\partial t_6} = \bar{t}_7 t_{7,6} = 1$$

$$\bar{t}_5 = \frac{\partial t_7}{\partial t_5} = \bar{t}_7 t_{7,5} = 1$$

$$\bar{t}_4 = \frac{\partial t_7}{\partial t_4} = \bar{t}_7 t_{7,4} + \bar{t}_6 t_{6,4} = 1$$

$$\bar{t}_3 = \frac{\partial t_7}{\partial t_3} = \bar{t}_7 t_{7,3} + \bar{t}_6 t_{6,3} = 1$$

$$\bar{t}_2 = \frac{\partial t_7}{\partial t_2} = \bar{t}_7 t_{7,2} + \bar{t}_6 t_{6,2} + \bar{t}_5 t_{5,2} + \bar{t}_3 t_{3,2} = x_1 + e^{x_2}$$

$$\bar{t}_1 = \frac{\partial t_7}{\partial t_1} = \bar{t}_7 t_{7,1} + \bar{t}_6 t_{6,1} + \bar{t}_4 t_{4,1} + \bar{t}_3 t_{3,1} + \bar{t}_2 t_{2,1} = x_2 + cos x_1$$

Adjoint mode AD - Example 1 ..

Reverse sweep

$$\bar{t}_7 = \frac{\partial f}{\partial t_7} = \frac{\partial t_7}{\partial t_7} = 1$$

$$\bar{t}_6 = \frac{\partial t_7}{\partial t_6} = \bar{t}_7 t_{7,6} = 1$$

$$\bar{t}_5 = \frac{\partial t_7}{\partial t_5} = \bar{t}_7 t_{7,5} = 1$$

$$\bar{t}_4 = \frac{\partial t_7}{\partial t_4} = \bar{t}_7 t_{7,4} + \bar{t}_6 t_{6,4} = 1$$

$$\bar{t}_3 = \frac{\partial t_7}{\partial t_3} = \bar{t}_7 t_{7,3} + \bar{t}_6 t_{6,3} = 1$$

$$\bar{t}_2 = \frac{\partial t_7}{\partial t_2} = \bar{t}_7 t_{7,2} + \bar{t}_6 t_{6,2} + \bar{t}_5 t_{5,2} + \bar{t}_3 t_{3,2} = x_1 + e^{x_2}$$

$$\bar{t}_1 = \frac{\partial t_7}{\partial t_1} = \bar{t}_7 t_{7,1} + \bar{t}_6 t_{6,1} + \bar{t}_4 t_{4,1} + \bar{t}_3 t_{3,1} + \bar{t}_2 t_{2,1} = x_2 + cos x_1$$

Adjoint mode AD computes the derivatives $\bar{x}_1$ and $\bar{x}_2$ in one sweep

Adjoint mode AD ..

In general, for a function f with n independent variables t_i $(i = 1 \cdots n)$, r intermediate variables t_{n+i} $(i = 1 \cdots r)$ and m dependent variables t_{n+r+i} $(i = 1 \cdots m)$, the adjoints are given by

$$\begin{aligned}
\bar{t}_{n+r+i} &= e_i \quad (\forall\ i = 1 \cdots m) \\
\bar{t}_k &= \frac{\partial t_j}{\partial t_k} \quad (\forall\ j = n+r+1 \cdots n+r+m) \ \textbf{and} \ (\forall\ k = 1 \cdots n+r) \\
&= \bar{t}_j t_{j,k} + \sum_{i \in I_k} \bar{t}_i t_{i,k} \ \textbf{and} \ I_k = (\forall\ i : max\,(n,k) < i \leq n+r)
\end{aligned}$$

Adjoint mode AD ..

Consider a function $F(\mathbf{X}) = \mathbf{Y}$ with $\mathbf{X} \in \mathbb{R}^n$ and $\mathbf{Y} \in \mathbb{R}^m$. Given the adjoints $\bar{\mathbf{Y}}$ of the dependent vector $\mathbf{Y}$, the reverse mode AD computes the transposed Jacobian vector product $\bar{\mathbf{X}} = \mathbf{J}^T \cdot \bar{\mathbf{Y}}$

$$\bar{\mathbf{X}} = \begin{pmatrix} \bar{x_1} \\ \bar{x_2} \\ . \\ . \\ \bar{x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \cdot \begin{pmatrix} \bar{y_1} \\ \bar{y_2} \\ . \\ . \\ \bar{y_m} \end{pmatrix} = \mathbf{J}^T \cdot \bar{\mathbf{Y}}$$

Adjoint mode AD ..

Consider a function $F(\mathbf{X}) = \mathbf{Y}$ with $\mathbf{X} \in \mathbb{R}^n$ and $\mathbf{Y} \in \mathbb{R}^m$. Given the adjoints $\bar{\mathbf{Y}}$ of the dependent vector $\mathbf{Y}$, the reverse mode AD computes the transposed Jacobian vector product $\bar{\mathbf{X}} = \mathbf{J}^T \cdot \bar{\mathbf{Y}}$

$$\bar{\mathbf{X}} = \begin{pmatrix} \bar{x_1} \\ \bar{x_2} \\ . \\ . \\ \bar{x_n} \end{pmatrix} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_n} & \frac{\partial y_2}{\partial x_n} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix} \cdot \begin{pmatrix} \bar{y_1} \\ \bar{y_2} \\ . \\ . \\ \bar{y_m} \end{pmatrix} = \mathbf{J}^T \cdot \bar{\mathbf{Y}}$$

- For a scalar output y $(m = 1)$, the reverse mode AD computes the gradients with respect to all the independent variables at once

$$\bar{\mathbf{X}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots \frac{\partial y_1}{\partial x_n} \end{pmatrix}^T \bar{y_1}$$

Computational complexity of a program

The computational complexity of a computer program is measured by the number of `flops`. Let us define some notations

- $C_{\pm}$: Computational time for 1 addition or subtraction $=$ 1 `flops`
- C_{*}: Computational time for 1 multiplication $=$ 1 `flops`
- $C_{\div}$: Computational time for 1 division $=$ 1 `flops`
- C_{I}: Computational time for computing 1 intrinsic function or non-linear operation $=$ 1 `nlops` $= r > 1$ `flops`

The computational cost of a program for a given function f is then measured as

$$\texttt{flops}(f) = \text{no. of } C_{\pm} + \text{no. of } C_{*} + \text{no. of } C_{\div} + \text{no. of } C_{I}$$

Computational cost - Elementary operations

Primal	Forward mode AD	Adjoint mode AD
$y = x_1 + x_2$ (1 flops)	$\dot{y} = \dot{x}_1 + \dot{x}_2$ (1 flops)	$\bar{x}_1 = \bar{x}_1 + \bar{y}$ (1 flops) $\bar{x}_2 = \bar{x}_2 + \bar{y}$ (1 flops)
$y = x_1 - x_2$ (1 flops)	$\dot{y} = \dot{x}_1 - \dot{x}_2$ (1 flops)	$\bar{x}_1 = \bar{x}_1 + \bar{y}$ (1 flops) $\bar{x}_2 = \bar{x}_2 - \bar{y}$ (1 flops)
$y = x_1 x_2$ (1 flops)	$\dot{y} = x_1 \dot{x}_2 + \dot{x}_1 x_2$ (3 flops)	$\bar{x}_1 = \bar{x}_1 + x_2 \bar{y}$ (2 flops) $\bar{x}_2 = \bar{x}_2 + x_1 \bar{y}$ (2 flops)
$y = \frac{x_1}{x_2}$ (1 flops)	$\dot{y} = \frac{\dot{x}_1}{x_2} - \frac{x_1 \dot{x}_2}{x_2^2}$ (5 flops)	$\bar{x}_1 = \bar{x}_1 + \frac{\bar{y}}{x_2}$ (2 flops) $\bar{x}_2 = \bar{x}_2 - \frac{x_1 \bar{y}}{x_2^2}$ (4 flops)

Computational cost - Elementary operations ..

Primal	Forward mode AD	Adjoint mode AD
$y = sin(x)$ (r `flops`)	$\dot{y} = cos(x)\dot{x}$ ($r+1$ `flops`)	$\bar{x} = \bar{x} + cos(x)\bar{y}$ ($r+2$ `flops`)
$y = log(x)$ (r `flops`)	$\dot{y} = \frac{\dot{x}}{x}$ (1 `flops`)	$\bar{x} = \bar{x} + \frac{\bar{y}}{x}$ (2 `flops`)
$y = e^x$ (r `flops`)	$\dot{y} = e^x\dot{x}$ ($r+1$ `flops`)	$\bar{x} = \bar{x} + e^x\bar{y}$ ($r+2$ `flops`)

Example 1: Cost of forward mode AD

Input: f

$t_1 = x_1$

$t_2 = x_2$

$t_3 = x_1 * x_2 = t_1 * t_2$ `% 1 flops`

$t_4 = sin(x_1) = sin(t_1)$ `% 1 nlops`

$t_5 = e^{x_2} = e^{t_2}$ `% 1 nlops`

$t_6 = t_3 + t_4$ `% 1 flops`

$t_7 = t_5 + t_6 = f$ `% 1 flops`

Output: $\dot{f}$

$\dot{t}_1 = \dot{x}_1$

$\dot{t}_2 = \dot{x}_2$

$\dot{t}_3 = t_1 * \dot{t}_2 + \dot{t}_1 * t_2$ `% 3 flops`

$\dot{t}_4 = cos(t_1) * \dot{t}_1$ `% 1 nlops + 1 flops`

$\dot{t}_5 = e^{t_2} * \dot{t}_2$ `% 1 nlops + 1 flops`

$\dot{t}_6 = \dot{t}_3 + \dot{t}_4$ `% 1 flops`

$\dot{t}_7 = \dot{t}_5 + \dot{t}_6$ `% 1 flops`

- Cost of f : 3 `flops` + 2 `nlops`
- Cost of $\dot{f}$: 7 `flops` + 2 `nlops`

Computational cost of forward mode AD

- 1 addition/subtraction in f ($1\ \texttt{flops}$) $\Rightarrow$ 1 add/sub in $\dot{f}$ ($1\ \texttt{flops}$)
- 1 multiplication in f ($1\ \texttt{flops}$) $\Rightarrow$ 2 mults + 1 add in $\dot{f}$ ($3\ \texttt{flops}$)
- 1 division in f ($1\ \texttt{flops}$) $\Rightarrow$ 2 divs + 2 mults + 1 sub in $\dot{f}$ ($5\ \texttt{flops}$)
- 1 intrinsic operation in f ($r\ \texttt{flops}$) $\Rightarrow$ 1 intrinsic operation + 1 mult in $\dot{f}$ ($r\ \texttt{flops} + 1\ \texttt{flops}$)

$$
\begin{aligned}
\texttt{flops}(\dot{f}) &=\leq Max\,(1, 3, 5, r+1) \\
\frac{\texttt{flops}(\dot{f})}{\texttt{flops}(f)} &=\leq Max\left(1, 3, 5, 1 + \frac{1}{r}\right) \\
&\leq 5
\end{aligned}
$$

Computational cost of adjoint mode AD

- 1 addition in primal (1 flops) $\Rightarrow$ 2 adds in adjoint (2 flops)
- 1 subtraction in primal (1 flops) $\Rightarrow$ 1 add + 1 sub in adjoint (2 flops)
- 1 multiplication in primal (1 flops) $\Rightarrow$ 2 mults + 2 adds in adjoint (4 flops)
- 1 division in primal (1 flops) $\Rightarrow$ 2 divs + 2 mults + 1 add + 1 sub in adjoint (6 flops)
- 1 intrinsic operation in primal (r flops) $\Rightarrow$ 1 intrinsic operation + 1 mult + 1 add in adjoint (r flops + 2 flops)

$$\begin{aligned}
\mathtt{flops}(adjoint) &=\leq Max\,(2,4,6,r+2) \\
\frac{\mathtt{flops}(adjoint)}{\mathtt{flops}(primal)} &=\leq Max\left(2,4,6,1+\frac{2}{r}\right) \\
&\leq 6
\end{aligned}$$

Implementation of AD

AD can be implemented in two ways:

- Operator overloading
  - Augments the original source code to compute the derivatives.
  - The basic idea of this approach is to overload operators and standard functions so that they compute the derivatives along with the primal values.
- Source transformation
  - Generates a new computer program that evaluates the derivatives.
  - Reads the source code and identifies the active variables and subroutines.
  - Then generates tangent or adjoint variables and subroutines.

List of AD tools

- Operator overloading
  - ADOL-C (C/C++)
  - TOMLAB/TomSym (MATLAB)
  - CppAD (C/C++)
- Source transformation
  - Tapenade (C/C++, Fortran77, Fortran95)
  - ADIFOR (FORTRAN, only forward mode)
  - ADIC (C/C++, only forward mode)
  - ADiMat (MATLAB, only forward mode)
  - TOMLAB/TomSym (MATLAB)
  - OpenAD (Language independent)
  - TAF (Fortran77, Fortran95)

For complete list of AD tools: http://www.autodiff.org

AD by use of Tapenade

- Based on source transformation. Available in FORTRAN/C/C++
- AD engine developed by TROPICS team at INRIA, Sophia Antipolis
- AD can be performed online at the Tapenade webserver : `http://tapenade.inria.fr:8080/tapenade/index.jsp`
- Can also run via a command line by installing locally
- For any variable `x` that is differentiable, Tapenade stores the derivatives as `xd` in forward mode and `xb` in reverse mode
- For any subroutine `function`, the derivative names in forward and reverse modes are `function_d` and `function_b` respectively
- User manual and references on Tapenade can be found on `http://www-sop.inria.fr/tropics/`

Tapenade: Commands and notations

- In Tapenade, the forward and revserse modes of AD can be performed by executing the following commands
  - tapenade -d -head function -vars " " -outvars " " filename
  - tapenade -b -head function -vars " " -outvars " " filename
- The flags `d` and `b` specify the forward and reverse modes of AD
- The flag `head` specifies the subroutine `function` that has to be differentiated while `filename` represents the name of the file that contains the subroutine `function`
- `vars` and `outvars` specify the independent and dependent variables respectively. One has to specify them within double quotes separated by blank space
- For details on flags and command options, refer to the user manual

Example 1: Primal code

```
      program example1
C
      double precision x1, x2, f
C
      read(*,*) x1, x2
      call eval_func(x1, x2, f)
      write(*,*) f
      end
C
      subroutine eval_func(x1,x2,f)
C
      double precision x1, x2, f
      f = x1*x2 + dsin(x1) + dexp(x2)
      end
```

Example 1: Forward mode AD code

```
C  Generated by TAPENADE     (INRIA, Tropics team)
C  Tapenade 3.5 (r3683) - 12 Feb 2011 10:38
C  Differentiation of eval_func in forward (tangent) mode:
C   variations   of useful results: f
C   with respect to varying inputs: x1 x2
C   RW status of diff variables: f:out x1:in x2:in
C
      SUBROUTINE EVAL_FUNC_D(x1, x1d, x2, x2d, f, fd)
      IMPLICIT NONE
      DOUBLE PRECISION x1, x2, f
      DOUBLE PRECISION x1d, x2d, fd
      INTRINSIC DEXP
      INTRINSIC DSIN
C
      fd = x1d*x2 + x1*x2d + x1d*DCOS(x1) + x2d*DEXP(x2)
      f = x1*x2 + DSIN(x1) + DEXP(x2)
      END
```

Example 1: Adjoint mode AD code

```
C Generated by TAPENADE     (INRIA, Tropics team)
C Tapenade 3.5 (r3683) - 12 Feb 2011 10:38
C Differentiation of eval_func in reverse (adjoint) mode:
C gradient of useful results: f
C with respect to varying inputs: f x1 x2
C RW status of diff variables: f:in-zero x1:out x2:out
C
      SUBROUTINE EVAL_FUNC_B(x1, x1b, x2, x2b, f, fb)
      IMPLICIT NONE
      DOUBLE PRECISION x1, x2, f
      DOUBLE PRECISION x1b, x2b, fb
C
      INTRINSIC DEXP
      INTRINSIC DSIN
      x1b = (DCOS(x1)+x2)*fb
      x2b = (DEXP(x2)+x1)*fb
      fb = 0.D0
      END
```

Example 1: Adjoint mode AD code

```
C  Generated by TAPENADE     (INRIA, Tropics team)
C  Tapenade 3.5 (r3906) - 16 May 2011 09:50
C  Differentiation of eval_func in reverse (adjoint) mode:
C   gradient     of useful results: f x1 x2
C   with respect to varying inputs: f x1 x2
C   RW status of diff variables: f:in-zero x1:incr x2:incr
C
      SUBROUTINE EVAL_FUNC_B(x1, x1b, x2, x2b, f, fb)
      IMPLICIT NONE
      DOUBLE PRECISION x1, x2, f
      DOUBLE PRECISION x1b, x2b, fb
C
      INTRINSIC DEXP
      INTRINSIC DSIN
      x1b = x1b + (DCOS(x1)+x2)*fb
      x2b = x2b + (DEXP(x2)+x1)*fb
      fb = 0.D0
      END
```

Example 2: Primal code

This example illustrates on how Tapenade builds the control structures

```
      subroutine eval_func(x1,x2,f)
      double precision x1(10), x2(10), f
C
      do i = 1, 10
      if(x1(i) .lt. 0.0) then
      f = x1(i) + x2(i)
      else f = x1(i) - x2(i)
      endif
      enddo
      end
```

Example 2: Forward mode AD code

```
      SUBROUTINE EVAL_FUNC_D(x1, x1d, x2, x2d, f, fd)
      IMPLICIT NONE
C
      DOUBLE PRECISION x1(10), x2(10), f
      DOUBLE PRECISION x1d(10), x2d(10), fd
      INTEGER i
      fd = 0.D0
C
      DO i=1,10
        IF (x1(i) .LT. 0.0) THEN
          fd = x1d(i) + x2d(i)
          f = x1(i) + x2(i)
        ELSE
          fd = x1d(i) - x2d(i)
          f = x1(i) - x2(i)
        END IF
      ENDDO
      END
```

Example 2: Adjoint mode AD code

Forward sweep

```
      SUBROUTINE EVAL_FUNC_B(x1, x1b, x2, x2b, f, fb)
      IMPLICIT NONE
      DOUBLE PRECISION x1(10), x2(10), f
      DOUBLE PRECISION x1b(10), x2b(10), fb
      INTEGER i
      INTEGER branch
      INTEGER ii1
C
      DO i=1,10
        IF (x1(i) .LT. 0.0) THEN
          CALL PUSHCONTROL1B(1)
        ELSE
          CALL PUSHCONTROL1B(0)
        END IF
      ENDDO
```

Continued on next page ..

Example 2: Adjoint mode AD code ..

Reverse sweep

```
DO ii1=1,10
  x2b(ii1) = 0.D0
ENDDO
DO i=10,1,-1
  CALL POPCONTROL1B(branch)
  IF (branch .EQ. 0) THEN
    x1b(i) = x1b(i) + fb
    x2b(i) = x2b(i) - fb
  ELSE
    x1b(i) = x1b(i) + fb
    x2b(i) = x2b(i) + fb
  END IF
  fb = 0.D0
ENDDO
END
```

Example 3: Primal code

This example illustrates on how Tapenade stores and retrieves the intermediate variables using the PUSH and POP operations in the adjoint code

```
      subroutine eval_func(x1, x2, f)
C
       double precision x1, x2, temp1, f
C
       temp1 = cos(x1) + sin(x2)
       x1 = x1*x2
       x2 = x1/x2
       temp1 = temp1 + (x1*x2)
       f = temp1
C
       end
```

Example 3: Forward mode AD code

```
      SUBROUTINE EVAL_FUNC_D(x1, x1d, x2, x2d, f, fd)
      IMPLICIT NONE
C
      DOUBLE PRECISION x1, x2, f
      DOUBLE PRECISION x1d, x2d, fd
      INTRINSIC DEXP
      INTRINSIC DSIN
C
      fd = x1d*x2 + x1*x2d + x1d*DCOS(x1) + x2d*DEXP(x2)
      f = x1*x2 + DSIN(x1) + DEXP(x2)
      END
```

Example 3: Adjoint mode AD code

Forward sweep

```
      SUBROUTINE EVAL_FUNC_B(x1, x1b, x2, x2b, f, fb)
      IMPLICIT NONE
      DOUBLE PRECISION x1, x2, temp1, f
      DOUBLE PRECISION x1b, x2b, temp1b, fb
      INTRINSIC COS
      INTRINSIC SIN
C
      CALL PUSHREAL8(x1)
      x1 = x1*x2
      CALL PUSHREAL8(x2)
      x2 = x1/x2
```

Continued on next page ..

Example 3: Adjoint mode AD code ..

Reverse sweep

```
temp1b = fb
x1b = x2*temp1b
x2b = x1*temp1b
CALL POPREAL8(x2)
x1b = x1b + x2b/x2
x2b = -(x1*x2b/x2**2)
CALL POPREAL8(x1)
x2b = x2b + COS(x2)*temp1b + x1*x1b
x1b = x2*x1b - SIN(x1)*temp1b
fb = 0.D0
END
```

Example 4: Primal code

Another example on PUSH and POP operations

```
      subroutine eval_func(x1,x2,f)
C
      double precision x1(10), x2(10), x3, x4, f
C
      do i = 1, 10
      x3 = x1(i) + x2(i)
      x1(i) = x1(i) + x4
      x4 = x1(i)*x3
      enddo
C
      f = x3 + x4
C
      end
```

Example 4: Adjoint mode AD code

```
      SUBROUTINE EVAL_FUNC_B(x1, x1b, x2, x2b, f, fb)
        IMPLICIT NONE
C
        DOUBLE PRECISION x1(10), x2(10), x3, x4, f
        DOUBLE PRECISION x1b(10), x2b(10), x3b, x4b, fb
        INTEGER i
        INTEGER ii1
C
        DO i=1,10
          CALL PUSHREAL8(x3)
          x3 = x1(i) + x2(i)
          x1(i) = x1(i) + x4
          x4 = x1(i)*x3
        ENDDO
        x3b = fb
        x4b = fb
```

Continued on next page ..

Example 4: Adjoint mode AD code ..

```
DO ii1=1,10
  x1b(ii1) = 0.D0
ENDDO
DO ii1=1,10
  x2b(ii1) = 0.D0
ENDDO
DO i=10,1,-1
  x1b(i) = x1b(i) + x3*x4b
  x3b = x3b + x1(i)*x4b
  x4b = x1b(i)
  CALL POPREAL8(x3)
  x1b(i) = x1b(i) + x3b
  x2b(i) = x2b(i) + x3b
  x3b = 0.D0
ENDDO
fb = 0.D0
END
```

On the memory/CPU requirements of adjoint code

- If the original program overwrites any intermediate variable x_k, then it must be stored in the forward sweep before it can be used in the reverse sweep for computing the adjoints
- These variables are stored/retrieved from a memory stack by using the `PUSH`/`POP` operations
- For programs with a large number of intermediate operations the memory requirements for taping (storing) can become significantly large or even prohibitively expensive
- At times the computational cost of storing/retrieving the data becomes quite significant