Adjoint Derivative Computation

Moritz Diehl and Carlo Savorgnan
There are several methods for calculating derivatives:

1. By hand
2. Symbolic differentiation
3. Numerical differentiation
4. “Imaginary trick” in MATLAB
5. Automatic differentiation
   - Forward mode
   - Adjoint (or backward or reverse) mode
Calculating derivatives by hand

Time consuming & error prone
Symbolic differentiation

We can obtain an expression of the derivatives we need with: Mathematica, Maple, ...
Symbolic differentiation

We can obtain an expression of the derivatives we need with: Mathematica, Maple, ...

Often this results in a very long code which is expensive to evaluate.
Consider a function $f : \mathbb{R}^n \to \mathbb{R}$

$$\nabla f(x)^T p \approx \frac{f(x + tp) - f(x)}{t}$$

Really easy to implement.
Consider a function $f : \mathbb{R}^n \to \mathbb{R}$

$$\nabla f(x)^T p \approx \frac{f(x + tp) - f(x)}{t}$$

Really easy to implement.

**Problem**

How should we choose $t$?
Problem
How should we chose $t$?

A rule of thumb
Set $t = \sqrt{\epsilon}$, where $\epsilon$ is set to machine precision or the precision of $f$.

The accuracy of the derivative is approximately $\sqrt{\epsilon}$. 
Consider an analytic function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Set $t = 10^{-100}$.

\[ \nabla f(x)^T p = \frac{\Im(f(x + itp))}{t} \]

$\nabla f(x)^T p$ can be calculated up to machine precision!
Consider a function $f : \mathbb{R}^n \to \mathbb{R}$ defined by using $m$ elementary operations $\phi_i$.

### Function evaluation

**Input:** $x_1, x_2, \ldots, x_n$

**Output:** $x_{n+m}$

for $i = n + 1$ to $n + m$

\[
x_i \leftarrow \phi_i(x_1, \ldots, x_{i-1})
\]

end for
Consider a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ defined by using $m$ elementary operations $\phi_i$.

### Function evaluation

**Input:** $x_1, x_2, \ldots, x_n$

**Output:** $x_{n+m}$

for $i = n + 1$ to $n + m$

\[
x_i \leftarrow \phi_i(x_1, \ldots, x_{i-1})
\]

end for

### Example

\[
f(x_1, x_2, x_3) = \sin(x_1 x_2) + \exp(x_1 x_2 x_3)
\]

Evaluation code (for $m = 5$ elementary operations):

\[
x_4 \leftarrow x_1 x_2; \quad x_5 \leftarrow \sin(x_4); \quad x_6 \leftarrow x_4 x_3;
\]

\[
x_7 \leftarrow \exp(x_6) \quad x_8 \leftarrow x_5 + x_7;
\]
Automatic differentiation: forward mode

Assume $x(t)$ and $f(x(t))$.

\[ \dot{x} = \frac{dx}{dt} \quad \dot{f} = \frac{df}{dt} = J_f(x) \dot{x} \]

For $i = 1, \ldots, m$

\[ \frac{dx_{n+i}}{dt} = \sum_{j=1}^{n+i-1} \frac{\partial \phi_{n+i}}{\partial x_j} \frac{dx_j}{dt} \]
Automatic differentiation: forward mode

Assume \( x(t) \) and \( f(x(t)) \).

\[
\dot{x} = \frac{dx}{dt} \quad \quad \dot{f} = \frac{df}{dt} = J_f(x)\dot{x}
\]

For \( i = 1, \ldots, m \)

\[
\frac{dx_{n+i}}{dt} = \sum_{j=1}^{n+i-1} \frac{\partial \phi_{n+i}}{\partial x_j} \frac{dx_j}{dt}
\]

Forward automatic differentiation

**Input**: \( \dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n \) and (and all partial derivatives \( \frac{\partial \phi_{n+i}}{\partial x_j} \))

**Output**: \( \dot{x}_{n+m} \)

for \( i = 1 \) to \( m \)

\[
\dot{x}_{n+i} \leftarrow \sum_{j=1}^{n+i-1} \frac{\partial \phi_{n+i}}{\partial x_j} \dot{x}_j
\]

end for
Automatic differentiation: reverse mode

Reverse automatic differentiation

**Input:** all $\frac{\partial \phi_i}{\partial x_j}$

**Output:** $\bar{x}_1, \ldots, \bar{x}_n$

$\bar{x}_1, \ldots, \bar{x}_n \leftarrow 0$

$\bar{x}_{n+m} \leftarrow 1$

for $j = n + m$ down to $n + 1$

for all $i = 1, 2, \ldots, j - 1$

\[ \bar{x}_i \leftarrow \bar{x}_i + \bar{x}_j \frac{\partial \phi_j}{\partial x_i} \]

end for

end for
Automatic differentiation summary so far

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]

Cost of forward mode per directional derivative

\[ \text{cost}(\nabla f^T p) \leq 2 \text{cost}(f) \]

For full gradient \( \nabla f \), need \( 2n \text{cost}(f) \)!
Automatic differentiation summary so far

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]

**Cost of forward mode per directional derivative**

\[
\text{cost}(\nabla f^T p) \leq 2 \text{cost}(f)
\]

For full gradient \( \nabla f \), need \( 2n \text{cost}(f) \)!

**Cost of reverse mode: full gradient**

\[
\text{cost}(\nabla f) \leq 3 \text{cost}(f)
\]

Independent of \( n \)!
Automatic differentiation summary so far

\[ f : \mathbb{R}^n \rightarrow \mathbb{R} \]

Cost of forward mode per directional derivative

\[ \text{cost}(\nabla f^T p) \leq 2 \text{cost}(f) \]

For full gradient \( \nabla f \), need \( 2n \text{cost}(f) \)!

Cost of reverse mode: full gradient

\[ \text{cost}(\nabla f) \leq 3 \text{cost}(f) \]

Independent of \( n \)! Only drawback: large memory needed for all intermediate values
Automatic differentiation can be used for any $f : \mathbb{R}^n \to \mathbb{R}^m$.

**Cost of forward mode for forward direction $p \in \mathbb{R}^n$**

$$\text{cost}(J_f p) \leq 2 \text{cost}(f)$$
Automatic differentiation can be used for any $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

**Cost of forward mode for forward direction $p \in \mathbb{R}^n$**

$$\text{cost}(J_f p) \leq 2 \text{cost}(f)$$

**Cost of reverse mode per reverse direction $p \in \mathbb{R}^m$**

$$\text{cost}(p^T J_f) \leq 3 \text{cost}(f)$$
Automatic differentiation can be used for any $f : \mathbb{R}^n \to \mathbb{R}^m$.

Cost of forward mode for forward direction $p \in \mathbb{R}^n$

$$\text{cost}(J_f p) \leq 2 \text{cost}(f)$$

Cost of reverse mode per reverse direction $p \in \mathbb{R}^m$

$$\text{cost}(p^T J_f) \leq 3 \text{cost}(f)$$

For computation of full Jacobian $J_f$, choice of best mode depends on size of $n$ and $m$. 
Regard function code as the computation of a vector which is “growing” at every iteration

\[ \tilde{x}_1 = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n+1} \end{bmatrix} = \Phi_1 \left( \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \right) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ \phi_{n+1}(x_1, x_2, x_3, \ldots, x_n) \end{bmatrix} \]

\[ \ldots \]

\[ \tilde{x}_m = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n+m} \end{bmatrix} = \Phi_m \left( \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n+m-1} \end{bmatrix} \right) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n+m-1} \\ \phi_{n+m}(x_1, x_2, x_3, \ldots, x_{n+m-1}) \end{bmatrix} \]
Evaluation of $f : \mathbb{R}^n \rightarrow \mathbb{R}^q$ can then be written as

$$f(x) = Q \Phi_m(\Phi_{m-1}(\ldots \Phi_2(\Phi_1(x)) \ldots))$$

with $Q \in \mathbb{R}^{q \times (n+m)}$ a 0-1 matrix selecting the output variables, e.g. for $q = 1$

$$Q = [0 \ 0 \ldots \ 0 \ 1]$$

Then the full Jacobian is given by

$$J_f(x) = Q J_{\Phi_m}(\tilde{x}_m) J_{\Phi_{m-1}}(\tilde{x}_{m-1}) \ldots J_{\Phi_1}(x)$$

where the Jacobians of $\Phi_i$ are

$$J_{\Phi_i} = \begin{bmatrix}
1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & 1 \\
\frac{\partial \phi_{n+i}}{\partial x_1} & \frac{\partial \phi_{n+i}}{\partial x_2} & \frac{\partial \phi_{n+i}}{\partial x_3} & \ldots & \frac{\partial \phi_{n+i}}{\partial x_{n+i-1}}
\end{bmatrix}$$
Forward mode:

\[ J_f p = Q J_{\Phi_m} J_{\Phi_{m-1}} \cdots J_{\Phi_1} p \]
\[ = Q ( J_{\Phi_m} ( J_{\Phi_{m-1}} \cdots ( J_{\Phi_1} p ))) \]

Adjoint mode:

\[ p^T J_f = p^T Q J_{\Phi_m} J_{\Phi_{m-1}} \cdots J_{\Phi_1} \]
\[ = (((p^T Q) J_{\Phi_m}) J_{\Phi_{m-1}}) \cdots J_{\Phi_1} \]

The adjoint mode corresponds just to the efficient evaluation of the vector matrix product \( p^T J_f \)!
**Software for Adjoint Derivatives**

<table>
<thead>
<tr>
<th>Generic Tools to Differentiate Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ADOL-C for C/C++, using operator overloading (open source)</td>
</tr>
<tr>
<td>- ADIC / ADIFOR for C/FORTRAN, using source code transformation (open source)</td>
</tr>
<tr>
<td>- TAPENADE, CppAD (open source), ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Differential Algebraic Equation Solvers with Adjoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>- SUNDIALS Suite CVODES / IDAS (Sandia, open source)</td>
</tr>
<tr>
<td>- DAESOL-II (Uni Heidelberg)</td>
</tr>
<tr>
<td>- ACADO Integrators (Leuven, open source)</td>
</tr>
</tbody>
</table>