Numerical Optimization

Lecture Notes #17

Calculating Derivatives — Automatic Differentiation

Peter Blomgren,
⟨blomgren.peter@gmail.com⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://terminus.sdsu.edu/

Fall 2018

San Diego State University

Outline

San Diego State University

Derivatives Needed!!! — Continued

So far, we looked at using **finite difference** methods for computing good numerical estimates for "missing" derivatives.

The finite difference approach comes at the price of —

(i) introducing some numerical error in the derivative expressions

(ii) a need for extra evaluations of the function (objective) and/or the gradient.

Finite difference approximations work very well, but if the analytical expressions for the gradient and Hessian can be provided it is the way to go!

Next, we look at **Automatic Differentiation Techniques.**

Automatic Differentiation

"Automatic differentiation (AD) is a technique for augmenting computer programs with derivative computations. It exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations such as additions or elementary functions such as exp(). By applying the chain rule of derivative calculus repeatedly to these operations, derivatives of arbitrary order can be computed automatically, and accurate to working precision."

`http://wiki.mcs.anl.gov/autodiff/` provides links and references to usable tools [`ADIC`, `ADIFOR`, `OpenAD/F`, `Rapsodia`] for AD for C/C++, Fortran 77, and Fortran 90. Page last updated: 4 March 2014

We will here take a very brief look at AD. Even more references and pointers can be found at `http://www.AutoDiff.org/`, including tools for MATLAB and python.

Micro-History of Automatic Differentiation

- First publications: early 1950s
- Most cited references
  - (1236) *"AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models"*; Fournier, Skaug, Ancheta, Ianelli, Magnusson, Maunder, Nielsen, and Sibert; Optimization Methods and Software, 27(2), pp. 233-249. 2012.
  - (992) *"Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++"*; Griewank, Juedes, and Utke. ACM Transactions on Mathematical Software (TOMS), 22(2), pp.131-167. 1996.
  - (990) *"Automatic Differentiation: Techniques and Applications"* Louis B. Rall. 1981.
- 1st International Conference on AD: USA, 1991
- IMA Special Workshop, Minneapolis, USA, 1997
- First European AD Workshop: France, 2005
- AD2016 - 7th International Conference on Algorithmic Differentiation September 2016, UK
- 21st EuroAD Workshop 19–20 November 2018, Germany
- 8th SIAM Workshop on Combinatorial Scientific Computing June 6-8, 2018, Bergen, Norway

San Diego State University

The Chain Rule — Forgotten(?) Calculus

The **Chain Rule** in its full vector glory takes the form

The Chain Rule

If $h : \mathbb{R}^m \to \mathbb{R}$, and $\bar{\mathbf{y}} : \mathbb{R}^n \to \mathbb{R}^m$, then for $\bar{\mathbf{x}} \in \mathbb{R}^n$ we can write

$$\nabla_{\bar{\mathbf{x}}} h(\bar{\mathbf{y}}(\bar{\mathbf{x}})) = \sum_{i=1}^{m} \frac{\partial h}{\partial y_i} \nabla y_i(\bar{\mathbf{x}}).$$

Automatic differentiation is essentially applying the chain rule **at the code level**. — There are two **modes** of AD, the *forward* and the *reverse* mode. We follow the example from Nocedal-Wright.

Automatic Differentiation — Example

Example Problem

We consider a function $f : \mathbb{R}^3 \to \mathbb{R}$

$$f(\bar{\mathbf{x}}) = (x_1 x_2 \sin(x_3) + e^{x_1 x_2})/x_3$$

This function can be evaluated in several different ways.

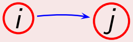
The computational environment will apply certain restictions (aiming for efficiency), *e.g.* *"the multiplication of $x_1 x_2$ must take place before the exponentiation $e^{x_1 x_2}$."* We illustrate one possibility with a (partially ordered) **computational graph** and several **intermediate variables.**

Automatic Differentiation — Example

Language

Some **graph theoretical** language:

- `node#i` is the **parent** of `node#j` (and hence `node#j` the **child** of `node#i`) if there is a directed arc from i to j.



- A node can be evaluated when all its parents are known, so the computation flows left-to-right (see next slide). This is known as a **forward sweep**.

Automatic Differentiation — Example

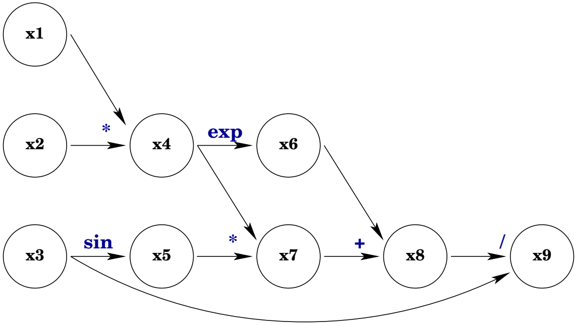


Figure: The computational graph associates with the function

$$f(\bar{\mathbf{x}}) = (x_1 x_2 \sin(x_3) + e^{x_1 x_2})/x_3$$

We have introduced the intermediate variables $x_4 = x_1 * x_2$, $x_5 = \sin(x_3)$, $x_6 = e^{x_4}$, $x_7 = x_4 * x_5$, $x_8 = x_6 + x_7$, and $x_9 = x_8 / x_3$.

Note: In AD, the **software** not the user identifies (either explicitly or implicitly) the intermediate steps. [But someone has to write the software...]

AD — Example / Forward Mode

In forward mode, a **directional derivative**, in the direction $\bar{\mathbf{p}} \in \mathbb{R}^n$, of each intermediate value x_i is evaluated and carried forward simultaneously with the evaluation of x_i itself.

Notation

The directional derivative for $\bar{\mathbf{p}} \in \mathbb{R}^n$ associated with x_i

$$D_{\bar{\mathbf{p}}} x_i \stackrel{\text{def}}{=} [\nabla x_i]^T \bar{\mathbf{p}} = \sum_{j=1}^{n} \frac{\partial x_i}{\partial x_j} p_j, \quad \forall i$$

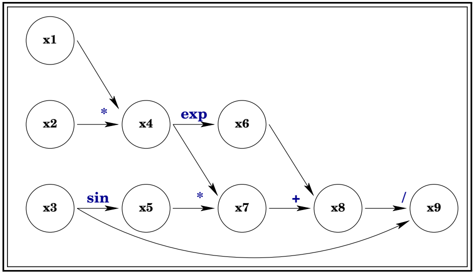
In our example $n = 3$, $i = 1, 2, \ldots, 9$, and our goal is to evaluate $D_{\bar{\mathbf{p}}} x_9 \equiv \nabla f(\bar{\mathbf{x}})^T \bar{\mathbf{p}}$.

Note: $D_{\bar{\mathbf{p}}} x_1 = p_1$, $D_{\bar{\mathbf{p}}} x_2 = p_2$, and $D_{\bar{\mathbf{p}}} x_3 = p_3$. $\bar{\mathbf{p}}$ is known as a **seed vector**.

SAN DIEGO STATE UNIVERSITY

AD — Example / Forward Mode

As soon as the value of x_i is known at a node, we can find $D_{\bar{\mathbf{p}}} x_i$ using the chain rule. *E.g.* when $x_7 = x_4 * x_5$ is known, we have



$$\nabla x_7 = \frac{\partial x_7}{\partial x_4} \nabla x_4 + \frac{\partial x_7}{\partial x_5} \nabla x_5 = x_5 \nabla x_4 + x_4 \nabla x_5$$

Hence, the directional derivative $\mathbf{D_{\bar{p}} x_7 = x_5 D_{\bar{p}} x_4 + x_4 D_{\bar{p}} x_5}$ can be evaluated.

The accumulation of the directional derivative follows the forward sweep, and at the end we have $D_{\bar{\mathbf{p}}} x_9 = D_{\bar{\mathbf{p}}} f = \nabla f(\bar{\mathbf{x}})^T \bar{\mathbf{p}}$.

AD — Example / Forward Mode

The key to **practical implementation** of forward-mode automatic differentiation is the concurrent evaluation of x_i and $D_{\bar{\mathbf{p}}} x_i$.

To obtain the complete gradient vector, the procedure is carried out simultaneously for n seed vectors $\bar{\mathbf{p}} = \{\, \bar{\mathbf{e}}_1, \bar{\mathbf{e}}_2, \ldots, \bar{\mathbf{e}}_n \,\}$.

The computational cost can be significant, for instance a single division operation w/y induces approximately $2n$ multiplications and n additions in the gradient calculation.

AD — Example / Forward Mode

An exact bound on the increase in computation (especially for a complicated expression) is hard to obtain [Worst-Case Bounds are not very useful here], since we have to take into account the cost of storing and retrieving the extra quantities $D_{\bar{\mathbf{e}}_j} x_i$.

AD/FM can be implemented in terms of a **pre-compiler** which transforms function evaluation code into augmented code that evaluates derivatives as well. Alternatively **operator-overloading** (in *e.g.* C++) can be used to transparently extend data structures and operations to perform the necessary bookkeeping and computations.

AD — Example / Reverse Mode

In reverse mode AD, the function value f is first computed in a **forward sweep**, then in a second **reverse sweep** the derivatives of f with respect to each variable x_i (independent and intermediate) are recovered.

We associate an **adjoint variable** $\bar{\mathbf{x}}_i$ with each node in the computational graph. In these adjoint variables we accumulate the partial derivative information $\partial f / \partial x_i$ during the reverse sweep. They are initialized $\bar{\mathbf{x}}_1 = \bar{\mathbf{x}}_2 = \cdots = \bar{\mathbf{x}}_{n-1} = 0$, and $\bar{\mathbf{x}}_n = 1$.

The reverse sweep is built on the chain rule:

$$\bar{\mathbf{x}}_i = \frac{\partial f}{\partial x_i} = \sum_{j \text{ child of } i} \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i}$$

San Diego State University

AD — Example / Reverse Mode

When a term in the sum becomes known, we accumulate it in $\overline{\mathbf{x}}_i$

$$\overline{\mathbf{x}}_i \mathrel{+}= \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial x_i}.$$

Once $\overline{\mathbf{x}}_i$ has received contributions from all its children, it is finalized and can communicate its value to its parent(s).

During the reverse sweep we work directly with **numerical values**, not with formulas or computer code describing the variables x_i and/or the partial derivatives $\partial f / \partial x_i$.

Let's work through the example...

AD — Example / Reverse Mode

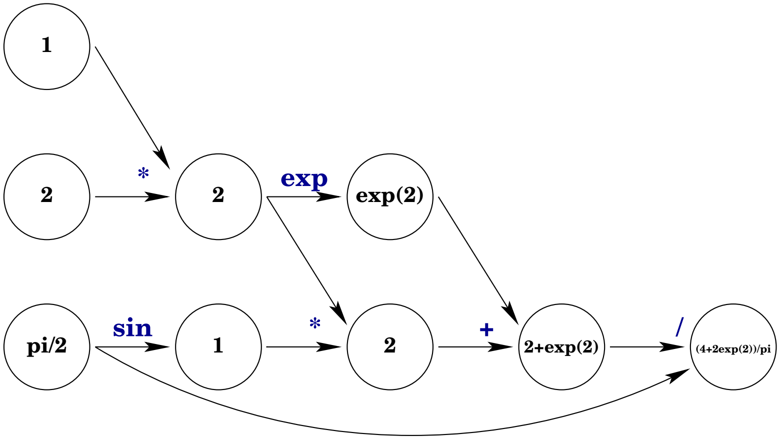


Figure: Forward sweep, the computational graph at level 4; — Sweep complete.

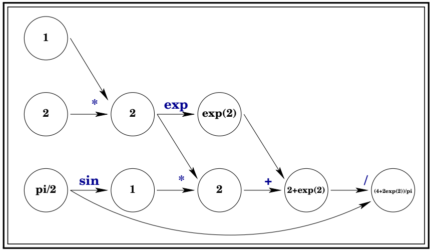
After the forward sweep is complete we initialize the adjoint variables, in particular $\bar{\mathbf{x}}_9 = 1$, and node#9 is finalized (it has no children).

AD — Example / Reverse Mode

The reverse sweep

$$\bar{x}_1 = 0,\ \bar{x}_2 = 0,\ \bar{x}_3 = 0,\ \bar{x}_4 = 0,\ \bar{x}_5 = 0,\ \bar{x}_6 = 0,\ \bar{x}_7 = 0,\ \bar{x}_8 = 0,\ \bar{x}_9 = \mathbf{1}$$

Node#9 is the child of nodes #3 and #8, we update the associated adjoint variables, finalizing node#8. ($x_9 = x_8/x_3$)



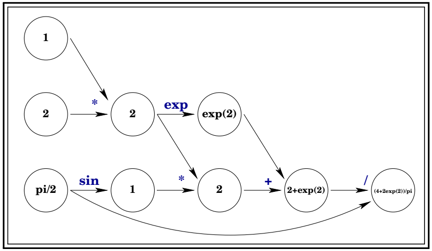
$$\bar{\mathbf{x}}_3 \ \mathbf{+=}\ \frac{\partial f}{\partial x_9}\frac{\partial x_9}{\partial x_3} = -\frac{2+e^2}{(\pi/2)^2} = \frac{-8-4e^2}{\pi^2}, \quad \bar{\mathbf{x}}_8 \ \mathbf{+=}\ \frac{\partial f}{\partial x_9}\frac{\partial x_9}{\partial x_8} = \frac{1}{\pi/2} = \frac{2}{\pi}$$

$$\left\{ \begin{array}{c} \bar{x}_1 = 0,\ \bar{x}_2 = 0,\ \bar{x}_3 = \frac{-8-4e^2}{\pi^2},\ \bar{x}_4 = 0,\ \bar{x}_5 = 0,\ \bar{x}_6 = 0,\ \bar{x}_7 = 0 \\ \bar{\mathbf{x}}_8 = \frac{\mathbf{2}}{\pi},\ \bar{\mathbf{x}}_9 = \mathbf{1} \end{array} \right\}$$

AD — Example / Reverse Mode

$$\left\{ \begin{array}{c} \bar{x}_1 = 0,\ \bar{x}_2 = 0,\ \bar{x}_3 = \frac{-8-4e^2}{\pi^2},\ \bar{x}_4 = 0,\ \bar{x}_5 = 0,\ \bar{x}_6 = 0,\ \bar{x}_7 = 0 \\ \bar{x}_8 = \frac{2}{\pi},\ \bar{x}_9 = 1 \end{array} \right\}$$

We now update the parents of #8 — #6 and #7, finalizing both. ($x_8 = x_6 + x_7$).

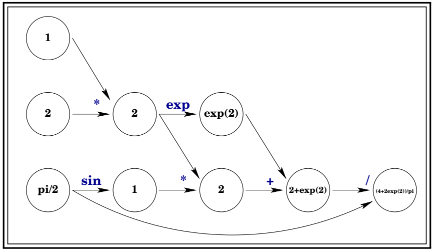


$$\bar{x}_6 \mathrel{+}= \frac{\partial f}{\partial x_8}\frac{\partial x_8}{\partial x_6} = \bar{x}_8 \cdot 1 = \frac{2}{\pi}, \quad \bar{x}_7 \mathrel{+}= \frac{\partial f}{\partial x_8}\frac{\partial x_8}{\partial x_7} = \bar{x}_8 \cdot 1 = \frac{2}{\pi}$$

$$\left\{ \begin{array}{c} \bar{x}_1 = 0,\ \bar{x}_2 = 0,\ \bar{x}_3 = \frac{-8-4e^2}{\pi^2},\ \bar{x}_4 = 0,\ \bar{x}_5 = 0 \\ \bar{x}_6 = \frac{2}{\pi},\ \bar{x}_7 = \frac{2}{\pi},\ \bar{x}_8 = \frac{2}{\pi},\ \bar{x}_9 = 1 \end{array} \right\}$$

AD — Example / Reverse Mode

Next, we use #6 and #7, to finalize #4 and #5. ($x_6 = e^{x_4}$, $x_7 = x_4 * x_5$).



$$\bar{\mathbf{x}}_4 \ \mathbf{+=} \ \frac{\partial f}{\partial x_6}\frac{\partial x_6}{\partial x_4} + \frac{\partial f}{\partial x_7}\frac{\partial x_7}{\partial x_4} = \bar{\mathbf{x}}_6 \cdot e^2 + \bar{\mathbf{x}}_7 \cdot 1 = \frac{2e^2+2}{\pi}$$

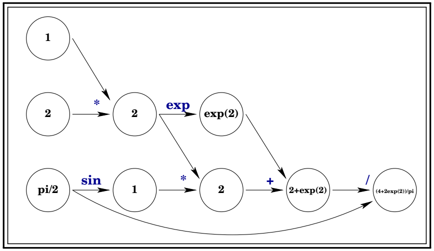
$$\bar{\mathbf{x}}_5 \ \mathbf{+=} \ \frac{\partial f}{\partial x_7}\frac{\partial x_7}{\partial x_5} = \bar{\mathbf{x}}_7 \cdot 2 = \frac{4}{\pi}$$

$$\left\{ \begin{array}{c} \bar{\mathbf{x}}_1 = 0, \ \bar{\mathbf{x}}_2 = 0, \ \bar{\mathbf{x}}_3 = \dfrac{-8-4e^2}{\pi^2} \\ \bar{\mathbf{x}}_4 = \dfrac{2e^2+2}{\pi}, \ \bar{\mathbf{x}}_5 = \dfrac{4}{\pi}, \ \bar{\mathbf{x}}_6 = \dfrac{2}{\pi}, \ \bar{\mathbf{x}}_7 = \dfrac{2}{\pi}, \ \bar{\mathbf{x}}_8 = \dfrac{2}{\pi}, \ \bar{\mathbf{x}}_9 = 1 \end{array} \right\}$$

San Diego State University

AD — Example / Reverse Mode

We now have the final pieces to complete the reverse sweep. ($x_4 = x_1 * x_2$, $x_5 = \sin(x_3)$).



$$\bar{\mathbf{x}}_3 \;\mathbf{+=}\; \frac{\partial f}{\partial x_5}\frac{\partial x_5}{\partial x_3} = \bar{\mathbf{x}}_5 \cdot (-\cos(\pi/2)) = 0$$

$$\bar{\mathbf{x}}_2 \;\mathbf{+=}\; \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_2} = \bar{\mathbf{x}}_4 \cdot 1 = \frac{2e^2+2}{\pi}, \quad \bar{\mathbf{x}}_1 \;\mathbf{+=}\; \frac{\partial f}{\partial x_4}\frac{\partial x_4}{\partial x_1} = \bar{\mathbf{x}}_4 \cdot 2 = \frac{4e^2+4}{\pi}$$

$$\left\{ \begin{array}{c} \bar{\mathbf{x}}_1 = \frac{4e^2+4}{\pi}, \; \bar{\mathbf{x}}_2 = \frac{2e^2+2}{\pi}, \; \bar{\mathbf{x}}_3 = \frac{-8-4e^2}{\pi^2} \\ \bar{\mathbf{x}}_4 = \frac{2e^2+2}{\pi}, \; \bar{\mathbf{x}}_5 = \frac{4}{\pi}, \; \bar{\mathbf{x}}_6 = \frac{2}{\pi}, \; \bar{\mathbf{x}}_7 = \frac{2}{\pi}, \; \bar{\mathbf{x}}_8 = \frac{2}{\pi}, \; \bar{\mathbf{x}}_9 = 1 \end{array} \right\}$$

We have $\nabla f(\bar{\mathbf{x}})\big|_{\bar{\mathbf{x}}=[1,2,\pi/2]^T} = [\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2, \bar{\mathbf{x}}_3]$.

San Diego State University

Reverse Mode Automatic Differentiation: Comments

Main appeal of AD/RM: low computational complexity for scalar functions. Extra arithmetic is at most 4–5 times the "pure" function evaluation.

Main drawback of AD/RM: the entire computational graph must be stored for the reverse sweep. Implementation and access patterns are quite straight-forward, but storage can be a problem:

— If each node can be stored in 20 bytes, then a function that requires one second of evaluation on a 100 megaflop computer may generate a graph which requires 2 Gb of storage!

— The storage requirement can be reduced at the cost of extra arithmetics by partial forward and reverse sweeps; re-evaluating portions instead of storing the whole graph. This is known as **checkpointing**.

Extending AD to Vector Functions

The idea of automatic differentiation is quite easily extended to vector valued functions,

$$F : \mathbb{R}^n \to \mathbb{R}^m,$$

so that the **Jacobian matrix** of first derivatives

$$J(\bar{\mathbf{x}}) = \left[\frac{\partial F_j}{\partial x_i}\right]_{\substack{j\,=\,1,2,\dots,m \\ i\,=\,1,2,\dots,n}}$$

can be formed using automatic differentiation.

All we need is additional bookkeeping for the m components of F.

Extending AD to the Hessian (2nd Derivatives)

The technique can also be adapted to evaluate the Hessian. In **forward mode** we need **2 seed vectors** $\bar{\mathbf{p}}$ and $\bar{\mathbf{q}}$, and we define

$$D_{\bar{\mathbf{p}}\bar{\mathbf{q}}} x_i = \bar{\mathbf{p}}^T \left[\nabla^2 x_i\right] \bar{\mathbf{q}}$$

these 2nd derivative quantities are propagated (using the chain rule) just like the 1st derivatives were propagated in our previous example.

At the end of the sweep the terminal node contains the value of

$$D_{\bar{\mathbf{p}}\bar{\mathbf{q}}} x_n = \bar{\mathbf{p}}^T \left[\nabla^2 x_i\right] \bar{\mathbf{q}} \equiv \bar{\mathbf{p}}^T \left[\nabla^2 f(\bar{\mathbf{x}})\right] \bar{\mathbf{q}}$$

Each pair $(\bar{\mathbf{p}}, \bar{\mathbf{q}}) = (\bar{\mathbf{e}}_i, \bar{\mathbf{e}}_j)$ gives the H_{ij} entry of the Hessian matrix.

Forward-Mode Hessian

Due to symmetry we only need to compute the $j \leq i$ entries H_{ij}, and it is quite clear how to exploit sparsity.

The increase in the number of arithmetic computations compared with evaluation of f alone is a multiplicative factor

$$\sim [1 + n + N_z(\nabla^2 f)]$$

where $N_z(\nabla^2 f)$ is the number of entries of the Hessian we decide to compute.

In **Newton**-CG algorithms we only need the effect of the Hessian-vector product $\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{r}}$, in this case we simply fix the second seed vector $\bar{\mathbf{q}} \equiv \bar{\mathbf{r}}$ and compute the remaining n entries

$$\bar{\mathbf{e}}_j^T \left[\nabla^2 f(\bar{\mathbf{x}})\right] \bar{\mathbf{r}} = \left[\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{r}}\right]_j, \quad j = 1, 2, \dots, n$$

using the forward sweep.

Reverse-Mode Hessian

It is also possible to implement evaluation of the Hessian, $\nabla^2 f(\bar{\mathbf{x}})$, or the Hessian-vector product, $\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{r}}$, in reverse mode.

In the latter case, first both $f(\bar{\mathbf{x}})$ and $\nabla f(\bar{\mathbf{x}})^T\bar{\mathbf{r}}$ are propagated during the forward sweep and the values accumulated in $(x_i, D_{\bar{\mathbf{r}}} x_i)$.

Then we apply the reverse sweep to the computed $\nabla f(\bar{\mathbf{x}})^T\bar{\mathbf{r}}$, At the completion of the reverse sweep we have

$$\frac{\partial}{\partial x_i}\left[\nabla f(\bar{\mathbf{x}})^T\bar{\mathbf{r}}\right] = \left[\nabla^2 f(\bar{\mathbf{x}})\bar{\mathbf{r}}\right]_{i=1,2,\dots,n}$$

in the nodes corresponding to the independent variables.

The increase in work over evaluation of f alone is a multiplicative factor not greater than

$$\sim 12 N_c(\nabla^2 f(\bar{\mathbf{x}}))$$

where $N_c(\nabla^2 f(\bar{\mathbf{x}})) = \#$of (right-)seed vectors used in computation.

Limitation of Automatic Differentiation

1. **$f(x)$ depends on the numerical solution of a PDE**

   In this case $f(\bar{\mathbf{x}})$ may contain truncation errors due to the scheme used to solve the PDE. Even though the truncation errors are small $|\tau(\bar{\mathbf{x}})| < \epsilon$, we cannot control the derivative (gradient) $\tau'(\bar{\mathbf{x}})$, hence errors in AD-computed $f'(\bar{\mathbf{x}})$ can potentially be large.

2. **Perverse code**

   Due to branching (if-statements, etc.) a function evaluation may be equivalent to the following **valid** but nasty piece of code:

   ```
   if (x == 1.0) then { f = 0.0; } else { f = x - 1.0; }
   ```

   Automatic differentiation would most likely give us $f'(1) = 0$, which does not seem like a Good Idea$^{\text{TM}}$.

SAN DIEGO STATE UNIVERSITY

Bottom Line

- Automatic differentiation provide a set of powerful tools/ideas.
- AD can enhance optimization algorithms, and can be applied successfully in many applications involving highly complex functions.
- AD facilitates interpretations of the computed optimal solutions, allowing the user to extract more information from the result.
- **AD does not absolve the user altogether from the responsibility of thinking about derivative calculations, and correctly interpreting and validating the results.**

San Diego State University

Index and References

Reference(s):

- Fournier, David A., Hans J. Skaug, Johnoel Ancheta, James Ianelli, Arni Magnusson, Mark N. Maunder, Anders Nielsen, and John Sibert. *AD Model Builder: using automatic differentiation for statistical inference of highly parameterized complex nonlinear models.* Optimization Methods and Software 27, no. 2 (2012): 233-249.
- Griewank, Andreas, David Juedes, and Jean Utke. *Algorithm 755: ADOL-C: a package for the automatic differentiation of algorithms written in C/C++.* ACM Transactions on Mathematical Software (TOMS) 22, no. 2 (1996): 131-167.
- Rall, Louis B. *Automatic differentiation: Techniques and applications.* (1981).

SAN DIEGO STATE UNIVERSITY