Piecewise Linear AD via Source Transformation

Torsten Bosse[*], Sri Hari Krishna Narayanan[†], Laurent Hascoët[‡]

Overview Algorithmic differentiation (AD) allows the efficient numerical computation of sensitivities for any mathematical function $y = F(x)$, $F : \mathbb{R}^n \to \mathbb{R}^m$ that is sufficiently smooth and given by a finite straight-line code.

$v_{i-n} = x_i$	for $i = 1, \ldots, n$	Initialization
$v_i = \varphi_i(v_j)_{j \prec i}$	for $i = 1, \ldots, l$	Evaluation
$y_{m-i} = v_{l-i}$	for $i = 0, \ldots, m-1$	Extraction

Here, $x \in X \subseteq \mathbb{R}^n$ denotes a vector of input variables, $y = y(x) \in \mathbb{R}^m$ the corresponding output variables, and φ some smooth intermediate assignments from a library $\Phi \equiv \{+, -, *, /, \sqrt{\cdot}, \exp, \sin, \cos, \ldots\}$. However, the smoothness assumption is violated in most real applications. For example, the evaluation routines of many physical applications contain nonsmooth expressions such as the absolute value, the maximum, and/or the minimum function, in order to avoid unrealistic quantities. In this case, the standard differentiation rules do not necessarily apply any more. Thus, the derivatives provided by standard AD tools become unreliable since they are based on the chain rule. Moreover, the simpler models are questionable even if the derivatives are evaluated at points $x \in \mathbb{R}^n$ where the function is differentiable, since they do not take nearby kinks or nonsmoothness into account. A remedy for this situation was recently proposed in [1], where the author presents a method to compute (directional) piecewise linear models of the original abs-factorable function instead of a simple linearization.

The piecewise linearization $\Delta y = \Delta F(x; \Delta x)$, $\Delta F : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^m$, at a point $x \in \mathbb{R}^n$ for a directional increment $\Delta x \in \mathbb{R}^n$ represents the function in a more appropriate way and can be derived by a minor modification of the original code. Similar to the standard forward mode, the idea is based on defining an extended evaluation routine using the propagation rules

$$\begin{aligned}
\Delta v_i &= \Delta v_j \pm \Delta v_k && \text{for } v_i = v_j \pm v_k \\
\Delta v_i &= v_j \Delta v_k + v_k \Delta v_j && \text{for } v_i = v_j * v_k \\
\Delta v_i &= c_{ij} * \Delta v_j && \text{for } v_i = \varphi(v_j) \not\equiv \mathrm{abs}(),
\end{aligned} \tag{1}$$

for all smooth intermediate expressions with corresponding partial derivatives $c_{ij} = (\partial v_i / \partial v_j)_{j \prec i}$. Instead of propagating the direction for the absolute value by $\Delta v_i = \mathrm{sign}(v_j)\Delta v_j$, one employs the rule for the absolute value:

$$\begin{aligned}
\Delta z_j &= v_j + \Delta v_j \\
\Delta v_i &= \mathrm{abs}(\Delta z_j) - v_i \quad \text{for } v_i = \varphi(v_j) \equiv \mathrm{abs}().
\end{aligned} \tag{2}$$

Accordingly, one can define corresponding rules for the maximum and minimum using the identities $\max(p, q) = (p + q + |p - q|)/2$ and $\min(p, q) = (p + q - |p - q|)/2$. As was shown in [2], the extended evaluation routine then provides a piecewise linearization of F, which can be algebraically represented by an abs-normal form (ANF)

$$\begin{bmatrix} \Delta z \\ \Delta y \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix} + \begin{bmatrix} Z & L \\ J & Y \end{bmatrix} \begin{bmatrix} \Delta x \\ |\Delta z| \end{bmatrix}$$

using an additional vector of switching variables $\Delta z \in \mathbb{R}^s$ for some suitable dimensions $m, n, s \in \mathbb{N}$ and matrices/vectors

$$a \in \mathbb{R}^s, \quad Z \in \mathbb{R}^{s \times n}, \quad L \in \mathbb{R}^{s \times s}, \quad b \in \mathbb{R}^m, \quad J \in \mathbb{R}^{m \times n}, \quad Y \in \mathbb{R}^{m \times s}.$$

The matrices might depend on x and can be interpreted as partial derivatives. In detail, if $z = (z_1, \ldots, z_s) \in \mathbb{R}^s$ denotes the arguments of all $s \in \mathbb{N}$ absolute value functions that occur in the original code, then the matrices are

$$Z = \frac{\partial z}{\partial x}, \quad L = \frac{\partial |z|}{\partial z}, \quad Y = \frac{\partial y}{\partial |z|}, \quad J = \frac{\partial y}{\partial x},$$

which consider only those parts of the mappings $x \to z$, $|z| \to z$, $|z| \to y$, and $x \to y$ that only involve smooth expressions. Both a simple evaluation of ΔF for given x and direction Δx and the computation of the complete ANF can be done by using techniques from operator overloading and were already implemented in the AD package ADOL-C [5]. In this paper, we explain our initial development effort for computing the entries of Z, L, J, and Y and the vectors a and b by making simple modifications to the runtime system of the two source transformation tools OpenAD [4] and Tapendade [3]. The method will be demonstrated and validated for some simple test problems.

[*]Corresponding author, Argonne National Laboratory, `tbosse@anl.gov`
[†]Argonne National Laboratory, `snarayan@mcs.anl.gov`
[‡]INRIA Sophia-Antipolis, `laurent.hascoet@inria.fr`

Propagating incremental directions To use the piecewise linear differentiation drivers in OpenAD and Tapenade, the user is required only to replace all calls of the `abs`, `min`, and `max` function in the original code by the stub methods

$$\mathtt{gabs(z,u)}\{\mathtt{u} = \mathtt{abs(z)}\},\ \mathtt{gmin(z1,z2,u)}\{\mathtt{u} = \mathtt{min(z1,z2)}\},\quad \text{and}\quad \mathtt{gmax(z1,z2,u)}\{\mathtt{u} = \mathtt{max(z1,z2)}\},\ \text{respectively.}$$

The stub methods for the nonsmooth parts are then automatically replaced by methods that implement the corresponding propagation rules for the piecewise linear differentiation without changing the original results of F.

Computing the entries of the ANF by using OpenAD For `n` input variables, `m` output variables, and intermediate active variables, the forward mode of OpenAD uses an `active` type containing an array, `d`, of size `n` to propagate directional derivatives according to the propagation rules (2). Thus, if the code is executed in the smooth case, each of the `m` output variables will contain a derivative array that represents one row of the Jacobian matrix `J`, if the derivative array of each of the input variables is assigned to one of the `n` basis vectors.

For the nonsmooth case, the runtime library of OpenAD changes the sizes of the derivative array `d` to `n+s` as described in Fig. 1(a). It also defines a global array `dz` and `du` that each have for any occurring absolute value one vector of size length `n+s`. By default, both arrays `dz` and `du` are initialized such that they form the Euclidean standard basis of dimension `n+s`. The propagation of derivatives ensures that the entries of `J`, `Z`, `L`, and `Y` are computed mechanically. The desired quantities are stored in the corresponding portions of the derivative arrays for the output variables `dz` and `du` as shown in Fig. 1(c). The hand-coded replacement for the absolute value that is required for the computation of the ANF is based on the propagation rules (2) and given in Fig. 1(b).

```
type active
sequence
  real(w2f__8) :: v
  real(w2f__8) , dimension(n+s) :: d = 0.0d0
end type
real(w2f__8) , dimension(s, n+s) :: dz = 0.0d0
real(w2f__8) , dimension(s, n+s) :: du = 0.0d0
```

(a) Modified runtime library

```
subroutine gabs(z, u)
use oad_active
implicit none
type(active) :: z
intent(inout) z
type(active) :: u
intent(inout) u
  oad_abs_s_index = oad_abs_s_index + 1
  du(oad_abs_s_index,1:n+s) = z%d(1:n+s)
  u%v = abs(z%v)
  u%d(1:n+s) = dz(oad_abs_s_index,1:n+s)
end subroutine
```

(b) Custom replacement for differentiated `gabs`

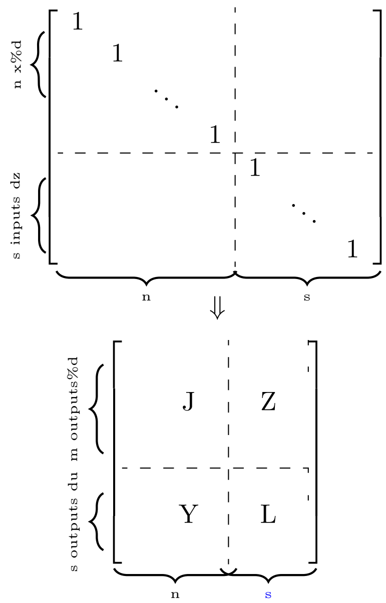

Figure 1: OpenAD code examples for the computation of the ANF and the resulting storage layout for the output variables containing the partial derivatives Z, L, J, and Y

This material is based upon work supported by the U.S. Department of Energy, Office of Science, under contract number DE-AC02-06CH11357.
The submitted manuscript has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne, a U.S. Department of Energy Office of Science laboratory, is operated under Contract No. DE-AC02-06CH11357. The U.S. Government retains for itself, and others acting on its behalf, a paid-up nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. `http://energy.gov/downloads/doe-public-access-plan`

References

[1] A. Griewank. On stable piecewise linearization and generalized algorithmic differentiation. *Optimization Methods and Software*, 28(6):1139–1178, 2013.

[2] A. Griewank, J.-U. Bernt, M. Radons, and T. Streubel. Solving piecewise linear systems in abs-normal form. *Linear Algebra and its Applications*, 471:500–530, 2015.

[3] L. Hascoët and V. Pascual. The Tapenade automatic differentiation tool: Principles, model, and specification. *ACM Transactions on Mathematical Software*, 39(3):20:1–20:43, 2013.

[4] J. Utke. OpenAD: Algorithm implementation user guide. Technical Memorandum ANL/MCS–TM–274, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 2004. available online at `ftp://info.mcs.anl.gov/pub/tech_reports/reports/TM-274.pdf`.

[5] A. Walther and A. Griewank. Getting started with adol-c. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*, chapter 7, pages 181–202. Chapman-Hall CRC Computational Science, 2012.