ALF with source-transformation AD

Laurent Hascoët

INRIA Sophia-Antipolis, France

Shonan, June 2018
Outline

1. ALF explained to a programmer
2. (no) adaption of Tapenade to ALF
3. Distances to nonSmooth
4. Other nonSmooth'es
Tangent AD of nonSmooth codes

- NonSmooth codes alternate smooth parts and nonSmooth bits:
  \[ v_2 = F_{12}(v_1) \]
  \[ v_3 = \text{ABS}(v_2) \]
  \[ v_4 = F_{34}(v_3) \]
  \[ v_5 = \text{ABS}(v_4) \]
  \[ v_6 = F_{56}(v_5) \]

- Classical approach: Tangent AD plus DD on nonSmooth:
  \[ Dv_2 = J_{12} \times Dv_1 \]
  \[ Dv_3 = \text{ABS}(v_2+Dv_2) - \text{ABS}(v_2) \]
  \[ Dv_4 = J_{34} \times Dv_3 \]
  \[ Dv_5 = \text{ABS}(v_4+Dv_4) - \text{ABS}(v_4) \]
  \[ Dv_6 = J_{56} \times Dv_5 \]

Isn’t this what ALF computes?
Searching for efficiency

\[ Dv2 = J12 \ast Dv1 \]
\[ v2 = F12(v1) \]
\[ Dv3 = \text{ABS}(v2+Dv2) - \text{ABS}(v2) \]
\[ v3 = \text{ABS}(v2) \]
\[ Dv4 = J34 \ast Dv3 \]
\[ v4 = F34(v3) \]
\[ Dv5 = \text{ABS}(v4+Dv4) - \text{ABS}(v4) \]
\[ v5 = \text{ABS}(v4) \]
\[ Dv6 = J56 \ast Dv5 \]
\[ v6 = F56(v5) \]
Searching for efficiency

\[ Dv2 = J12 \times Dv1 \]
\[ v2 = F12(v1) \]
\[ Dv3 = \text{ABS}(v2+Dv2) - \text{ABS}(v2) \]
\[ v3 = \text{ABS}(v2) \]
\[ Dv4 = J34 \times Dv3 \]
\[ v4 = F34(v3) \]
\[ Dv5 = \text{ABS}(v4+Dv4) - \text{ABS}(v4) \]
\[ v5 = \text{ABS}(v4) \]
\[ Dv6 = J56 \times Dv5 \]
\[ v6 = F56(v5) \]

**Must run at many** \( Dv1 \)
Searching for efficiency

\[
\begin{align*}
Dv2 &= J12 \times Dv1 \\
v2 &= F12(v1) \\
Dv3 &= \text{ABS}(v2+Dv2) - \text{ABS}(v2) \\
v3 &= \text{ABS}(v2) \\
Dv4 &= J34 \times Dv3 \\
v4 &= F34(v3) \\
Dv5 &= \text{ABS}(v4+Dv4) - \text{ABS}(v4) \\
v5 &= \text{ABS}(v4) \\
Dv6 &= J56 \times Dv5 \\
v6 &= F56(v5)
\end{align*}
\]

Must run at many $Dv1 \Rightarrow$ precompute $Dv1$-independent!
Searching for efficiency

\[ Dv_2 = J_{12} \ast Dv_1 \]
\[ v_2 = F_{12}(v_1) \]

\[ Dv_3 = \text{ABS}(v_2+Dv_2) - \text{ABS}(v_2) \]
\[ v_3 = \text{ABS}(v_2) \]

\[ Dv_4 = J_{34} \ast Dv_3 \]
\[ v_4 = F_{34}(v_3) \]

\[ Dv_5 = \text{ABS}(v_4+Dv_4) - \text{ABS}(v_4) \]
\[ v_5 = \text{ABS}(v_4) \]

\[ Dv_6 = J_{56} \ast Dv_5 \]
\[ v_6 = F_{56}(v_5) \]

Must run at many \( Dv_1 \Rightarrow \text{precompute} \ Dv_1\text{-independent!} \)

Precomputed is linear
Searching for efficiency

\[
\begin{align*}
Dv2 &= J_{12} \ast Dv1 \\
v2 &= F_{12}(v1) \\
Dv3 &= \text{ABS}(v2+Dv2) - \text{ABS}(v2) \\
v3 &= \text{ABS}(v2) \\
Dv4 &= J_{34} \ast Dv3 \\
v4 &= F_{34}(v3) \\
Dv5 &= \text{ABS}(v4+Dv4) - \text{ABS}(v4) \\
v5 &= \text{ABS}(v4) \\
Dv6 &= J_{56} \ast Dv5 \\
v6 &= F_{56}(v5)
\end{align*}
\]

Must run at many \( Dv1 \Rightarrow \text{precompute} \ Dv1\text{-independent!} \)

Precomputed is linear \( \Rightarrow \text{accept affine!} \)
Searching for efficiency

arg1 = (0.25)•Dv1 + 0.6

abs1 = \text{ABS} (arg1)

arg2 = (0.83, -0.5)•(Dv1 abs1) + 0.14

abs2 = \text{ABS} (arg2)

Dv6 = (-0.25, 0.12, 0.62)•(Dv1 abs1 abs2) - 0.07

Must run at many Dv1 \Rightarrow \text{precompute Dv1-independent!}

Precomputed is linear \Rightarrow \text{accept affine!}

Resulting “repeated” is only affine+nonSmooth
Outline

1. ALF explained to a programmer
2. (no) adaption of Tapenade to ALF
3. Distances to nonSmooth
4. Other nonSmooth’es
(no) adaption of Tapenade to ALF

No special development needed to produce ALF!
⇒ uses the current distributed Tapenade

- remove any library-defined special behavior for ABS, MIN, MAX...
- Differentiate as is, Tangent+Vector mode
  
  
  \>$\text{tapenade \ -ext\ noinlinelib\ program.f}$
  \>-head\ "F(y)/(x_1\ x_2)" \ -d \ -vector

- Provide implementation of black-box ABS_DV, MIN_DV, MAX_DV...
- Write a driver to fill gnuplot data.
SUBROUTINE ABS_DV(z, zd, u, ud, nbdirs)
   INCLUDE 'DIFFSIZES.inc'  !defines nbdirsmx = 10
   DOUBLE PRECISION z,zd(nbdirsmx),u,ud(nbdirsmx)
   INTEGER nbdirs, nbabs, affi
   DOUBLE PRECISION absvard(nbabsmax,nbdirsmx)
   COMMON /ABSVARDS/absvard,nbabs
   affi = nbdirs-nbabs  !the index for the affine part of the extended Jac
   nbdirs = nbdirs+1  !add one dimension to the extended Jac
   nbabs = nbabs+1
   absvard(nbabs,:) = zd(:)!incorporate z+Dz into the extended Jac
   absvard(nbabs,affi) = absvard(nbabs,affi) + z
   u = ABS(z)
   ud(:) = 0.d0  !incorporate ABS(z+Dz)-ABS(z) into the ex
   ud(affi) = -u
   ud(nbdirs) = 1.d0
END
One popular illustration

\[ y = \text{MAX}(0, x_1^2 - \text{MAX}(x_1, 0)) \]

Tapenade-generated ALF: \((at \ x_1 = -0.5, \ x_2 = 0.5)\)

<table>
<thead>
<tr>
<th></th>
<th>affine:</th>
<th>(\Delta x_1)</th>
<th>(\Delta x_2)</th>
<th>ABS#1</th>
<th>ABS#2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS#1 arg</td>
<td>0.60</td>
<td>1.00</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS#2 arg</td>
<td>-0.14</td>
<td>-0.50</td>
<td>0.80</td>
<td>-0.50</td>
<td></td>
</tr>
<tr>
<td>final (\Delta y)</td>
<td>-0.07</td>
<td>-0.25</td>
<td>0.40</td>
<td>-0.25</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Outline

1. ALF explained to a programmer
2. (no) adaption of Tapenade to ALF
3. Distances to nonSmooth
4. Other nonSmooth’es
Controlling the size of the ALF

- The Abs Linear Form consumes one new variable per run-time occurrence of a nonSmooth... Costly!
- Can we restrict to the occurrences that are “close”? 
- For every ABS call, we try to evaluate a “distance to switching”, at linearization time.

We can neglect ABS that are “far enough”
Open questions

- Distances to switching become approximate across nonSmooth’es
  Can we solve for $Dx$ in $\text{ALF}(Dx) = Dz$?
- We can drop nonSmooth a priori ($\text{dist} > \text{max}$) . . . but not a posteriori ($\text{new-dist} < \text{current-max-dist}$): needs access to all current derivative vectors!
Outline

1. ALF explained to a programmer
2. (no) adaption of Tapenade to ALF
3. Distances to nonSmooth
4. Other nonSmooth’es
A naïve remark

To an AD developer, ALF is basically:

- make the nonSmooth primitives black-box
- plug in the “true” difference as the nonSmooth differential
- evaluate the linearized “form” as quick dot-products and nonSmooth calls

From this standpoint, ABS could be any nonSmooth... ...but does it make sense or is it useful?
Let’s try anyway...

Use another non-smooth MYJUMP

• ... with 2 arguments!

• ... not even continuous! (jumps)

Hand-write MYJUMP_DV, same pattern as ABS_DV.
Tapenade-generated ALF: \( (at \ x_1 = -0.6, \ x_2 = -0.8) \)

<table>
<thead>
<tr>
<th>affine:</th>
<th>( \Delta x_1 )</th>
<th>( \Delta x_2 )</th>
<th>ABS#1</th>
<th>ABS#2</th>
<th>MYJUMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS#1 arg</td>
<td>0.60</td>
<td>-1.00</td>
<td>-2.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ABS#2 arg</td>
<td>-1.00</td>
<td>1.00</td>
<td>2.00</td>
<td>0.00</td>
<td></td>
</tr>
<tr>
<td>MYJUMP arg#1</td>
<td>0.77</td>
<td>2.56</td>
<td>-3.84</td>
<td>-1.92</td>
<td>1.92</td>
</tr>
<tr>
<td>MYJUMP arg#2</td>
<td>0.20</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.50</td>
<td>0.50</td>
</tr>
<tr>
<td>final ( \Delta y )</td>
<td>-0.74</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.50</td>
<td>0.50</td>
</tr>
</tbody>
</table>