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:

```
v2 = F12(v1)
v3 = ABS(v2)
v4 = F34(v3)
v5 = ABS(v4)
v6 = F56(v5)
```

- Classical approach: Tangent AD plus DD on nonSmooth:

```
Dv2 = J12 * Dv1
Dv3 = ABS(v2+Dv2) - ABS(v2)
Dv4 = J34 * Dv3
Dv5 = ABS(v4+Dv4) - ABS(v4)
Dv6 = J56 * Dv5
```

Isn't this what ALF computes?

Searching for efficiency

```
Dv2 = J12 * Dv1
v2 = F12(v1)
Dv3 = ABS(v2+Dv2) - ABS(v2)
v3 = ABS(v2)
Dv4 = J34 * Dv3
v4 = F34(v3)
Dv5 = ABS(v4+Dv4) - ABS(v4)
v5 = ABS(v4)
Dv6 = J56 * Dv5
v6 = F56(v5)
```

Searching for efficiency

```
Dv2 = J12 * Dv1
v2 = F12(v1)
Dv3 = ABS(v2+Dv2) - ABS(v2)
v3 = ABS(v2)
Dv4 = J34 * Dv3
v4 = F34(v3)
Dv5 = ABS(v4+Dv4) - ABS(v4)
v5 = ABS(v4)
Dv6 = J56 * Dv5
v6 = F56(v5)
```

Must run at many `Dv1`

Searching for efficiency

```
Dv2 = J12 * Dv1
v2 = F12(v1)
Dv3 = ABS(v2+Dv2) - ABS(v2)
v3 = ABS(v2)
Dv4 = J34 * Dv3
v4 = F34(v3)
Dv5 = ABS(v4+Dv4) - ABS(v4)
v5 = ABS(v4)
Dv6 = J56 * Dv5
v6 = F56(v5)
```

Must run at many `Dv1` $\Rightarrow$ **precompute** `Dv1`-**independent!**

Searching for efficiency

```
Dv2 = J12 * Dv1
v2 = F12(v1)
Dv3 = ABS(v2+Dv2) - ABS(v2)
v3 = ABS(v2)
Dv4 = J34 * Dv3
v4 = F34(v3)
Dv5 = ABS(v4+Dv4) - ABS(v4)
v5 = ABS(v4)
Dv6 = J56 * Dv5
v6 = F56(v5)
```

Must run at many `Dv1` $\Rightarrow$ **precompute** `Dv1`**-independent!**

Precomputed is linear

Searching for efficiency

```
Dv2 = J12 * Dv1
v2 = F12(v1)
Dv3 = ABS(v2+Dv2) - ABS(v2)
v3 = ABS(v2)
Dv4 = J34 * Dv3
v4 = F34(v3)
Dv5 = ABS(v4+Dv4) - ABS(v4)
v5 = ABS(v4)
Dv6 = J56 * Dv5
v6 = F56(v5)
```

Must run at many `Dv1` $\Rightarrow$ **precompute** `Dv1`**-independent!**

Precomputed is linear $\Rightarrow$ **accept affine!**

Searching for efficiency

$$\texttt{arg1} = (0.25)\bullet\texttt{Dv1} + 0.6$$

$$\texttt{abs1} = \boxed{\texttt{ABS(arg1)}}$$

$$\texttt{arg2} = (0.83, -0.5)\bullet(\texttt{Dv1 abs1}) + 0.14$$

$$\texttt{abs2} = \boxed{\texttt{ABS(arg2)}}$$

$$\texttt{Dv6} = (-0.25, 0.12, 0.62)\bullet(\texttt{Dv1 abs1 abs2}) - 0.07$$

Must run at many `Dv1` $\Rightarrow$ precompute `Dv1`-independent!

Precomputed is linear $\Rightarrow$ 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

```
$> tapenade -ext noinlinelib program.f
     -head "F(y)/(x1 x2)" -d -vector
```

- Provide implementation of black-box `ABS_DV`, `MIN_DV`, `MAX_DV`...
- Write a driver to fill `gnuplot` data.

ABS_DV

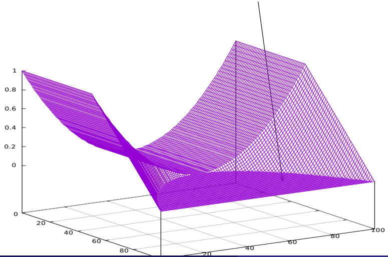
```
SUBROUTINE ABS_DV(z, zd, u, ud, nbdirs)
 INCLUDE 'DIFFSIZES.inc'  !defines nbdirsmax = 10
 DOUBLE PRECISION z,zd(nbdirsmax),u,ud(nbdirsmax)
 INTEGER nbdirs, nbabs, affi
 DOUBLE PRECISION absvard(nbabsmax,nbdirsmax)
 COMMON /ABSVARDS/absvard,nbabs
 affi = nbdirs-nbabs       !the index for the affine part of the extended
 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
```

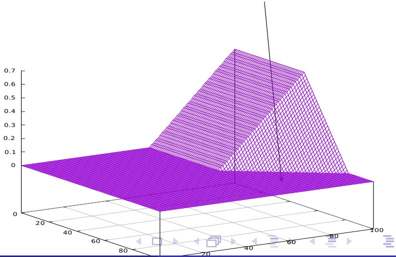
One popular illustration

$$y = \mathtt{MAX}(0, x_2^2 - \mathtt{MAX}(x_1, 0))$$

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

	affine:	Δx_1	Δx_2	ABS#1	ABS#2
ABS#1 arg	0.60	1.00	0.00		
ABS#2 arg	-0.14	-0.50	0.80	-0.50	
final Δy	-0.07	-0.25	0.40	-0.25	0.50





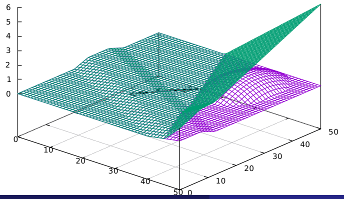
Outline

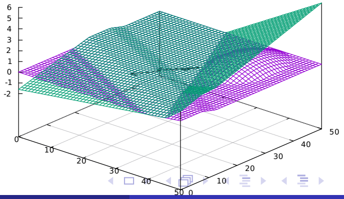
1. ALF explained to a programmer
2. (no) adaption of Tapenade to ALF
3. Distances to nonSmooth
4. Other nonSmooth'es

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 evalate 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 $\mathtt{Dx}$ in $\text{ALF}(\mathtt{Dx}) = \mathtt{Dz}$?
- We can drop nonSmooth a priori (dist>max)...
  but not a posteriori (new-dist<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`.

Results

Tapenade-generated ALF: $(at\ x_1 = -0.6,\ x_2 = -0.8)$

	affine:	Δx_1	Δx_2	ABS#1	ABS#2	MYJUMP
ABS#1 arg	0.60	-1.00	-2.00			
ABS#2 arg	-1.00	1.00	2.00	0.00		
MYJUMP arg#1	0.77	2.56	-3.84	-1.92	1.92	
MYJUMP arg#2	0.20	0.00	0.00	-0.50	0.50	
final Δy	-0.74	0.00	0.00	-0.50	0.50	1.00