Certification of Derivatives Computed by Automatic Differentiation

Mauricio Araya Polo & Laurent Hascoët

INRIA SOPHIA ANTIPOLIS

Project TROPICS

WSEAS, Cancún, México, May 13, 2005.

Plan

- **Introduction** *(Background)*
- **Automatic Differentiation**
  - **Direct Mode**
  - **Reverse Mode**
- **The Problem**
  - **Example**
- **Our Approach**
  - **Description**
  - **Numerical Result**
- **Conclusions**
- **Future Work**

Introduction *(Background)*

- *Automatic Differentiation (A.D.) :* **Given program that evaluates function F, builds new program that evaluates derivatives of F.**

- *Scientific Applications :* **Derivatives are useful in optimization, sensitivity analysis and inverse problems.**

- *Non-differentiability :* **Introduced in programs by conditional statements (tests). May produced wrong derivatives.**

- *Lack of Validation :* **A.D. models (neither A.D Tools) include verification of the differentiability of the functions.**

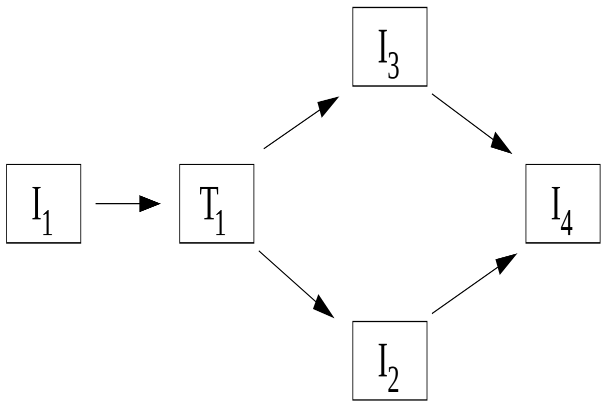
- *Novel A.D. model with Validation :* **We evaluate interval around input data where no non-differentiability problem arises, this information propagated through conditional statements.**

Automatic Differentiation

Programs Structure: set of concatenated sequence of instructions I_i

$$P = I_1; I_2; ...; I_{p-1}; I_p$$

but control flow (flowgraph):



depending on the inputs the example program might be:

$$P = I_1; T_1; I_2; I_4$$

or

$$P = I_1; T_1; I_3; I_4$$

instruction T_1 represents the conditional statement (test).

Mathematical Models: composition of elementary functions f_i

$$Y = F(X) = f_p \circ f_{p-1} \circ ... \circ f_2 \circ f_1$$

Program P evaluates the model F, for every function f_i we have a computational representation I_i, in right order.

Automatic Differentiation (2)

Direct Mode: directional derivatives.

$$Y' = F'(X) \cdot dX = f'_p(x_{p-1}) \cdot f'_{p-1}(x_{p-2}) \cdot \ldots \cdot f'_1(x_0) \cdot dX$$

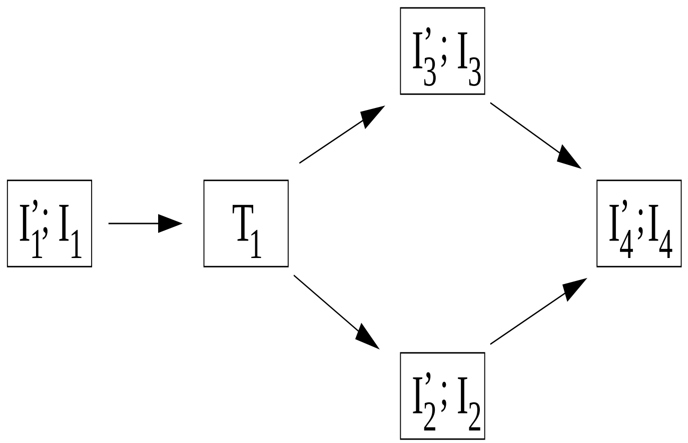
with $x_i = f_i \circ \ldots \circ f_1$**, and** $f'_i()$ **jacobians.**

then the new program P',

$$P' = I'_1; I_1; I'_2; I_2; \ldots; I'_{p-1}; I_{p-1}; I'_p$$

with I'_i **corresponding to** $f'_i()$

flowgraph again:



depending on the inputs the differentiated example program might be:

$$P = I'_1; I_1; T_1; I'_2; I_2; I'_4; I_4$$

or

$$P = I'_1; I_1; T_1; I'_3; I_3; I'_4; I_4$$

the differentiated example program retains the control flow structure of the original program.

Automatic Differentiation (3)

<table>
<tr><th>Original Code</th><th>Direct Differentiated Code</th></tr>
<tr><td>

subroutine sub1(x,y,o1)

I_1 $x = y * x$
I_2 $o1 = x * x + y * y$

T_1 **if (** $o1 > 190$**) then**
I_3 $o1 = -o1 * o1/2$
else
I_4 $o1 = o1 * o1 * 20$
endif
end

</td><td>

subroutine sub1_d(x, xd, y, yd, o1, o1d)

I'_1 $xd = yd * x + y * xd$
I_1 $x = y * x$

I'_2 $o1d = 2 * x * xd + 2 * y * yd$
I_2 $o1 = x * x + y * y$

T_1 **if (**$o1 > 190$**) then**
I'_3 $o1d = -(o1d * o1)$
I_3 $o1 = -(o1 * o1/2)$
else
I'_4 $o1d = 40 * o1d * o1$
I_4 $o1 = o1 * o1 * 20$
endif
end

</td></tr>
</table>

Table 1: Example of Direct Mode of AD.

Automatic Differentiation (3)

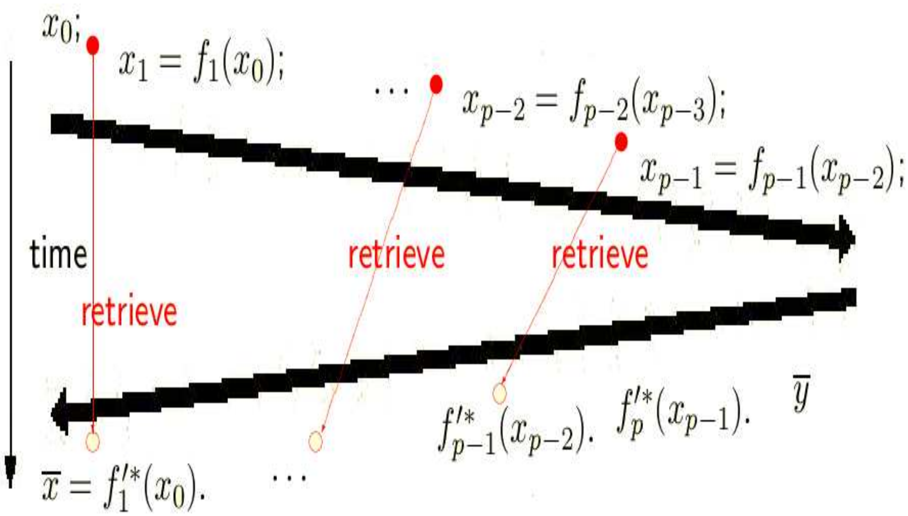
Reverse Mode: adjoints, gradients.

$$\bar{X} = F'^{*}(X) \cdot \bar{Y} = f_1'^{*}(x_0) \cdot f_2'^{*}(x_1) \cdot \ldots \cdot f_p'^{*}(x_{p-1}) \cdot \bar{Y}$$

then the new program $\bar{P}$,

$$\bar{P} = I_1; I_2; \ldots; I_{p-1}; I_p; \bar{I_p}; \bar{I}_{p-1}; \ldots; \bar{I_2}; \bar{I_1} \quad or \quad \bar{P} = \overrightarrow{P}; \overleftarrow{P}$$

with $\bar{I_i}$ corresponding to $f_i'^{t}()$.



Remark: **The reverse sweep ($\overleftarrow{P}$) eventually needs some values of the forward sweep ($\overrightarrow{P}$), but x_0 and others x_i might be modified by the forward sweep, thus we have to store them, which for some programs leads to important memory consumption.**

Automatic Differentiation (4)

Original Code	Reverse Differentiated Code

subroutine sub1(x,y,o1)
 	
I_1 $x = y * x$
I_2 $o1 = x * x + y * y$
 	
T_1 if ($o1 > 190$) then
I_3 $o1 = -o1 * o1/2$
else
I_4 $o1 = o1 * o1 * 20$
endif
end
	subroutine sub1_b(x, xb, y, yb, o1, o1b)
	
	PUSH(x)
	I_1 $x = y * x$
	I_2 $o1 = x * x + y * y$
	
	T_1 if ($o1 > 190$) then
	$\overleftarrow{I_3}$ $o1b = -(o1 * o1b)$
	else
	$\overleftarrow{I_4}$ $o1b = 40 * o1 * o1b$
	endif
	
	$\overleftarrow{I_2}$ $\left\{ \begin{array}{l} xb = xb + 2 * x * o1b \\ yb = yb + 2 * y * o1b \end{array} \right.$
	
	POP(x)
	$\overleftarrow{I_1}$ $\left\{ \begin{array}{l} yb = yb + x * xb \\ xb = y * xb \end{array} \right.$
	end

Table 2: Example of Reverse Mode of AD.

The Problem

Motivation:

The question of derivatives being valid only in a certain domain is a crucial problem of AD. If derivatives returned by AD are used outside their domain of validity, this can result in errors that are very hard to detect.

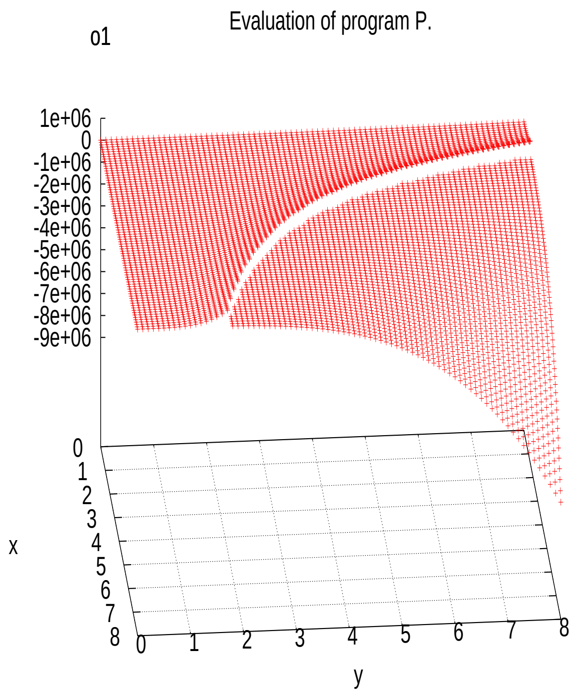
Description:

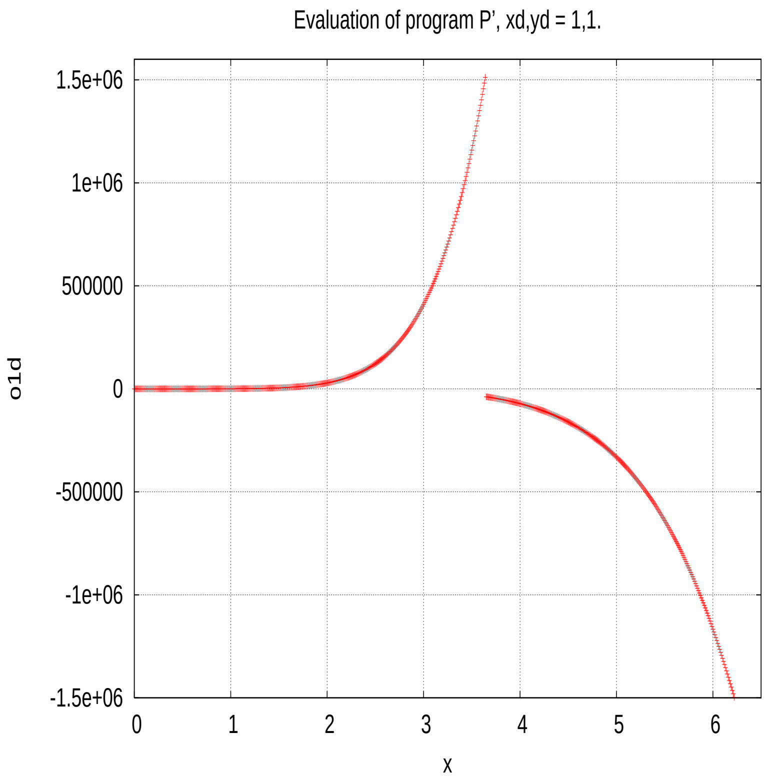
- **Programs have control flow structure, including conditional statements (tests). Some of the test are introduced by intrinsic functions like abs, min, max, etc.**

- **Differentiated program keeps the control flow structure of given program. Sometimes the derivatives depends in the control flow structure.**

- **When some input is too close to a switch of the control flow, the resulting derivative may be very different or wrong, to the point of be useless.**

The Problem (2)

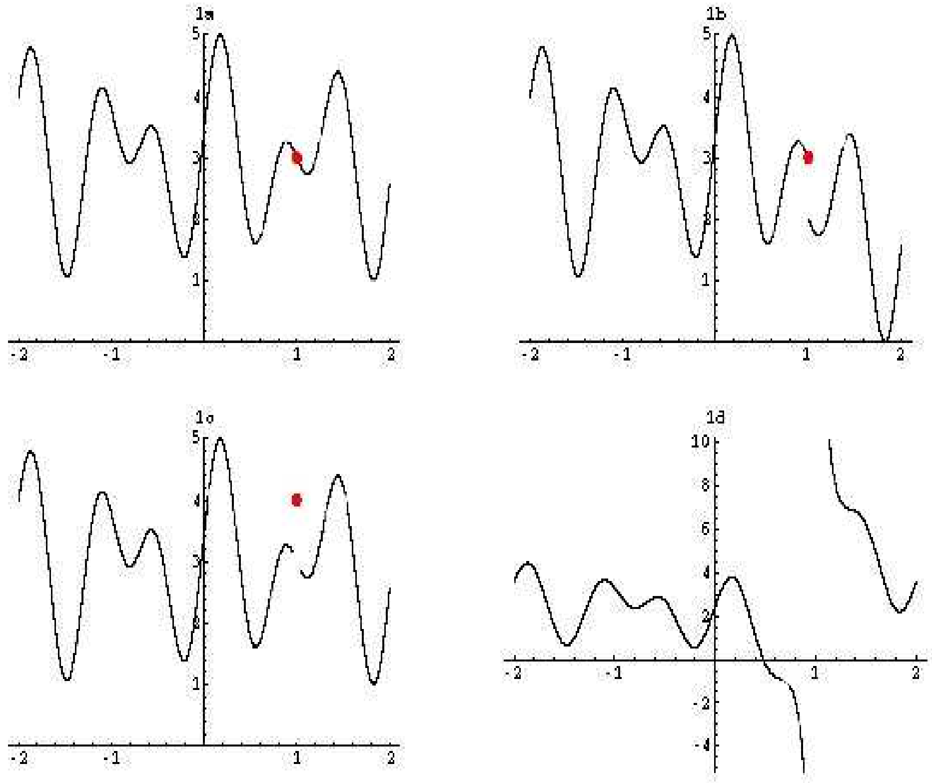




Plot of left shows the evaluation of program example with discontinuity problem. Plot of right shows the evaluation of differentiated program example with input space direction (1,1).

(x=3.64,o1d=1512117.125) and (x=3.65,o1d=-38513.449) !!!

The Problem (3)



Main cases of problems introduced by conditional statements.
(from B. Kearfott paper)

Our Approach

- every test (t) is analyzed, under small change in the input the test must remain in the same “side” of the inequality.

variables used by instructions

needed to built the current test

$$[redacted] + t_i \geq 0 \qquad (1)$$

- [redacted] have to be expressed in terms of the intermediates variables (B_i).

$$\Delta t_i = J(T_i) \cdot \Delta B_i$$

- and the variation of the intermediates variables is

$$\Delta B_i = J(B_i; \ldots; B_0) \cdot \Delta X = J(B_i) \cdot \ldots \cdot J(B_0) \cdot \Delta X$$

where ΔX represents the variation of the inputs values.

- re-composing the expression $\Delta t_i + t_i \geq 0$ from (1),

$$< J(T_i) \cdot J(B_i) \cdot \ldots \cdot J(B_0) \cdot \Delta X | e_j > \geq - < t_i | e_j > \qquad (2)$$

Our Approach (2)

- we want isolate ΔX, a good way to do that is transpose the jacobians in (2)

$$< \Delta X \cdot J(B_0)^* \cdot ... \cdot J(B_i)^* \cdot J(T_i)^* \cdot e_j > \geq - < t_i | e_j > \tag{3}$$

- we can use the reverse mode of AD to compute $J(B_0)^* \cdot ... \cdot J(B_i)^* \cdot J(T_i)^* \cdot e_j$ in (3).
- unfortunately, in real situations the number of tests is so large that the computation of this approach is not practical.
- Solutions:
  - combine constraints to propagate just one. half-spaces.
  - reduce the size of the problem. less tests or less inputs, or both.

Our Approach (3)

- **we analyze one test (t_0), under small change in the input the test must remain in the same "side" of the inequality.**

$$if\ \ t_0 \geq 0\ \ then\ \ \Delta t_0 + t_0 \geq 0 \tag{4}$$

- **the variation of t (Δt_0) have to be expressed in terms of the intermediates variables (B_0).**

$$\Delta t_0 = J(T_0) \cdot \Delta B_0$$

- **and the variation of the intermediates variables is**

$$\Delta B_0 = J(B_0) \cdot \beta \cdot \dot{X}$$

  where $\beta \cdot \dot{X}$ represents the variation of the inputs values. β the magnitude and $\dot{X}$ the direction of the variation.

- **re-composing the expression (4),**

$$\beta \cdot J(T_0) \cdot J(B_0) \cdot \dot{X} \geq -t_0$$

Our Approach (4)

the following expression give us the magnitude of change of the input values, without change the sign of the test.

$$\beta \geq \frac{-t_0}{J(T_0) \cdot J(B_0) \cdot \dot{X}} \tag{5}$$

to compute expression (5) we introduced a function call that propagate the effect of every test trough the program, resulting in a interval of validity, as follows:

Direct Differentiated Code	Direct Differentiated Code with Validation

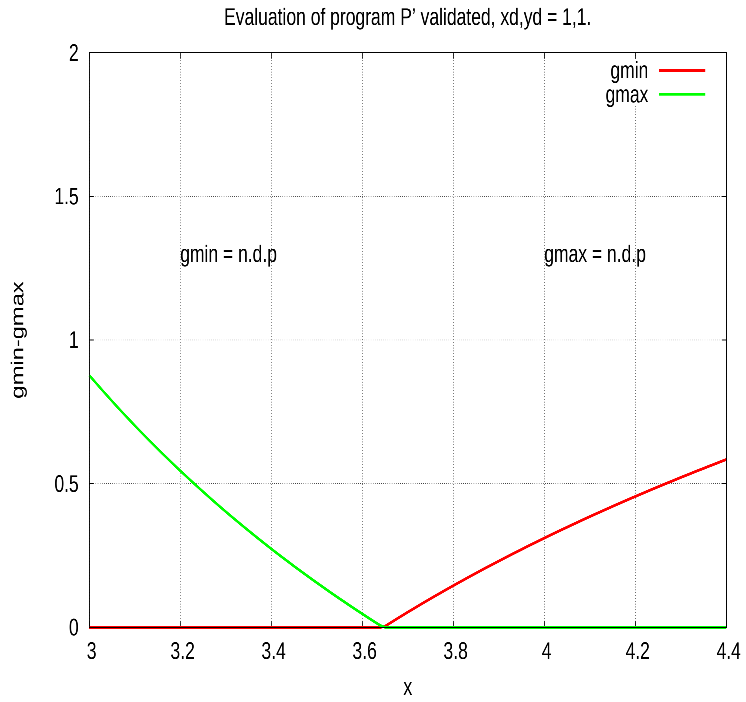
subroutine sub1_d(x,xd,y,yd,o1,o1d)
 	
I_1' $xd = yd * x + y * xd$
I_1 $x = y * x$
I_2' $o1d = 2 * x * xd + 2 * y * yd$
I_2 $o1 = x * x + y * y$
 	
T_1 if ($o1 > 190$) then
I_3' $o1d = -(o1d * o1)$
I_3 $o1 = -(o1 * o1/2)$
else
I_4' $o1d = 40 * o1d * o1$
I_4 $o1 = o1 * o1 * 20$
endif
end
	subroutine sub1_dva(x,xd,y,yd,o1,o1d)
	
	I_1' $xd = yd * x + y * xd$
	I_1 $x = y * x$
	I_2' $o1d = 2 * x * xd + 2 * y * yd$
	I_2 $o1 = x * x + y * y$
	
	V_1 CALL VALIDITY_TEST(o1 - 190, o1d)
	T_1 if ($o1 > 190$) then
	I_3' $o1d = -(o1d * o1)$
	I_3 $o1 = -(o1 * o1/2)$
	else
	I_4' $o1d = 40 * o1d * o1$
	I_4 $o1 = o1 * o1 * 20$
	endif
	end

Our Approach (5)

- In the example, the β magnitude is:

$$\beta \geq \frac{-(o1 - 190)}{o1d} = \frac{190 - (x^2 + y^2)}{2 \cdot x \cdot (yd \cdot x + y \cdot xd) + 2 \cdot y \cdot yd}$$

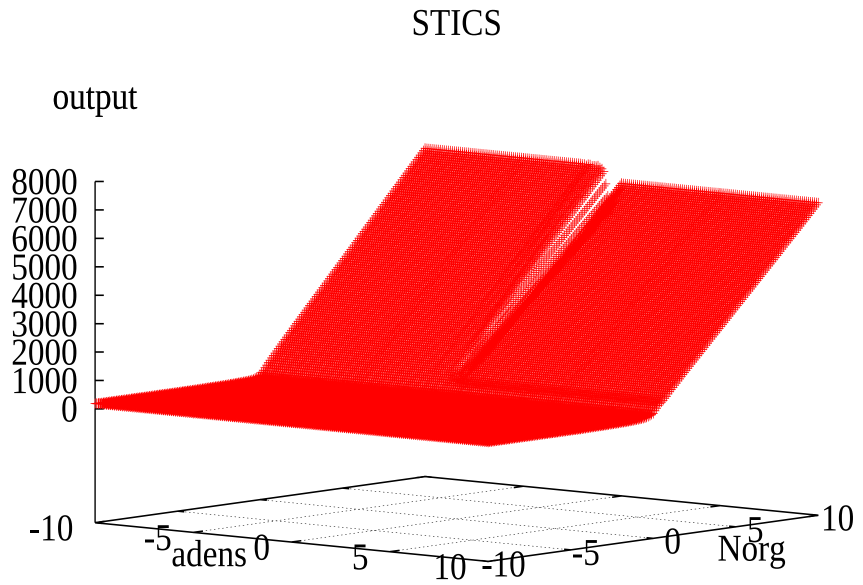
- We can access global variables *gmin* and *gmax*, which hold the upper and lower bounds of the validity interval. The numerical results of the example are:



x	o1d	gmin	gmax
3.60	1402902.000	n.d.p	0.046
3.61	1429547.625	n.d.p	0.036
3.62	1456628.250	n.d.p	0.026
3.63	1484149.250	n.d.p	0.016
3.64	1512117.125	n.d.p	0.005
3.65	-38513.449	0.004	n.d.p
3.66	-39235.445	0.014	n.d.p
3.67	-39969.062	0.023	n.d.p
3.68	-40714.464	0.033	n.d.p
3.69	-41471.812	0.043	n.d.p

Our Approach (6)

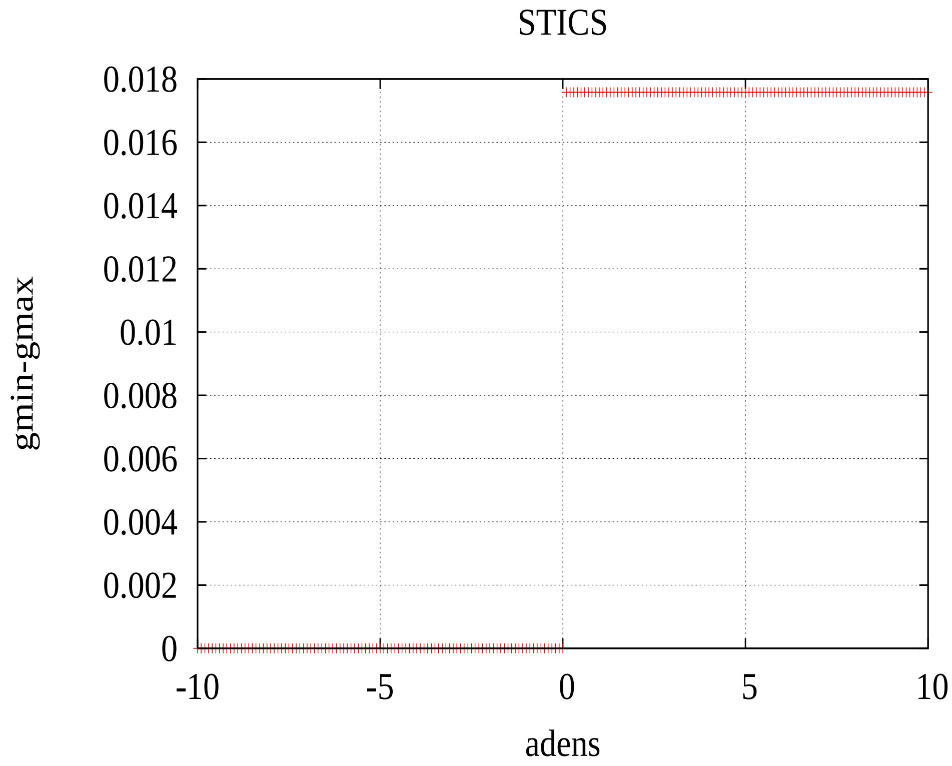
The following numerical result was obtained using a CFD solver STICS, 21.200 l.o.c., the differentiated version has 59.320 l.o.c, 542 validated tests from 2.582 total tests.



STICS with input=(norg,adens,+) and output=(azomes,qnplante,resmes,+).

Our Approach (7)

- **Preliminary results of validation.**



STICS with input=(norg=4,adens,+) and output=gmin.

Conclusions

- Users overlook the problem of wrong derivatives due changes in control-flow. AD tools must be able to detect this kind of situation and provide warning.

Project Tropics, INRIA.

- [illegible] tion about valid domains of input f[illegible] ect differentiated programs.

- The proposed model was developed inside the A.D Tool Tapenade. http://www-sop.inria.fr/tropics.

- The overhead of use our method is only 3% over plain direct mode, this figure was obtain testing the model in several real-life codes.

Future Work

- Integrate the approach to real-life algorithms, applications. (underway). Promising adaptation to Non-Smooth Optimization.

- Extend the approach (or propose a new one) for the reverse mode of AD.

Bibliography

- **Araya-Polo M., Hascoët L.,** *Domain of Validity of Derivatives Computed by Automatic Differentiation*, **Rapport de Recherche RR-5237, INRIA Sophia-Antipolis, 2004.**

- **Hascoët, L., Pascual, V.,** *TAPENADE 2.1 User's guide*. **Technical report #224. INRIA, 2004. http://www-sop.inria.fr/tropics.**

- **Berz, M., Bischof, G., Corliss, G., and Griewank, editors.** *Computational Differentiation: Techniques, Applications, and Tools*. **SIAM, Philadelphia, PA, 1996.**

- **Corliss, G., Faure, Ch., Griewank, A., Hascoët, L., and Naumann, U.** *Automatic Differentiation of Algorithms, from Simulation to Optimization*, **Springer, Selected papers from AD2000, 2001.**

- **Kearfott, R. B.,** *Treating Non-Smooth Functions as Smooth Functions in Global Optimization and Nonlinear Systems Solvers*, **Scientific Computing and Validated Numerics, ed. G. Alefeld a nd A. Frommer, Akademie Verlag, pp. 160-172, 1996.**

TROPICS Project, INRIA Sophia-Antipolis

Team

Laurent Hascoet (leader)
Valérie Pascual
Benjamin Dauvergne
Stephen Wornom
Alain Dervieux
Christophe Massol
Bruno KOOBUS
Mauricio Araya
and Nathalie Bellesso.

Theme

- Scientific Computing and Optimisation.
- Computer Science for analysis and transformation of scientific programs. (Parallelization and Differentiation).

Tool

TAPENADE: analysis and A.D. of source programs.

Applications

- Sensitivity Analysis.
- Optimum Design (Aeronautics).
- Inverse Problems & Data Assimilation (Weather forecast).

Questions?