# Improved Interval Constraint Propagation for Constraints on Partial Derivatives *

Evgueni Petrov and Frédéric Benhamou

IRIN — Université de Nantes
2 rue de la Houssinière BP 92208
44322 Nantes Cedex 03 France
{evgueni.petrov, frederic.benhamou}@irin.univ-nantes.fr

**Abstract.** Automatic differentiation (AD) automatically transforms programs which calculate elementary functions into programs which calculate the gradients of these functions. Unlike other differentiation techniques, AD allows one to calculate the gradient of any function at the cost of at most 5 values of the function (in terms of time). Interval constraint programming (ICP) is a part of constraint programming focused on representation and processing of nonlinear constraints. We adapt AD to the context of ICP and obtain an algorithm which transforms elementary functions into constraints specifying their gradient. We describe some experiments with implementation of our algorithm in the logic programming language ECL$^i$PS$^e$.

## 1 Introduction

Automatic differentiation (AD) automatically transforms programs which calculate real functions into programs which calculate the gradients of these functions [7, 5]. Unlike other differentiation techniques, AD allows one to calculate the gradient at the cost of at most 5 values of the function (in terms of time).

Interval constraint programming (ICP) [1] is a part of constraint programming [6, 2] focused on representation and processing of nonlinear constraints. One of classical concepts of ICP is interval constraint propagation which means incremental calculation of multidimensional rectangles bounding solutions to nonlinear constraints.

Applications of AD in ICP are limited to fast calculation of the first coefficients of interval Taylor series of nonlinear constraints [8, 4].

We adapt AD to the context of ICP and obtain an algorithm which transforms elementary functions into constraints specifying their gradient. With respect to the techniques of naïve symbolic differentiation, our algorithm generates a smaller number of constraints which are more effectively processed by interval constraint propagation.

* Financially supported by Centre Franco-Russe Liapunov (Project 06–98), by European project COCONUT IST–2000–26063.

The paper is structured as follows. Section 2 introduces basic notions of the paper. Section 3 describes our transformation technique. Section 4 states its properties (complexity and accuracy of interval constraint propagation). Section 5 contains the data of experiments with an implementation of our technique in the logic programming language ECL$^i$PS$^e$ [3]. Section 6 concludes the paper.

## 2 Definitions, notation

We give some definitions first. Real intervals are closed convex subsets of the set $\mathbf{R}$ of real numbers. Real relations are subsets of points of $\mathbf{R}^2$, $\mathbf{R}^3$, etc. Symbol $\mathbf{R}^\infty$ denotes the set of countable sequences of real numbers. A projection function is a function which returns a specific component of these sequences.

Symbols $v_0$, $v_1$, $v_2$, etc. denote the formal variables. Constraints are pairs consisting of a relation and an ordered set of variables. A constraint is primitive, if it relates its variables by the graph of some "basic" function. We assume that (1) every basic function has one or two arguments, (2) the arithmetic operations, the trigonometric functions, the functions exp, log are basic functions. Sequence $p \in \mathbf{R}^\infty$ is a solution to constraint $((v_i, v_j, \ldots), f)$, if it satisfies $(p_i, p_j, \ldots) \in f$.

Real terms are terms constructed of the formal variables, real numbers, and the basic functions. Real terms specify real functions of the sequences from $\mathbf{R}^\infty$. The variables specify the projection functions, real numbers specify the constant functions, compound real terms $f(t)$, $f(t, t')$ specify the composition of basic function $f$ and the functions specified by the real terms $t$, $t'$.

Symbol $D_\ell$ denotes differentiation operator which maps every real function to its partial derivative with respect to argument $\ell$.

Templates of real terms are terms which are constructed of the formal variables, variables $\blacksquare_1$, $\blacksquare_2$, real numbers, the basic functions, and which do not contain multiple occurrences of $\blacksquare_1$, $\blacksquare_2$.

Let $F$, $t$, $t'$ be a template of real term and two real terms. Expression $F\langle t, t'\rangle$ denotes the result of replacing $\blacksquare_1$, $\blacksquare_2$ with $t$, $t'$. If $F$ does not contain $\blacksquare_2$, then $t'$ is omitted. Partial derivatives of basic function $f$ are expressed by templates $f'_x$, $f'_y$, i.e., real terms $f'_x\langle v_0, v_1\rangle$, $f'_y\langle v_0, v_1\rangle$ specify non-zero partial derivatives of the real function specified by real term $f(v_0, v_1)$. The derivative of unary basic function $f$ is expressed by template $f'$.

## 3 AD in terms of constraints

Before going into details, we explain our idea informally in terms of program analysis. Linear blocks are sequences of assignments. Single assignment programs are programs in which every variable is assigned a value only once. As a program transformation technique, AD has an important property: it transforms linear blocks into linear blocks, single assignment programs into single assignment programs. The class of single assignment programs consisting of a signle linear block is equivalent to the class of sets of primitive constraints (because,

in such context, every assignment is a valid equation). Thus, AD can be made applicable to functions specified by primitive constraints. In what follows, we give a more detailed description of this idea.

Given a real term which specifies some function $h$ and does not contain other variables than $v_0$, $v_2$, ..., $v_{2n}$, our algorithm generates such set $C$ of primitive constraints that the projection of the set of solutions to $C$ onto the first $2n+2$ coordinates is the following subset of $\mathbf{R}^{2n+2}$ (see theorem 1):

$$\{(p_0, \ldots, p_{2n+1}) \mid \forall \ell \in [0, n]\ p_{2\ell+1} = (D_{2\ell} h)(p)\}\,.$$

### 3.1 Decomposition algorithm

The following rules define function $\mathsf{ad}$ which implements our decomposition algorithm. The rule to be applied should be selected in the "top down" fashion. Because rule 1 does not commute with rules 3, 4, this assumption is essential. Rules 1 through 4 assume that $2i$ is greater than the subscript of any variable occurring in their left hand sides ($v_{2i}$ must be a "free" variable). Function $\mathsf{dec}$ maps real terms to sets of primitive constraints. It returns primitive constraints whose solutions annihilate the real function specified by its argument (see implementation in section 4).

**Rule 1 (Elimination of multiple occurrences)**

$$\mathsf{ad}F\langle v_{2j}, v_{2j}\rangle = \mathsf{ad}F\langle v_{2i}, v_{2i+2}\rangle \cup \{v_{2j} = v_{2i}, v_{2j} = v_{2i+2}, v_{2j+1} = v_{2i+1} + v_{2i+3}\}$$

**Rule 2 (Elimination of constants)** $\mathsf{ad}F\langle c\rangle = \mathsf{ad}F\langle v_{2i}\rangle \cup \{v_{2i} = c\}$

**Rule 3 (Composition-1)**

$$\mathsf{ad}F\langle f(v_{2j})\rangle = \mathsf{ad}F\langle v_{2i}\rangle \cup \{v_{2i} = f(v_{2j})\} \cup \mathsf{dec}\,(v_{2j+1} - v_{2i+1} f'\langle v_{2j}\rangle)$$

**Rule 4 (Composition-2)**

$$\begin{gathered}\mathsf{ad}F\langle f(v_{2j}, v_{2k})\rangle = \mathsf{ad}F\langle v_{2i}\rangle \cup \{v_{2i} = f(v_{2j}, v_{2k})\} \cup \\ \mathsf{dec}\,(v_{2j+1} - v_{2i+1} f'_x\langle v_{2j}, v_{2k}\rangle) \cup \mathsf{dec}\,\big(v_{2k+1} - v_{2i+1} f'_y\langle v_{2j}, v_{2k}\rangle\big)\end{gathered}$$

**Rule 5 (Termination)** $\mathsf{ad}v_{2j} = \{v_{2j+1} = 1\}$

In the ECL$^i$PS$^e$ implementation of $\mathsf{ad}$, in order to reduce the number of generated primitive constraints, we simplify the right hand side of rules 3, 4 for the basic functions exp, sqrt, arctan and replace rule 2 with four additional ones which process the cases where one or two arguments of some basic function are real numbers (one instance of rule 3 and three instances of rule 4).

Real terms specify real functions. Our rules reduce calculation of the gradient of one real function (specified by a real term) to calculation of the gradient of some other (specified by some other term). Figures 1, 2, 3 illustrate how our rules change the graphs of these functions in some particular cases.

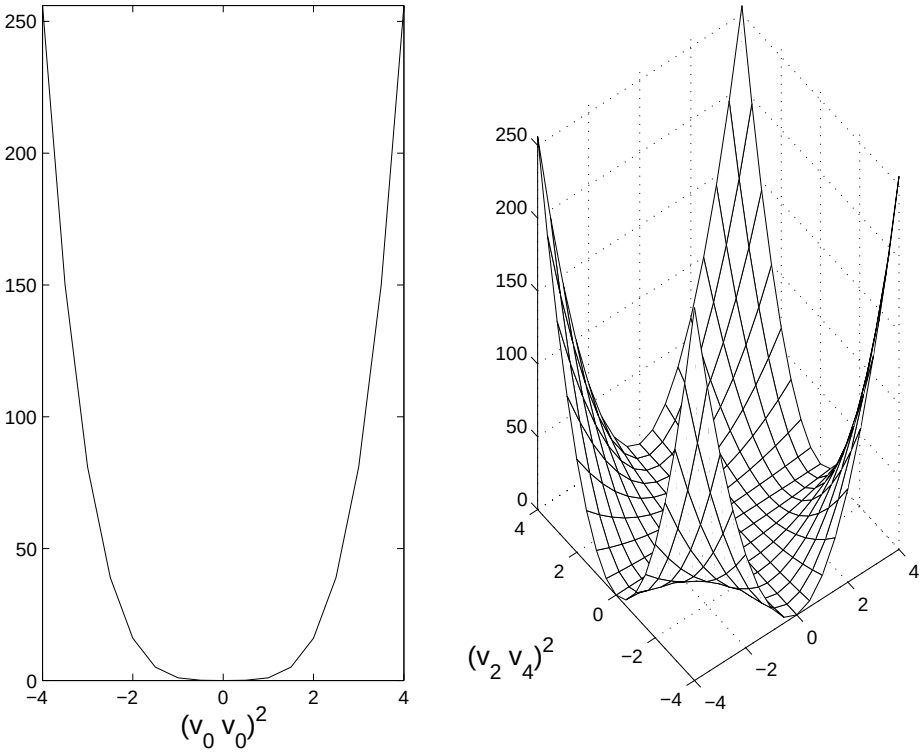


**Fig. 1.** Application of rule 1 to $(v_0 v_0)^2$.

We conclude this section by calculation of $\mathsf{ad}\left(v_0^2 + (v_0 + v_2)^2\right)$ (Schwefel function 1.2 in 2 dimensions):

$$\mathsf{ad}\left(v_0^2 + (v_0 + v_2)^2\right) \overset{\mathbf{R1}}{=} \{v_0 = v_4, v_0 = v_6, v_1 = v_5 + v_7\} \cup \mathsf{ad}\left(v_4^2 + (v_6 + v_2)^2\right)$$
$$\mathsf{ad}\left(v_4^2 + (v_6 + v_2)^2\right) \overset{\mathbf{R4}}{=} \{v_8 = v_6 + v_2, v_7 = v_9, v_3 = v_9\} \cup \mathsf{ad}\left(v_4^2 + v_8^2\right)$$
$$\mathsf{ad}\left(v_4^2 + v_8^2\right) \overset{\mathbf{R3}}{=} \{v_{12} = v_4^2, v_5 = 2v_{11}, v_{11} = v_4 v_{13}\} \cup \mathsf{ad}\left(v_{12} + v_8^2\right)$$
$$\mathsf{ad}\left(v_{12} + v_8^2\right) \overset{\mathbf{R3}}{=} \{v_{16} = v_8^2, v_9 = 2v_{15}, v_{15} = v_8 v_{17}\} \cup \mathsf{ad}\left(v_{12} + v_{16}\right)$$
$$\mathsf{ad}(v_{12} + v_{16}) \overset{\mathbf{R4}}{=} \{v_{18} = v_{12} + v_{16}, v_{13} = v_{19}, v_{17} = v_{19}\} \cup \mathsf{ad}(v_{18})$$
$$\mathsf{ad}(v_{18}) \overset{\mathbf{R5}}{=} \{v_{19} = 1\}$$

## 4 Properties of the decomposition algorithm

Let $H$ be a real term which contains variables $v_0$, $v_2$, …, $v_{2n}$ only.

**Theorem 1 (Correctness).** *Let $h$ be the real function specified by real term $H$. The following statements are true:*

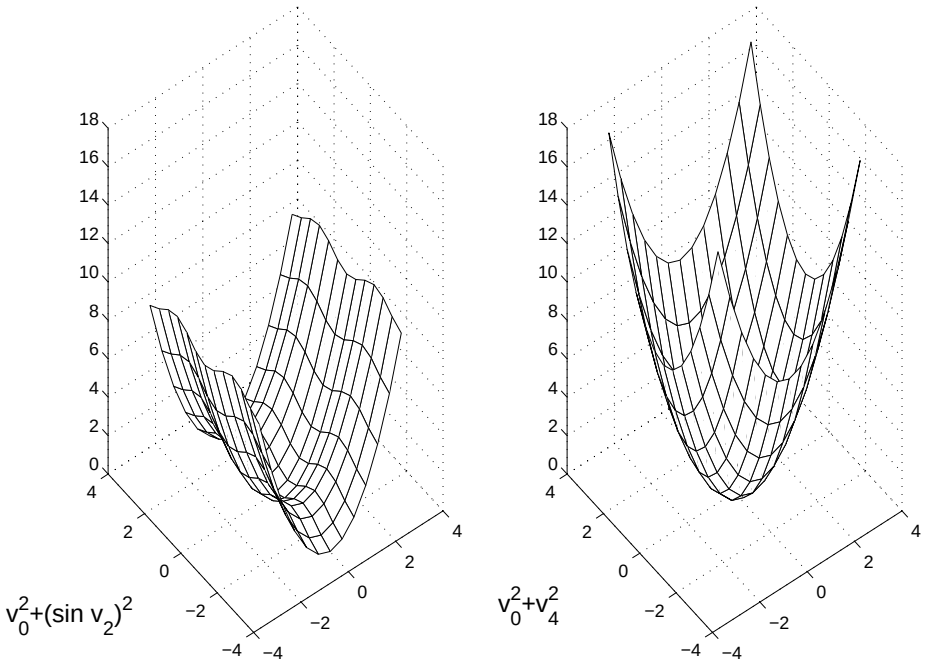


**Fig. 2.** Application of rule 3 to $v_0^2 + (\sin v_2)^2$.

1. *Every solution $p$ to* $\mathsf{ad}(H)$ *satisfies the equation* $p_{2\ell+1} = (D_{2\ell}h)(p)$ *for each variable* $v_{2\ell}$ *from* $H$.
2. *If all the derivatives* $(D_{2\ell}h)(p)$*'s exist for some* $p$*, then the constraints* $\mathsf{ad}(H)$, $\{v_{2\ell} = p_{2\ell} \mid 0 \le \ell \le n\}$, $\{v_{2\ell+1} = (D_{2\ell}h)(p) \mid 0 \le \ell \le n\}$ *have a common solution.*

**Theorem 2 (Number of primitive constraints).** *Let function* $\mathsf{D}_i$ *map real terms specifying real functions to real terms specifying their partial derivatives with respect to argument* $i$.

*Let functions* dec *(see section 3),* $\mathsf{D}_i$ *be defined by the following rules:*

$$\begin{array}{ll}
\mathsf{dec}F\langle v_j\rangle = \{v_j = 0\} & \mathsf{D}_\ell\ c = \mathsf{D}_\ell\ v_j = 0 \\
\mathsf{dec}F\langle c\rangle = \mathsf{dec}F\langle v_i\rangle \cup \{v_i = c\} & \mathsf{D}_\ell\ v_\ell = 1 \\
\mathsf{dec}F\langle f(v_j)\rangle = \mathsf{dec}F\langle v_i\rangle \cup \{v_i = f(v_j)\} & \mathsf{D}_\ell\ f(t) = f'\langle t\rangle \cdot \mathsf{D}_\ell\ t \\
\mathsf{dec}F\langle f(v_j, v_k)\rangle = \mathsf{dec}F\langle v_i\rangle \cup \{v_i = f(v_j, v_k)\} & \mathsf{D}_\ell\ f(t,t') = f'_x\langle t,t'\rangle \cdot \mathsf{D}_\ell\ t + f'_y\langle t,t'\rangle \cdot \mathsf{D}_\ell\ t'
\end{array}$$

*Let symbol* $||\cdot||$ *denote cardinality. Let real term* $H$ *contain* $N$ *basic functions. The following statements are true:*

1. $||\mathsf{dec}(H)|| = N$,
2. $||\mathsf{ad}(H)|| \le \mathrm{const}N$,
3. $||\mathsf{dec}(\mathsf{D}_0(\underbrace{\cos(\cos(\ldots\cos(}_{\cos \text{ occurs } N \text{ times}} v_0)\ldots)))))|| = N(N+3)/2$.

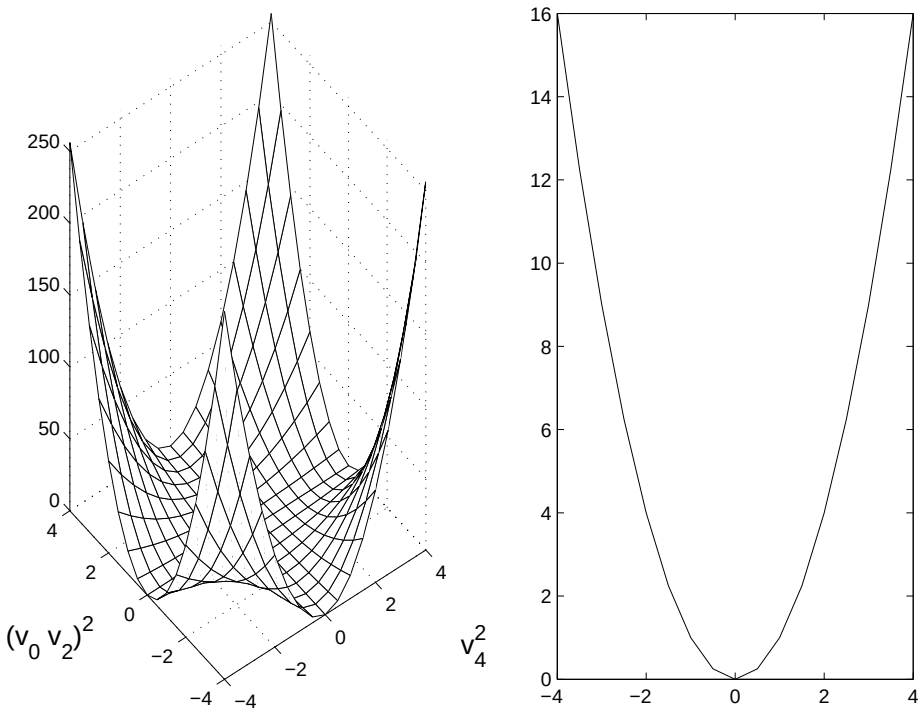


**Fig. 3.** Application of rule 4 to $(v_0 v_2)^2$.

**Theorem 3 (ad and interval constraint propagation).** *Let $C_\ell$ be the set of constraints generated from $\mathsf{dec}(v_{2\ell+1} - \mathsf{D}_\ell(H))$ by renaming variables so that different $C_\ell$'s share variables $v_0$, $v_2$, ..., $v_{2n}$ only.*

*Let intervals $I_0$, ..., $I_{2n+1}$ be calculated by the interval constraint propagation algorithm HC3 [1] for the variables $v_0$, ..., $v_{2n+1}$ and the constraints $\mathsf{ad}(H)$. Let intervals $J_0$, ..., $J_{2n+1}$ be calculated by the same algorithm for the same variables and the constraints $\bigcup_\ell C_\ell$.*

*Then $I_0 \times I_1 \times \ldots \times I_{2n} \times I_{2n+1} \subseteq J_0 \times J_1 \times \ldots \times J_{2n} \times J_{2n+1}$.*

## 5 Experiments

We have applied our decomposition algorithm to minimization of generalized Schwefel function 1.2 and 3.2, Rosenbrock function in different dimensions. Each of these functions is specified by a polynomial of low degree and has exactly one local minimum. Consequently, there are no contraindications for minimization of these functions by interval constraint propagation.

We have run the ECL$^i$PS$^e$ implementation [9] of interval constraint propagation algorithm HC3 [1] on the primitive constraints generated by function ad and by naïve symbolic differentiation $\mathsf{D}_\ell$.

For each objective function, we give a table showing the results of our experiments. Its columns indicate the dimension (column 1), the number of the constraints and the reduction of the initial multidimensional rectangle when HC3 is applied to these constraints for the constraints generated by our transformation technique (columns 2, 3) and by naïve symbolic differentiation (columns 4, 5). The content of columns 3, 5 describes how HC3 changes the initial rectangle. The output rectangles of a diameter comparable with the machine precision are denoted by "exact". The large output rectangles which differ from the initial one at least in one dimension are denoted by "partial". The output rectangle identical to the initial one is denoted by "no reduction".

### 5.1 Schwefel function 1.2

Generalized Schwefel function 1.2 in $n$ dimensions is specified by real term $H = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} v_j \right)^2$. The minimum is achieved at the point $(0, \ldots, 0) \in \mathbf{R}^n$. The standard initial rectangle is $[-5, 10]^n$.

| | AD | | Naïve | |
|---|---|---|---|---|
| $n$ | $\|\|\mathsf{ad}(H)\|\|$ | Reduction | $\|\|\mathsf{dec}(H)\| + \sum_\ell \|\|\mathsf{dec}(\mathsf{D}_\ell(H))\|\|$ | Reduction |
| 1 | 6 | exact | 2 | exact |
| 2 | 17 | exact | 9 | exact |
| 3, 4, 5 | 24, 36, 50 | exact | 23, 46, 80 | partial |
| 6–80 | 66–6800 | exact | 127–180280 | no reduction |

Minimization of Schwefel function 1.2 in the rectangle $[-5, 10]^n$.

The number of primitive constraints generated by AD (column 2) grows quadratically with respect to $n$ (and linearly with respect to $||\mathsf{dec}(H)||$). The number of primitive constraints generated by naïve symbolic differentiation (column 4) grows cubically with respect to $n$. Besides that, HC3 is more effective on the constraints generated by AD (cf. columns 3, 5).

### 5.2 Schwefel function 3.2

Generalized Schwefel function 3.2 in $n$ dimensions is specified by real term $H = \sum_{i=2}^{n} \left( v_1 - v_i^2 \right)^2 + (v_i - 1)^2$. The minimum is achieved at the point $(1, \ldots, 1) \in \mathbf{R}^n$. The standard initial rectangle is $[-10, 10]^n$.

| | AD | | Naïve | |
|---|---|---|---|---|
| $n$ | $\|\|\mathsf{ad}(H)\|\|$ | Reduction | $\|\|\mathsf{dec}(H)\| + \sum_\ell \|\|\mathsf{dec}(\mathsf{D}_\ell(H))\|\|$ | Reduction |
| 2 | 12 | exact | 20 | partial |
| 3–10 | 26–124 | partial | 40–180 | partial |
| 11–80 | 138–1104 | no reduction | 200–1580 | no reduction |

Minimization of Schwefel function 3.2 in the rectangle $[-10, 10]^n$.

The number of primitive constraints grows linearly for AD as for naïve symbolic differentiation (columns 2, 4). However, AD is generates fewer primitive constraints. As far as effectiveness of HC3 is concerned, there is no significant difference between the constraints generated by AD and by naïve symbolic differentiation (cf. columns 3, 5).

### 5.3 Rosenbrock function

Generalized Rosenbrock function in $n$ dimensions is specified by real term $H = \sum_{i=1}^{n-1} 100\left(v_i - v_{i+1}^2\right)^2 + (1 - v_{i+1})^2$. The minimum is achieved at the point $(1, \ldots, 1) \in \mathbf{R}^n$. The standard initial rectangle is $[-2.048, 2.048]^n$. We give also the data for the rectangle $[0.1, 2.048]^n$.

| | AD | | Naïve | |
|---|---|---|---|---|
| $n$ | $\|\|\mathsf{ad}(H)\|\|$ | Reduction | $\|\|\mathsf{dec}(H)\|\| + \sum_\ell \|\|\mathsf{dec}(\mathsf{D}_\ell(H))\|\|$ | Reduction |
| in rectangle $[-2.048, 2.048]^n$ | | | | |
| 2 | 15 | exact | 20 | partial |
| 3–80 | 32–1341 | partial | 40–1580 | partial |
| in rectangle $[0.1, 2.048]^n$ | | | | |
| 2–80 | 15–1341 | exact | 20–1580 | partial |

Minimization of Rosenbrock function in different rectangles.

AD generates a smaller number of constraints. With respect to effectiveness of HC3, there is no big difference between the constraints generated by AD and by naïve symbolic differentiation (cf. columns 3, 5). However, this situation changes, if we start interval constraint propagation in the rectangle $[0.1, 2.048]^n$.

## 6 Conclusion

Automatic differentiation is successfully used in computational mathematics for more than twenty years (as of year 2002). However, applications of AD in interval constraint programming were limited to fast calculation of the first coefficients of Taylor series of nonlinear equations. Up to now, the only means of working with the gradient in optimization problems of ICP was naïve symbolic differentiation in combination with this or that symbolic transformation method. As we have seen in section 4, this approach cannot guarantee an acceptable size of the symbolic expressions for the gradient.

In this paper, we have introduced a technique which allows one to involve the gradient into interval constraint propagation at a low cost in terms of number of constraints. Our experiments indicate that a valuable by-product of our technique is a certain increase in accuracy of interval constraint propagation.

*Acknowledgements* We thank the referees N° 26 and N° 27 for their comments.

## References


1. F. Benhamou. Interval Constraint Logic Programming. In A. Podelski, editor, *Constraint Programming: basics and trends*, volume 910 of *Lecture Notes in Computer Science*, pages 1–21. Springer-Verlag, 1995.
2. R. Dechter. *Principles and Practice of Constraint Programming — CP 2000*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, 2000.
3. ECRC. *ECL$^i$PS$^e$ 3.5: ECRC Common Logic Programming System. User's Guide*, 1995.
4. L. Granvilliers and F. Benhamou. Progress in the solving of a circuit design problem. *Journal of Global Optimization*, 2001.
5. A. Griewank. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM Publications, 2000.
6. K. Marriott and P. J. Stuckey. *Programming with Constraints. An Introduction*. The MIT Press, 1998.
7. L. B. Rall. Automatic differentiation: techniques and applications. volume 120 of *Lecture Notes in Computer Science*. Springer, 1981.
8. P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: a Modelling Language for Global Optimization*. The MIT Press, Cambridge, MA, 1997.
9. T. M. Yakhno, V. Z. Zilberfaine, and E. S. Petrov. Applications of ECL$^i$PS$^e$: Interval Domain Library. In *Proc. Int. Conf. Practical Application of Constraint Technology*, pages 339–357, Westminster Central Hall, London, UK, 1997.