KYBERNETIKA — VOLUME *45* (2009), NUMBER 5, PAGES 865 – 883

AUTOMATIC DIFFERENTIATION AND ITS PROGRAM REALIZATION

JAN HARTMAN, LADISLAV LUKŠAN AND JAN ZÍTKO

Automatic differentiation is an effective method for evaluating derivatives of function, which is defined by a formula or a program. Program for evaluating of value of function is by automatic differentiation modified to program, which also evaluates values of derivatives. Computed values are exact up to computer precision and their evaluation is very quick. In this article, we describe a program realization of automatic differentiation. This implementation is prepared in the system UFO, but its principles can be applied in other systems. We describe, how the operations are stored in the first part of the derivative computation and how the obtained records are effectively used in the second part of the computation.

Keywords: automatic differentiation, modeling languages, systems for optimization

AMS Subject Classification: 26A24, 65K99

1. INTRODUCTION

When computing (not only) complicated technical problems, we face the problem of how to quickly and repeatedly calculate the value of the derivative, the gradient or the values of elements of the Hessian matrix. Automatic differentiation, created according to these requests, is the method which computes the derivatives. The gained values of the derivatives are accurate up to the machine precision and the evaluation of the derivative is very fast. Moreover, it is possible to use the sparsity of the Jacobian or Hessian matrix to compute the derivatives faster and more effectively.

The first papers in this area can be found in 1960's, when the forward mode was described by Wengert (1964). Further improvement were described later, for example, by Kedem (1980) and Rall (1981). The basics of the reverse mode were independently discovered by several mathematicians, for example, by Linnainmaa (1969), Spielpenning (1980), Kim (1984) or others.

The development and usage of automatic differentiation is directly connected with the properties and capability of programming languages, their compilers and computers at all. Thanks to very quick development in this area and thanks to its properties, the popularity of automatic differentiation is increasing. Its theoretical properties are studied, as well as its extensions, methods of implementation, new algorithms and also practical applications, e. g. [1, 2, 3, 5, 11].

Interactive system for functional optimization UFO (Universal Functional Optimization) has been developed in the Institute of Computer Science of Academy of Sciences of the Czech Republic. Large system has been created since 1989. It solves many types of optimization problems and is available for wide sphere of users. Classes of problems which can be solved by UFO system can be found in [10].

Because of popularity of automatic differentiation and its properties, the implementation of techniques of automatic differentiation in the system UFO was designed and created. However, used principles can be easily transformed to other computer systems. The aim of this article is to introduce the implementation of automatic differentiation generally and in details.

The whole implementation with all the details is described in the technical report [8].

Automatic differentiation in FORTRAN77 has been implemented in other systems, e. g.

- ADIFOR, see `http://www.mcs.anl.gov/research/projects/adifor/`,
- TAPENADE, see `http://www-sop.inria.fr/tropics/`,
- REVOLVE, see `http://www.math.tu-dresden.de/wir/project/revolve/`
- or others, see `http://www.autodiff.org/?module=Tools&language=Fortran77`

The presented implementation uses and exploits specific properties of the system UFO, as will be described in section 3.2

The paper is organized as follows. In the first part, we briefly mention the basic properties of automatic differentiation and we describe algorithms for computing of derivatives in the forward and reverse mode. The main principles of the implementation are described in the second part. Finally, we demonstrate advantages and efficiency of automatic differentiation on one of the testing programs from the UFO system.

2. BASIC PROPERTIES OF AUTOMATIC DIFFERENTIATION

In this section, the basic properties of automatic differentiation are briefly introduced. More details can be found e. g. in [5, 7] or [9].

2.1. Code list

Let us assume, that the computer program, which we want to transform, computes the value of the components of the vector function

$$f = [f_1, f_2, \dots, f_m]^T : D \subset \mathbb{R}^n \longrightarrow \mathbb{R}^m.$$

Each component of f is a composition of *basic functions* (e. g. adding, multiplication etc.) and *elementary functions* (i. e. functions of the programming language, e. g. sinus etc.).

The program computes step by step result values of the basic or elementary functions and uses these values for evaluation of forthcoming basic or elementary operations. Let us denote v_i the value of ith elementary or basic function.

Moreover, let us order the values v_i by the following way:

$$\underbrace{v_{1-n}, \ldots, v_0}_{x}, v_1, v_2, \ldots, v_{l-m}, \underbrace{v_{l-m+1}, \ldots, v_l}_{y}, \tag{1}$$

where $x = (x_1, \ldots x_n)$ are input (independent) variables and $y = (y_1, \ldots y_m) = f(x)$ are output (dependent) variables.

We say that $j \prec i$ if and only if the variable value v_i depends *directly* on the variable v_j and we will write

$$v_i = \varphi_i(v_j)_{j \prec i}. \tag{2}$$

Using above defined notation, the computer program can be expressed by the code list in Figure 1.

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

Fig. 1. Code list.

The input values $x_1, \ldots, x_n$ are assigned into the variables $v_{1-n}, \ldots, v_0$ in the first row. Then the components of $f(x)$ are computed and finally the output variables $(y_1, \ldots, y_m) = f(x)$ are assigned in the third row. Let us note that partial derivatives of the basic and elementary functions $\frac{\partial \varphi_i}{\partial v_j}$ must exist and must be known in points where the derivative is to be evaluated.

The principles of automatic differentiation will be demonstrated on the following sample example.

Example 1. Let $M = \{[x_1, x_2] \in \mathbb{R}^2; x_1 \neq 0 \text{ and } x_2 \neq 0\}$ and consider the function $f : M \subseteq \mathbb{R}^2 \to R$

$$f(x_1, x_2) = \frac{\sin x_1}{x_1 x_2} + x_1 x_2. \tag{3}$$

Figure 2 demonstrates the code list for evaluation of $y = f(\pi/4, 1)$.

2.2. The first derivative computation – forward mode

The *forward* mode evaluates directional derivatives of all dependent variables. We assign a new *adjoint* variable $\dot{v}_i$ to every variable v_i,

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j, \tag{4}$$

which is a derivative of (2) where the chain rule has been used. The *adjoint* code list is in Figure 3.

$$
\begin{array}{lcl|c|l}
v_{-1} & = & x_1 & = & \frac{\pi}{4} = 0.7854 \\
v_0 & = & x_2 & = & 1.0000 \\
\hline
v_1 & = & \sin v_{-1} & = & 0.7071 \\
v_2 & = & v_{-1} * v_0 & = & 0.7854 \\
v_3 & = & v_1 / v_2 & = & 0.9003 \\
v_4 & = & v_2 + v_3 & = & 1.6857 \\
\hline
y & = & v_4 & = & 1.6857
\end{array}
$$

Fig. 2. Code list for Example 1.

$$
\begin{array}{lcll}
v_{i-n} & = & x_i & \multirow{2}{*}{$\Big\}\ i = 1, \dots n$} \\
\dot{v}_{i-n} & = & \dot{x}_i & \\
\hline
v_i & = & \varphi_i(v_k)_{k \prec i} & \multirow{2}{*}{$\Big\}\ i = 1, \dots l$} \\
\dot{v}_i & = & \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j & \\
\hline
y_{m-i} & = & v_{l-i} & \multirow{2}{*}{$\Big\}\ i = m-1, \dots 0$} \\
\dot{y}_{m-i} & = & \dot{v}_{l-i} &
\end{array}
$$

Fig. 3. Code list derived by forward mode.

We evaluate the derivative in the point $x = (x_1, x_2, \dots, x_n)$ in the direction $\dot{x}$. If $\dot{x} = e_i$, then the partial derivative with respect to the ith variable is computed.

Example 2. Let us continue in the sample Example 1. Now the gradient of the function f defined by (3) at the point $(\pi/4, 1)$ will be computed.

At first, the derivative with respect to x_1 is computed by the code list in Figure 4. Let us note that the function value $f(\pi/4, 1)$ is computed by this code list too.

To obtain the derivative with respect to x_2 in the same point, it is necessary to execute the code list in Figure 4 again but with the initial values $(\dot{x}_1, \dot{x}_2) = (0, 1)$.

In general, the schema in Figure 3 can be expressed in the form of a matrix multiplication (see e. g. [5])

$$f'(x) = Q_m A_l A_{l-1} \cdots A_2 A_1 P_n^T. \tag{5}$$

The matrix A_i is the identity matrix except the $(i+n)$th row where the element $\frac{\partial \varphi_i}{\partial v_j}$ lies in the $(j+n)$th column for all $j \prec i$.

Example 3. Let us continue in the sample Example 2. In this part of our example, the matrices A_1 and A_2 are presented for calculation of the derivatives of the function (3).

$$
\begin{array}{lcll}
x_1 & = & \frac{\pi}{4} & = \quad 0.7854 \\
\dot{x}_1 & = & 1 & = \quad 1.0000 \\
x_2 & = & 1 & = \quad 1.0000 \\
\dot{x}_2 & = & 0 & = \quad 0.0000 \\
\hline
v_1 & = & \sin x_1 & = \quad 0.7071 \\
\dot{v}_1 & = & \cos x_1 * \dot{x}_1 & = \quad 0.7071 \\
v_2 & = & x_1 * x_2 & = \quad 0.7854 \\
\dot{v}_2 & = & x_1 * \dot{x}_2 + x_2 * \dot{x}_1 & = \quad 1.0000 \\
v_3 & = & v_1 / v_2 & = \quad 0.9003 \\
\dot{v}_3 & = & (\dot{v}_1 - v_3 * \dot{v}_2)/v_2 & = \quad -0.2460 \\
v_4 & = & v_2 + v_3 & = \quad 1.6857 \\
\dot{v}_4 & = & \dot{v}_2 + \dot{v}_3 & = \quad 0.7540 \\
\hline
y & = & v_4 & = \quad 1.6857 \\
\dot{y} & = & \dot{v}_4 & = \quad 0.7540
\end{array}
$$

Fig. 4. Code list derived by forward mode for Example 2.

It holds that $n = 2$, $m = 1$ and $l = 4$. Matrices A_1 and A_2 have the form

$$
A_1 = \left(\begin{array}{cc|c|ccc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
\hline
\frac{\partial \varphi_1}{\partial v_{-1}} & \frac{\partial \varphi_1}{\partial v_0} & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
= \left(\begin{array}{cc|c|ccc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
\hline
\cos v_{-1} & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
$$

$$
= \left(\begin{array}{cc|c|ccc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
\hline
0.7071 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right) \in \mathbb{R}^{6\times 6}
$$

and

$$
A_2 = \left(\begin{array}{ccc|c|cc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
\hline
\frac{\partial \varphi_2}{\partial v_{-1}} & \frac{\partial \varphi_2}{\partial v_0} & \frac{\partial \varphi_2}{\partial v_1} & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
= \left(\begin{array}{ccc|c|cc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
\hline
v_0 & v_{-1} & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right)
$$

$$
= \left(\begin{array}{ccc|c|cc}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 \\
\hline
1 & 0.7854 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{array}\right) \in \mathbb{R}^{6\times 6}.
$$

On this sample example, it is easy to see that the matrix multiplication (5) corresponds to the Figure 4.

2.3. The first derivative computation – reverse mode

By the reverse mode, we evaluate a linear combination of $\bigtriangledown f_i$. We want to compute the derivatives in the form (see [5])

$$\bar{x} = \bar{y} \cdot f'(x). \tag{6}$$

where the parameter $\bar{y} = (\bar{y}_1, \bar{y}_2, \dots, \bar{y}_m)$ is an input row weight vector. Let us define new variable

$$\bar{v}_i = \frac{\partial(\bar{y} \cdot f(x))}{\partial v_i}, \qquad i = 1-n, \dots, l. \tag{7}$$

To understand better the definition (7), let us formulate the following theorem.

Theorem 2.1. The following equalities hold:

$$\bar{v}_j = \bar{y}_{j-l+m} \qquad \text{for} \qquad j = l-m+1, \dots, l, \tag{8}$$

$$\bar{v}_j = \sum_{i\ ;\ i \succ j} \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j} \qquad \text{for} \qquad j = 1-n, \dots, l-m. \tag{9}$$

P r o o f . The equalities (8) follow immediately from the relation (1).
For the simplicity, let us show the equality (9) in the special case for $m = 1$ and three variables v_{i_1}, v_{i_2} and v_{i_3} which depend directly on v_j, i. e.

$$j \prec i_1, \quad j \prec i_2 \quad \text{and} \quad j \prec i_3. \tag{10}$$

The variable $v_l \in \mathbb{R}^1$ can be express as a function of v_j and some other variables $v_{p_1}, \dots, v_{p_r}$, which do not depend on v_j. Let us express this fact by the relation

$$v_l = v_l(v_j, v_{p_1}, \dots, v_{p_r}).$$

In view of (10), it holds

$$v_l = v_l(v_{i_1}, v_{i_2}, v_{i_3}, v_{p_1}, \dots, v_{p_r})$$

and

$$\begin{aligned} v_{i_1} &= v_{i_1}(v_j, \dots), \\ v_{i_2} &= v_{i_2}(v_j, \dots), \\ v_{i_3} &= v_{i_3}(v_j, \dots) \end{aligned}$$

where dots $(\dots)$ are used instead of variables not depending on v_j. Hence we can write

$$v_l = v_l(v_{i_1}(v_j, \dots), v_{i_2}(v_j, \dots), v_{i_3}(v_j, \dots), v_{p_1}, \dots, v_{p_r}).$$

Application of the chain rule yields the formulas

$$\begin{aligned}
\frac{\partial v_l}{\partial v_j} &= \frac{\partial v_l}{\partial v_{i_1}} \cdot \frac{\partial v_{i_1}}{\partial v_j} + \frac{\partial v_l}{\partial v_{i_2}} \cdot \frac{\partial v_{i_2}}{\partial v_j} + \frac{\partial v_l}{\partial v_{i_3}} \cdot \frac{\partial v_{i_3}}{\partial v_j} + \\
&\quad \frac{\partial v_l}{\partial v_{p_1}} \cdot \frac{\partial v_{p_1}}{\partial v_j} + \cdots + \frac{\partial v_l}{\partial v_{p_r}} \cdot \frac{\partial v_{p_r}}{\partial v_j} \\
&= \frac{\partial v_l}{\partial v_{i_1}} \cdot \frac{\partial v_{i_1}}{\partial v_j} + \frac{\partial v_l}{\partial v_{i_2}} \cdot \frac{\partial v_{i_2}}{\partial v_j} + \frac{\partial v_l}{\partial v_{i_3}} \cdot \frac{\partial v_{i_3}}{\partial v_j}
\end{aligned}$$

due to

$$\frac{\partial v_{p_1}}{\partial v_j} = \cdots = \frac{\partial v_{p_r}}{\partial v_j} = 0$$

because $v_{p_1}, \ldots, v_{p_r}$ do not depend on v_j.

Now it is easy to see that in general case

$$\frac{\partial v_l}{\partial v_j} = \sum_{i\ ;\ i \succ j} \frac{\partial v_l}{\partial v_i} \cdot \frac{\partial v_i}{\partial v_j},$$

i. e.

$$\bar{v}_j = \sum_{i\ ;\ i \succ j} \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}.$$

$\square$

By reformulating of the sum (9), the code list described in Figure 5 is obtained. This code list can be also derived from the relations (5) and (6).

```
$v_{i-n} = x_i$                         $i = 1, \ldots, n$
$v_i = \varphi_i(v_k)_{k \prec i}$       $i = 1, \ldots, l$
$y_{m-i} = v_{l-i}$                     $i = m-1, \ldots, 0$
$\bar{v}_{l-i} = \bar{y}_{m-i}$         $i = 0, \ldots, m-1$
$\bar{v}_i = 0$                         $i = l-m, \ldots, 1-n$
for $i = l, \ldots, 1$ do
    for $j \prec i$ do
        $\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$
    end for
end for
$\bar{x}_i = \bar{v}_{i-n}$             $i = n, \ldots, 1$
```

Fig. 5. Code list derived by reverse mode.

After execution of the code list in Figure 5, the computed derivatives are stored in variables (see (9))

$$\bar{x} = \bar{y} \cdot f'(x).$$

Example 4. Let us continue in the Example 3. We want to compute the gradient of the function f in the point $(\pi/4, 1)$.

The code list derived by the reverse mode for Example 4 is in Figure 6. Not only the value of gradient $\nabla f(\pi/4,1) = (\bar{v}_{-1}, \bar{v}_0) = (0.7540, -0.1149)$ was computed but also the function value $y = f(\pi/4, 1) = v_4 = 1.6857$ was evaluated.

v_{-1}	$=$	x_1	$=$	$\frac{\pi}{4} = 0.7854$
v_0	$=$	x_2	$=$	1.0000
v_1	$=$	$\sin v_{-1}$	$=$	0.7071
v_2	$=$	$v_{-1} * v_0$	$=$	0.7854
v_3	$=$	v_1 / v_2	$=$	0.9003
v_4	$=$	$v_2 + v_3$	$=$	1.6857
y	$=$	v_4	$=$	1.6857
$\bar{v}_4$	$=$	1.0000		
$\bar{v}_i$	$=$	0		$i = 3, \ldots, -1$
$\bar{v}_3$	$=$	$\bar{v}_3 + \bar{v}_4$	$=$	1.0000
$\bar{v}_2$	$=$	$\bar{v}_2 + \bar{v}_4$	$=$	1.0000
$\bar{v}_1$	$=$	$\bar{v}_1 + \bar{v}_3 / v_2$	$=$	1.2732
$\bar{v}_2$	$=$	$\bar{v}_2 + \bar{v}_3 * (-v_1/v_2^2)$	$=$	
	$=$	$\bar{v}_2 - \bar{v}_3 * v_3 / v_2$	$=$	-0.1463
$\bar{v}_0$	$=$	$\bar{v}_0 + \bar{v}_2 * v_{-1}$	$=$	-0.1149
$\bar{v}_{-1}$	$=$	$\bar{v}_{-1} + \bar{v}_2 * v_0$	$=$	-0.1463
$\bar{v}_{-1}$	$=$	$\bar{v}_{-1} + \bar{v}_1 * \cos v_{-1}$	$=$	0.7540
$\frac{\partial f}{\partial x_1}$	$=$	$\bar{v}_{-1}$	$=$	0.7540
$\frac{\partial f}{\partial x_2}$	$=$	$\bar{v}_0$	$=$	-0.1149

Fig. 6. Code list derived by reverse mode for Example 4.

2.4. The second (and higher) derivative computation

The program for evaluating of the second (or higher) derivatives can be derived by a combination of the forward and reverse mode, see e. g. [5]. Firstly, we use the reverse mode to obtain the gradient of the component of the function f. Then we apply the forward mode and we get a schema for the second derivative evaluation in the form

$$\bar{y} \cdot f''(x) \cdot \dot{x} + \dot{\bar{y}} \cdot f'(x), \tag{11}$$

where $\dot{\bar{y}}$ is an input parameter, similar to $\dot{x}$ or $\bar{y}$. Let us denote the expression (11) as $\dot{\bar{x}}$. If we want to compute only the second derivative of the form $\bar{y} \cdot f''(x) \cdot \dot{x}$, it is necessary to assign $\dot{\bar{y}} = 0$. The expression $\bar{y} \cdot f''(x) \cdot \dot{x}$ can be interpreted as

$$\bar{y} \cdot f''(x) \cdot \dot{x} = \frac{\partial}{\partial \alpha} \, \bar{y} \cdot f'(x + \alpha \dot{x}) \bigg|_{\alpha=0} \in \mathbb{R}^n.$$

The code list can be obtained by application of the forward mode to a code list obtained by the reverse mode. The expression

$$\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$$

is transformed to

$$\dot{\bar{v}}_j = \dot{\bar{v}}_j + \dot{\bar{v}}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} + \bar{v}_i \cdot \frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i}, \tag{12}$$

where

$$\frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i} = \sum_{l \prec i} \frac{\partial^2 \varphi_i}{\partial v_j \partial v_l}(v_k)_{k \prec i} \cdot \dot{v}_l.$$

The code list for the evaluation of the second derivatives is in Figure 6. The values of the second derivatives $\bar{y} \cdot f''(x) \cdot \dot{x}$ are the components of the vector

$$\dot{\bar{x}} = (\dot{\bar{x}}_1, \ldots, \dot{\bar{x}}_m).$$

```
$v_{i-n} = x_i$
$\dot{v}_{i-n} = \dot{x}_i$                                            } $i = 1, \ldots, n$
----------------------------------------------------------------------------------
$v_i = \varphi_i(v_k)_{k \prec i}$
$\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j$      } $i = 1, \ldots l$
----------------------------------------------------------------------------------
$y_{m-i} = v_{l-i}$
$\dot{y}_{m-i} = \dot{v}_{l-i}$                                        } $i = m-1, \ldots 0$
==================================================================================
$\bar{v}_{l-i} = \bar{y}_{m-i}$
$\dot{\bar{v}}_{l-i} = 0$                                              } $i = 0, \ldots, m-1$
----------------------------------------------------------------------------------
$\bar{v}_i = 0$
$\dot{\bar{v}}_i = 0$                                                  } $i = 1-n, \ldots, l-m$
----------------------------------------------------------------------------------
for $j = l, \ldots, 1$ do
    for $j \prec i$
        $\bar{v}_j = \bar{v}_j + \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i}$
        $\dot{\bar{v}}_j = \dot{\bar{v}}_j + \dot{\bar{v}}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} + \bar{v}_i \cdot \frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i}$
    end for
end for
----------------------------------------------------------------------------------
$\bar{x}_i = \bar{v}_{i-n}$
$\dot{\bar{x}}_i = \dot{\bar{v}}_{i-n}$                                } $i = n, \ldots, 1$
```

Fig. 7. Code list derived by reverse and forward mode for the second derivatives.

3. AUTOMATIC DIFFERENTIATION IN THE UFO SYSTEM

In the previous section, we briefly reviewed basic principles of automatic differentiation. In this part, we describe how to implement these techniques in a computer system in general and also in particular in the UFO system. The implementation will be demonstrated on a simple example. More details about the UFO system can be found in the technical report [10].

3.1. A brief description of the UFO system

The UFO system can be used for solution of optimization problems and preparation of optimization algorithms. The typical task is to find a local minimum of a function F.

Solution of the optimization problem consists of four steps:

1. Specification of the optimization problem and selection of the method. It is done by the UFO control language and it is saved into a file.
2. This file is transformed by the UFO preprocessor. According to its commands, the computer program in FORTRAN 77 is automatically generated which solves the original optimization problem.
3. This program is compiled and linked with library subroutines.
4. The solution of the optimization problem is obtained after running the program.

Let us demonstrate the above formulated four stages on the following example. We want to find a local minimum of the Rosenbrock function

$$F(x) = 100(x_1^2 - x_2)^2 + (x_1 - 1)^2.$$

Let $x = (-1.2, 1.0)$ be the initial approximation. According to the first stage, the input file for the UFO system is prepared:

```
$SET(INPUT)
  X(1)=-1.2D0; X(2)=1.0D0
$ENDSET
$SET(FMODELF)
  FF = 1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
$NF=2
$NOUT=1
$BATCH
$STANDARD
```

This file sets the initial approximation (`INPUT`) for the iterative method, object function (`FMODELF`), number of its independent variables (`NF`) and the form of the output (`NOUT`). The successful processing of this file yields the results:

```
FF=   .2333078060D-15
X=    .9999999847D+00   .9999999694D+00
TIME= 0:00:00.66
```

The local minimum was found at the point `X` and the value $F(\texttt{X})$ is stored in the variable `FF`.

The formula for the object function is defined by the variable `FMODELF`. It is possible to define, moreover, the variables `GMODELF` and `HMODELF` with formulae for Jacobi and Hessian matrix respectively. In absence of these variables, the derivatives are evaluated numerically by the divided differences. The instruction `$BATCH` suppresses the dialogue mode and the statement `$STANDARD` creates the program that solves the optimization problem.

Let us remark that the UFO preprocessor is based on the interpreter BEL (Batch Editor Language), which was developed as a part of the UFO system. The BEL interpreter was modified during implementation of the automatic differentiation.

3.2. Automatic differentiation in the UFO system

Since the beginning of automatic differentiation, its implementation has been prepared in several systems. However, the advantages and features of our implementation in the UFO system are following:

- automatic differentiation is the part of the UFO system and does not need any additional programs or parameters,
- the UFO system generates program for solving of optimization problem and automatic differentiation is only a part of the system,
- automatic differentiation is a part of the interpreter BEL and it has and exploits knowledge about optimization problem, e. g. the sparsity of matrices,
- the UFO system works with macro-variables that can be used e. g. as parametres,
- compared to e. g. LANCELOT system for functional optimization (see [12]), the description of the objective function in the UFO system is much more simple.

By the automatic differentiation, it is possible to evaluate the first or the second derivatives of the functions defined by the variables `FMODELF`, `FMODELA` or `FMODELC`. Let us introduce three new variables `$IADF`, `$IADA` and `$IADC`. They express whether and what derivatives will be evaluated by the automatic differentiation.

- `$IADF`=0 (default value)
  The derivatives of the function `FF` defined by the variable `FMODELF` are not evaluated by the automatic differentiation.
- `$IADF`=1
  The first derivatives of the function `FF` defined by the variable `FMODELF` are evaluated by the reverse mode of the automatic differentiation. The new variable `FGMODELF` is created and it contains the formulae for the evaluation of the function `FF` and also its gradient `GF`. The variable `FMODELF` is deleted.
- `$IADF`=2
  The second derivatives of the function `FF` defined by the variable `FMODELF` are evaluated by the combination of the reverse and forward modes. The new variable `FGMODELF` is created and it contains the formulae for the evaluation of

the function FF and also its gradient GF by the reverse mode. The new variable HMODELF is also created and it contains the formulae for the evaluation of the function FF, its gradient GF and Hessian matrix by the reverse and forward modes. Then, variable FMODELF is deleted.

The variables $IADA and $IADC are used in the same way, but they change the functions FA and FC defined by the variables FMODELA and FMODELC. New variables FGMODELA or HMODELA and FGMODELC or HMODELC are created.

There are a few limitations by application of the variables FMODELF, FMODELA and FMODELC which can be found in [8]. These limitations are not restrictive and it is easy to meet them.

3.3. The implementation of the automatic differentiation in the UFO system

3.3.1. The first derivative

When the first derivatives are evaluated, the reverse mode of the automatic differentiation is used. The independent variables and all basic and elementary operations are subsequently stored in arrays, including their values, arguments and the type of the operation.

If $IADF=1 or 2, the following arrays are used :

- REAL*8 V($NADARR) – the array where the values $v_i = \varphi_i(v_j)_{j \prec i}$ are stored; see the relation (2)
- REAL*8 VBAR($NADARR) – the array where the values $\bar{v}_i = \sum_{j \succ i} \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}$ are stored; see the relation (9)
- INTEGER OPCODE($NADARR) – the array stores the type of the performed operation, e. g. addition corresponds to number 10, sin corresponds to number 60 etc.
- INTEGER ARG1($NADARR) – the array stores the pointers (array indexes) to the first argument of the basic or elementary operation
- INTEGER ARG2($NADARR) – the array stores the pointers (array indexes) to the second argument of the basic or elementary operation

For $IADF=2, there are two more arrays:

- REAL*8 VDOT($NADARR) – the array where the values $\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j$ are stored; see the relation (4)
- REAL*8 VBARDT($NADARR) – is the array which stores the values $\dot{\bar{v}}_i$, see the relation (12)

Each basic or elementary operation $\varphi_i(v_j)_{j \prec i}$ is replaced by a subprogram which performs not only the operation $\varphi_i(v_j)_{j \prec i}$, but records its call to the arrays V, VBAR, OPCODE, ARG1, ARG2, and possibly to VDOT, VBARDT. For example, the operation multiplication or sin are replaced by the subprograms `BMULTG`, resp. `SING` which are listed below.

```
!--- transformed operation multiplication ---
INTEGER FUNCTION BMULTG(IARG1, IARG2)  !input parameters: indexes (order)
                                       !to the arrays, which values are
                                       !to be multiplied
                                       !output variable: index (order)
                                       !to the arrays for this operation
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  INTEGER IARG1, IARG2
  V(INDARR)=V(IARG1)*V(IARG2)          !evaluation of the operation multiplication
  VBAR(INDARR)=0.0D0                   !the variable v_i with the bar is set to 0
                                       !(we will add to it some values)
  OPCODE(INDARR)=30                    !store the code of this operation
                                       !(30 = multiplication)
  ARG1(INDARR)=IARG1                   !store the pointer (index) to the 1st argument
  ARG2(INDARR)=IARG2                   !store the pointer (index) to the 2nd argument
  BMULTG=INDARR                        !the index (order) of this operation
  INDARR=INDARR+1                      !shift the pointer to the arrays - preparation
                                       !for the next basic or elementary operation
END

!--- transformed operation sin ---
INTEGER FUNCTION SING(IARG1)
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  INTEGER IARG1
  V(INDARR)=SIN(V(IARG1))
  VBAR(INDARR)=0.0D0
  OPCODE(INDARR)=60                     !store the code of this operation
                                        !(60 = sin)
  ARG1(INDARR)=IARG1                    !sin has only one argument
  SING=INDARR
  INDARR=INDARR+1
END
```

All independent variables and basic and elementary operations are represented by the order of the operation. This order is the same as the index in the arrays, where information about it is stored.

The subroutine `RVRSWP` is called in the last stage of the derivative evaluation. It goes through the arrays `V`, `VBAR`, `OPCODE`, `ARG1`, `ARG2` backwards and evaluates the derivatives (9), i. e.

$$\bar{v}_i = \sum_{j \succ i} \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}.$$

It can be reformulated to

$$\bar{v}_i = \bar{v}_i + \bar{v}_j \cdot \frac{\partial \varphi_j}{\partial v_i}(v_k)_{k \prec j} \quad \text{pro} \quad i \prec j \qquad j = l, \ldots, 1.$$

The listing of the subroutine RVRSWP follows:

```
SUBROUTINE RVRSWP()
  COMMON /AD_F1/ V, VBAR, OPCODE, ARG1, ARG2, INDARR
    REAL*8 V($NADARR), VBAR($NADARR)
    INTEGER OPCODE($NADARR), ARG1($NADARR), ARG2($NADARR)
    INTEGER INDARR
  REAL*8 DERIV
  INTEGER I
  DO 999, I=INDARR-1, 1, -1             !the loop through operations backwards
                                        !i.\,e. through arrays backwards
   .
   .
    IF(OPCODE(I).EQ.30) THEN            !operation multiplication
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*V(ARG2(I))   !addition to the
                                                       !variables v_. with bar
      VBAR(ARG2(I))=VBAR(ARG2(I))+VBAR(I)*V(ARG1(I))   !addition to the
                                                       !variables v_. with bar
   .
   .
    ELSEIF(OPCODE(I).EQ.60) THEN        !operation sin
      DERIV=COS(V(ARG1(I)))                            !temporary variable
      VBAR(ARG1(I))=VBAR(ARG1(I))+VBAR(I)*DERIV        !addition to the
                                                       !variables v_. with bar
   .
   .
    ELSEIF(OPCODE(I).EQ.2) THEN         !operation - the definition
                                        !of the independent variable
      CONTINUE                          !do nothing
   .
   .
    ENDIF
999 CONTINUE
    END
```

The values of the derivatives are stored in the corresponding elements of the array VBAR, after processing of RVRSWP.

Example 5. We will demonstrate, how the values are stored in the arrays V, VBAR, OPCODE, ARG1 a ARG2.

Let us define the function $f(x_1, x_2) = x_1x_2+1$. The program computes this value by the statement

```
X(1) * X(2) + 1.
```

Automatic differentiation transforms this statement to

```
BPLUSG(BMULTG(IAD_X(1),IAD_X(2)),MKCNST(DBLE(1))),
```

where IAD_X(1) and IAD_X(2) are pointers to arrays (array indexes) where independent values X(1) a X(2) are stored.

The main principles of data storing in the arrays V, VBAR, OPCODE, ARG1 and ARG2 are shown in Figure 8. The subroutine MKINDP and MKCNST store the value of the independent variable $x(i)$ and the value of a constant, respectively, into above declared arrays.

the order of the performed operation	1	2	3	4	5	...
performed operation	definition of independent variable $X(1)$	definition of independent variable $X(2)$	$x_1 x_2$	definition of constant 1	$x_1 x_2 + 1$	...
subroutine which created the record in the arrays	MKINDP (X(1))	MKINDP (X(2))	BMULTG (.,.)	MKCNST (DBLE(1))	BPLUSG (.,.)	...
array index	1	2	3	4	5	...
array `V`	value x_1	value x_2	value $x_1 \cdot x_2$	value 1	value $x_1 \cdot x_2 + 1$	...
array `VBAR`	0	0	0	0	0	...
array `OPCODE`	2	2	30	1	10	...
array `ARG1`	0	0	1	0	3	...
array `ARG2`	0	0	2	0	4	...

Fig. 8. The main principles of the data storing in the arrays.

3.3.2. The second derivatives

Firstly, the reverse mode is applied to get the program for the first derivatives. Secondly, the forward mode is applied to this program to get the second derivatives. The subroutines `BMULTH` and `SINH` not only store the values, but also the first derivatives (directional derivatives) are evaluated. The subroutines for the second derivatives are listed in [8].

4. EXAMPLE

The following example demonstrates advantages of the automatic differentiation. Let us denote $x \in \mathbb{R}^N$ and define the function $F : \mathbb{R}^N \to \mathbb{R}$

$$F(x) = \sum_{i=1}^{N} (N + i - P_i)^2 , \tag{13}$$

where

$$P_i = \sum_{j=1}^{N} \left(5\,(1 + \mathrm{mod}(i,5) + \mathrm{mod}(j,5)) \sin(x_j) + \frac{i+j}{10} \cos(x_j) \right),$$

and $\mathrm{mod}(a, b)$ is a remainder for the division of a by b. We want to calculate a local minimum of the function F. Let $x_0 = (1, 1/2, \dots, 1/N)$ be an initial approximation. The input file for the UFO system is in Figure 9. Let us describe this file briefly.

The derivatives of the function defined by the variable `FMODELA` are to be evaluated by the automatic differentiation. Thus the command `$IADA=1` on the line number 21 is stated. The number of the independent variables `X(*)` is defined by the variable

```
 ! declaration of temporary variables
INTEGER IAD_W($$NF+1)                                              !01
REAL*8 W($$NF+1)                                                   !02
 ! initial settings:
$SET(INPUT)                                                        !03
      DO 80 I=1, $$NF                                              !04
      X(I)=1.0D0/I                                                 !05
   80 CONTINUE                                                     !06
$ENDSET                                                            !07
 ! elements of a sum of the objective function:
$SET(FMODELA)                                                      !08
      W(1)=0.0D0                                                   !09
      DO 81 I=1, $$NF                                              !10
        A=5.0D 0*(1.0D 0+MOD(I,5)+MOD(KA,5))                       !11
        B=DBLE(I+KA)/1.0D1                                         !12
        W(I+1)=W(I)+A*SIN(X(I))+B*COS(X(I))                        !13
   81 CONTINUE                                                     !14
      FA=(DBLE($$NF+KA)-W($$NF+1))**2                              !15
$ENDSET                                                            !16
 ! type of the optimization problem:
$MODEL='AF'                                                        !17
 ! number of independent variables:
$NF=50                                                             !18
$NA=50                                                             !19
$NOUT=1                                                            !20
 ! automatic differentiation for FMODELFA:
$IADA=1                                                            !21
$BATCH                                                             !22
$STANDARD                                                          !23
```

Fig. 9. Input file for the UFO system for the example with automatic differentiation.

`$NF` on the line 18, the type of the optimization problem is defined by the variables `MODEL` and `$NA` on lines number 17 and 19. See also [10]. The evaluation of

$$(N + i - P_i)^2 \tag{14}$$

(see the relation 13) is defined by the term `FA` in the variable `FMODELA` on lines 8 – 16. The initial approximation x_0 is in the variable `INPUT` on lines 3 – 7.

The input file (Figure 9) is transformed by the UFO system to a program, which computes a local minimum of a function F. The first step of the input file processing is the deletion of the variable `FMODELA` and the creation of the variable `FGMODELA`, because the value `$IADA`= 1. The value of the variable `FGMODELA` is shown in Figure 10.

Let us describe this transformed variable `FGMODELF` briefly. The variables `X(.)` are denoted as the independent variables on lines 2 – 4. The statement on the line 13 in Figure 9 is transformed to lines 9 and 10 in Figure 10. Moreover, the statement on the line 15 in Figure 9 is transformed to lines 12 and 13 in Figure 10.

The parameters of the subroutines `BPLUSG`, `BMULTG` etc. are array indexes and are created as a composition of the string `IAD_` and the original variable name, e. g. `IAD_W`. The computed value of `FA` is assigned on line 14. The statement on line 15

```
   INDARR=1                                                           !01
    ! denoting of the independent variables:
   DO 85 IADCOUNT=1,50                                                !02
   IAD_X(IADCOUNT)=MKINDP(X(IADCOUNT))                                !03
85 CONTINUE                                                           !04
    ! transformed evaluation of FA:
   IAD_W(1)=MKCNST(DBLE(0.0D0))                                       !05
   DO 81 I=1, 50                                                      !06
   A=5.0D 0*(1.0D 0+MOD(I,5)+MOD(KA,5))                               !07
   B=DBLE(I+KA)/1.0D1                                                 !08
   IAD_W(I+1)=BPLUSG(BPLUSG(IAD_W(I),BMULTG(MKCNST(DBLE(A)),SING(IAD_  !09
  &    X(I)))),BMULTG(MKCNST(DBLE(B)),COSG(IAD_X(I))))                !10
81 CONTINUE                                                           !11
   IAD_FA=BEXPG(BMINUG(MKCNST(DBLE(DBLE(50+KA))),IAD_W(50+1)),MKCNST(  !12
  &    DBLE(2)))                                                      !13
    ! computed value of FA:
   FA=V(IAD_FA)                                                       !14
    ! weight assigning:
   VBAR(IAD_FA)=1.0D0                                                 !15
    ! go through the arrays backwards and compute the derivatives:
   CALL RVRSWP()                                                      !16
    ! assigning of computed derivatives:
   DO 86 IADCOUNT=1,50                                                !17
   GA(IADCOUNT)=VBAR(IAD_X(IADCOUNT))                                 !18
86 CONTINUE                                                           !19
```

Fig. 10. The value of the variable FGMODELF for the example with automatic differentiation.

corresponds to the fourth line in Figure 5, i. e.

$$\bar{v}_l = 1.$$

The subroutine RVRSWP() goes backwards through the arrays. After its execution, the computed values of the derivatives are written into variables GA on the lines 17 – 19.

There are no changes on the lines 10 – 12 in Figure 9, i. e. the lines 6 – 8 in Figure 10, because they do not depend on the independent variables X(.). The initial setting of the array index counter is on the line 1 in Figure 10.

Table. Comparison of evaluation time for a local minimum search.

Number of indep. variables	automatic differentiation	divided differences
$N = 10$	0.11 s	0.16 s
$N = 20$	0.49 s	0.94 s
$N = 50$	1.37 s	6.97 s
$N = 100$	3.36 s	44.93 s

The evaluation times for a local minimum search are compared in the Table. In the second and third column, there are times for the case that automatic differentiation and divided differences are used respectively, for various N. The table shows that automatic differentiation yields faster convergence.

Moreover, it is clear that the numerical values computed by automatic differentiation are more accurate than the values gained by divided differences.

ACKNOWLEDGEMENT

The work is a part of the research project MSM0021620839 financed by the Ministry of Education, Youth and Sport of the Czech Republic. The work was also supported by the Grant Agency of the Academy of Sciences of the Czech Republic, project IAA1030405, and the institutional research plan AV0Z10300504.
The authors gratefully thank the anonymous referee for very valuable remarks leading to improving this paper.

(Received December 18, 2007.)

REFERENCES

[1] Automatic Differentiation of Algorithms: Theory, Implementation, and Application (A. Griewank and G. F. Corliss, eds.). SIAM, Philadelphia 1992.

[2] Automatic Differentiation: Applications, Theory, and Implementations (H. M. Bucker, G. F. Corliss, P. D. Hovland, U. Naumann, and B. Norris, eds.). Springer–Verlag, Berlin 2005.

[3] Computational Differentiation – Techniques, Applications, and Tools (M. Berz, C. H. Bischof, G. F. Corliss, and A. Griewank, eds.). SIAM, Philadelphia 1996.

[4] R. Griesse and A. Walther: Evaluating gradients in optimal control – Continuous adjoints versus automatic differentiation. J. Optim. Theory Appl. *122* (2004), 1, 63–86.

[5] A. Griewank: Evaluation Derivatives: Principles and Techniques of Algorithmic Differentiation. SIAM, Philadelphia 2000.

[6] A. Griewank and A. Walther: Introduction to automatic differentiation. PAMM *2* (2003), 45–49.

[7] J. Hartman: Realizace metod pro automatické derivování (Implementation of Methods for Automatic Differentiation). Diploma Thesis. Faculty of Mathematics and Physics, Charles University, Prague 2001.

[8] J. Hartman and L. Lukšan: Automatické derivování v systému UFO (Automatic Differentiation in System UFO). Technical Report V-1002. ICS AS CR, Prague 2007.

[9] J. Hartman and J. Zítko: Principy automatického derivování (Principles of Automatic Differentiation). Technical Report, Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Prague 2006.

[10] L. Lukšan, M. Tůma, J. Hartman, J. Vlček, N. Ramešová, M. Šiška, and C. Matonoha: UFO 2006 – Interactive System for Universal Functional Optimization. Technical Report V-977. ICS AS CR, Prague 2006.

[11] A. Verma: Structured Automatic Differentiation. Ph.D. Thesis, Cornell University, 1988.

[12] A. Walther, A. Griewank, and O. Vogel: ADOL-C: Automatic differentiation using operator overloading in C++. PAMM *2* (2003), 41–44.

Jan Hartman, Charles University, Faculty of Mathematics and Physics, Department of Numerical Mathematics, Sokolovská 83, 186 75 Praha 8. Czech Republic.
e-mail: jan.hartman@email.cz

Ladislav Lukšan, Institute of Computer Science Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Praha 8. Czech Republic.
e-mail: luksan@cs.cas.cz

Jan Zítko, Charles University, Faculty of Mathematics and Physics, Department of Numerical Mathematics, Sokolovská 83, 186 75 Praha 8. Czech Republic.
e-mail: zitko@karlin.mff.cuni.cz