Institute of Computer Science
Academy of Sciences of the Czech Republic

Automatic differentiation and its program realization

J.Hartman, L.Lukšan, J.Zítko

Technical report No. 992

December 2007

Pod Vodárenskou věží 2, 182 07 Prague 8 phone: +420 2 688 42 44, fax: +420 2 858 57 89,
e-mail:e-mail:ics@cs.cas.cz

Institute of Computer Science
Academy of Sciences of the Czech Republic

Automatic differentiation and its program realization

J.Hartman, L.Lukšan, J.Zítko [1]

Technical report No. 992

December 2007

Abstrakt:

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 report, 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 this records are effectively used in the second part of the computation.

Keywords:
Automatic Differentiation, Modeling Languages, Systems for Optimization.

[1]This work is a part of the research project MSM0021620839 supported by MSMT. It was also supported by the Grant Agency of the Czech Academy of Sciences, project code IAA1030405, and the institutional research plan No. AV0Z10300504. L.Lukšan is also from Technical University of Liberec, Hálkova 6, 461 17 Liberec.

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 but also practical applications, e.g. [3], [4], [1], [2], [9].

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 [8].

We have used the system UFO for implementation of techniques of automatic differentiation. 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.

Because of popularity and properties of automatic differentiation, the system UFO has been extended. We described it in the technical report [6].

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 [3], [5] or [7].

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

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

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:

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

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

$$v_i = \varphi_i(v_j)_{j \prec i}. \tag{1}$$

Using above defined notation, the computer program can be expressed by the code list at the picture 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	$i = m-1, \ldots, 0$

Figure 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.

2.2 The first derivative computation - forward mode

The *forward* mode evaluates directional derivatives of all dependent variables. We assign a new *adjoint* function to every basic or elemental function (1):

$$\dot{v}_i = \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j. \tag{2}$$

$$\begin{array}{l}
\left.\begin{array}{lcl} v_{i-n} & = & x_i \\ \dot{v}_{i-n} & = & \dot{x}_i \end{array}\right\} i = 1, \ldots n \\ \hline
\left.\begin{array}{lcl} v_i & = & \varphi_i(v_k)_{k \prec i} \\ \dot{v}_i & = & \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j \end{array}\right\} i = 1, \ldots l \\ \hline
\left.\begin{array}{lcl} y_{m-i} & = & v_{l-i} \\ \dot{y}_{m-i} & = & \dot{v}_{l-i} \end{array}\right\} i = m-1, \ldots 0
\end{array}$$

Figure 2: Code list derived by forward mode.

which is a derivative of (1). The *adjoint* code list is at the picture 2.
We evaluate the derivative in the point $x = (x_1, x_2, \ldots, x_n)$ in the direction $\dot{x}$. If $\dot{x} = e_i$, then the partial derivative with respect to the ith variable is computed.

The schema (2) can be expressed, in general, in the form of a matrix multiplication (see e.g. [3])

$$f'(x) = Q_m A_l A_{l-1} \cdots A_2 A_1 P_n^T. \tag{3}$$

The matrix A_i is the identity matrix except the ith row, where the element $\frac{\delta \varphi_i}{\delta v_j}$ lies in the jth column for all $j \prec i$.

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 [3])

$$\bar{x} = \bar{y} \cdot f'(x). \tag{4}$$

From the relations (3) and (4) we obtain a code list described at the picture 3. The values $\bar{v}_j$ can be interpreted by the formula

$$\bar{v}_j = \sum_{i\ ;\ i \succ j} \bar{v}_i \cdot \frac{\partial \varphi_i}{\partial v_j}. \tag{5}$$

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. [3]. 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

$$\bar{y} \cdot f''(x) \cdot \dot{x} + \dot{\bar{y}} \cdot f'(x), \tag{6}$$

where $\dot{\bar{y}}$ is an input parameter, similar to $\dot{x}$ or $\bar{y}$. Let us denote the expression (6) 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

```
$v_{i-n} = x_i$                                   $i = 1, \ldots, n$
$v_i = \varphi_i(v_k)_{k \prec i}$                $i = 1, \ldots, l$
$y_{m-i} = v_{l-i}$                               $i = m-1, \ldots, 0$
$\bar{v}_{l-i} = \bar{y}_{m-i}$                   $i = 0, \ldots, m-1$
$\bar{v}_i = 0$                                   $i = l-m, \ldots, 1-n$
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
$\bar{x}_i = \bar{v}_{i-n}$                       $i = n, \ldots, 1$
```

Figure 3: Code list derived by reverse mode.

to assign $\dot{\bar{y}} = 0$. The expression $\bar{y} \cdot f''(x) \cdot \dot{x}$ can be interpreted as

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

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

$$\dot{\bar{v}}_j = \dot{\bar{v}}_j + \dot{\bar{v}}_i \cdot \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} + \bar{v}_i \cdot \frac{\partial \dot{\varphi}_i}{\partial v_j}(v_k)_{k \prec i}, \tag{7}$$

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 at the picture 4. The values of the second derivatives $\bar{y} \cdot f''(x) \cdot \dot{x}$ are the components of the vector

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

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

$$\left.\begin{array}{lcl} v_{i-n} & = & x_i \\ \dot{v}_{i-n} & = & \dot{x}_i \end{array}\right\} \quad i = 1, \ldots, n$$

$$\left.\begin{array}{lcl} v_i & = & \varphi_i(v_k)_{k \prec i} \\ \dot{v}_i & = & \sum_{j \prec i} \frac{\partial \varphi_i}{\partial v_j}(v_k)_{k \prec i} \cdot \dot{v}_j \end{array}\right\} \quad i = 1, \ldots l$$

$$\left.\begin{array}{lcl} y_{m-i} & = & v_{l-i} \\ \dot{y}_{m-i} & = & \dot{v}_{l-i} \end{array}\right\} \quad i = m-1, \ldots 0$$

$$\left.\begin{array}{lcl} \bar{v}_{l-i} & = & \bar{y}_{m-i} \\ \dot{\bar{v}}_{l-i} & = & 0 \end{array}\right\} \quad i = 0, \ldots, m-1$$

$$\left.\begin{array}{lcl} \bar{v}_i & = & 0 \\ \dot{\bar{v}}_i & = & 0 \end{array}\right\} \quad i = 1-n, \ldots, l-m$$

```
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
```

$$\left.\begin{array}{lcl} \bar{x}_i & = & \bar{v}_{i-n} \\ \dot{\bar{x}}_i & = & \dot{\bar{v}}_{i-n} \end{array}\right\} \quad i = n, \ldots, 1$$

Figure 4: Code list derived by reverse and reverse mode for the second derivatives.

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 [8].

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(`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 od 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

By the automatic differentiation, it is possible to evaluate the first or the second derivatives of the functions defined by the variables `FMODELF`, `FMODELA` nebo `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 differentiaion. 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 [6]. 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 (1)

- `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 (5)

- `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 (2)
- REAL*8 VBARDT($NADARR) – is the array which stores the values $\dot{\bar{v}}_i$, see the relation (7)

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, VDOT, 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, VDOT, OPCODE, ARG1, ARG2 backwards and evaluates the derivatives (5), 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

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

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
    .
    .
    ENDIF
999 CONTINUE
    END
```

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

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

Let us define the function $f(x_1, x_2) = x_1 x_2 + 1$. 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 at the picture 5. 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.

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 [6].

4 Example

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

$$F(x) = \sum_{i=1}^{N} (N + i - P_i)^2 , \tag{8}$$

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_1x_2	definition of constant 1	x_1x_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	...

Figure 5: The main principles of the data storing in the arrays

where

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

where $mod(a,b)$ is a remainder for the division of a by b. We want to calculate a local minimum of the function F. Let $x_0 = (1, 1/2, \ldots, 1/N)$ be an initial approximation. The input file for the UFO system is at the picture 6. 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 `$IADA=1` on line number 21 is stated. The number of the independent variables `X(*)` is defined by the variable `$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 [8]. The evaluation of

$$(N + i - P_i)^2 \tag{9}$$

(see the relation 8) 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 (picture 6) 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 of `$IADA` was set to 1. The value of the variable `FGMODELA` is shown at the picture 7.

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 at

```
 ! 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
```

Figure 6: Input file for the UFO system for the example with automatic differentiation.

the picture 6 is transformed to lines 9 and 10 at the picture 7. Moreover, the statement on the line 15 at the picture 6 is transformed to lines 12 and 13 at the picture 7.

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 corresponds to the fourth line in the picture 3, 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 at the picture 6, i.e. the lines 6 – 8 at the picture 7, because they do not depend on the independent variables `X(.)`. The initial setting of the array index counter is on the line 1 at the picture 7.

The evaluation times for a local minimum search are compared in the table 1. In the second and third column, there are times for the case that automatic differentiation and

```
   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
```

Figure 7: The value of the variable `FGMODELF` for the example with automatic differentiation.

divided differences are used respectively, for various N. The table shows that automatic differentiation yields faster convergence.

Number of independent 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

Table 1: Comparison of evaluation time for a local minimum search.

Reference

[1] Computational differentiation – techniques, applications, and tools. (M. Berz, C. H. Bischof, G. F. Corliss, A. Griewank, ed.), SIAM, Philadelphia, 1996.

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

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

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

[5] J. Hartman: Realizace metod pro automatické derivování. Diploma Thesis. Faculty of Mathematics and Physics, Charles University, Prague, 2001.

[6] J. Hartman and L. Lukšan: Automatické derivování v systému UFO. Technical Report V-1002. ICS AS CR, Prague, 2007.

[7] J. Hartman and J. Zítko: Principy automatického derivování. Výzkumná zpráva. Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Prague, 2006.

[8] 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.

[9] A. Verma: Structured Automatic Differentiation. Ph.D. thesis, Cornell University, 1988.

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

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

[12] R. Griesse and A. Walther Evaluating Gradients in Optimal Control - Continuous Adjoints versus Automatic Differentiation. Journal of Optimization Theory and Applications, 122(1), pp. 63 – 86 (2004).