UFO 2000. Interactive System for Universal Functional Optimization

Lukšan, Ladislav
2000

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

UFO 2000
Interactive System for Universal Functional Optimization

L.Lukšan, M.Tůma, M.Šiška, J.Vlček, N.Ramešová

Technical report No. 826

December 2000

Institute of Computer Science, Academy of Sciences of the Czech Republic
Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic
phone: (+4202) 6884244 fax: (+4202) 8585789
e-mail: uivt@uivt.cas.cz

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

UFO 2000
Interactive System for Universal Functional Optimization [1]

L.Lukšan, M.Tůma, M.Šiška, J.Vlček, N.Ramešová

Technical report No. 826
December 2000

Abstract

This report contains a description of the interactive system for universal functional optimization UFO, version 2000.

Keywords

[1] This work was supported under grant No. 201/00/0080 given by the Czech Republic Grant Agency

Contents

1. Introduction to the UFO system

The universal functional optimization (UFO) system is an interactive modular system for solving both dense medium-size and sparse large-scale optimization problems. The UFO system can be used for the following applications:

1. Formulation and solution of particular optimization problems that are described in chapter 2.
2. Preparation of specialized optimization routines (or subroutines) based on methods described in chapter 3.
3. Designing and testing new optimization methods. The UFO system is a very useful tool for optimization algorithms development.

The special realization of the UFO system described in the subsequent text makes this system portable and extensible and we continue with its further development.

1.1. Philosophy of the UFO system

The UFO system is an open software system for solving a broad class of optimization problems. An optimization problem solution is processed in four phases. In the first phase the optimization problem is specified and an optimization method is selected. This can be made in three different ways:

1. The full dialogue mode: The problem specification and the method selection are realized by using a conversation between the user and the UFO system.
2. The batch mode: The problem specification and the method selection are realized by using the UFO control language. An input file written in the UFO control language has to be prepared and stored.
3. The combined mode: Only a part of the specification is written in the input file. The rest of the specification is obtained as in the dialogue mode. This possibility is usually the best one since the problem functions can be defined beforehand by using a convenient text editor.

The second phase is realized by using the UFO preprocessor. This preprocessor is written in the Fortran 77 language and its output is a Fortran 77 control program. This conception is very advantageous for the following reasons:

1. The Fortran 77 (full ANSI norm) is a sufficiently high and portable programming language. Moreover, this language is very suitable for numerical computations, and a broad class of subroutines is available in this field.
2. A control program, generated by the UFO preprocessor, calls for necessary modules only and its specification is very easy. Moreover, control program global declarations are determined by the problem size, which decreases storage requirements. This way overcomes an impossibility of dynamical declarations in the Fortran 77 language.
3. The UFO system is open. When a new class of optimization problems or optimization methods has to be included, one only needs to change the system templates and prepare new modules. The control program is composed of individual modules by using specifications in the first phase. This fact allows us to create a great number of various optimization methods and their modifications.

In the third phase, the control program is translated by using a Fortran 77 compiler and a final program is linked by using library modules. In the fourth phase, the final program is executed and thus results which can be viewed by using extensive output means are obtained.

The above conception is enabled by a special form of source modules. These modules usually consist of two parts, the interface template and the Fortran 77 realization. The interface template is used by the UFO preprocessor only and it serves for the control program generation (the part of control program corresponding to a given module is coded in the template). These templates also contain knowledge bases for an automatic selection of the optimization method. If the UFO system has to be extended then usually only templates, which do not need to be compiled, are changed. Besides interface templates, which are a part of source modules, special templates controlling the UFO preprocessor exist. A batch input file written in the UFO control language is one of these special templates.

The UFO macroprocessor works in two stages. In the first pass, the file `P.TMP` is created. This file is a control program ancestor containing some macroinstructions and macrovariables which are replaced in the second pass. The control program `P.FOR` is the result of the second pass.

1.2. Execution of the UFO system

The UFO system contains three basic procedures `GENER`, `COMPIL` and `UFOGO`. The UFO preprocessor is called if the statement

```
GENER input_name
```

is typed. Then the control program, written in the Fortran 77 language, is obtained. Furthermore, the compilation of the control program, followed by its loading and executing, is started if the statement

```
COMPIL output_name
```

is typed. Finally, all the UFO system phases are performed if the statement

```
UFOGO problem_name
```

is typed. Here `input_name` is the first part of the batch file name `input_name.UFO` that is used as a batch input for the control program generation, `output_name` is the first part of the text file name `output_name.OUT` that is used as a text output from the UFO system and `problem_name` is the first part of both the batch file name `problem_name.UFO` and the text file name `problem_name.OUT`. All these names have to be typed with capital letters in UNIX versions of the UFO system. If `GENER` and `UFOGO` statements do not contain a file name specification, then a full dialogue mode is considered (the batch file name is `STANDARD.UFO` in this case) and the standard text file name is `P.OUT`. If `COMPIL` statement does not contain a file name specification, then the standard text file name is `P.OUT`. The `UFOGO` statement has the same meaning as two consecutive statements `GENER` and `COMPIL`.

First we show how the batch mode proceeds. We suppose that the model function has the form

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

(the Rosenbrock function) and the starting point is $x_1 = -1.2$ and $x_2 = 1.0$. If we prepare the batch input file `P.UFO` of the form

```
$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
$MOUT=1
$NOUT=1
$BATCH
$STANDARD
```

and type the statement `UFOGO P`, then the following results appear in the output file `P.OUT`

```
CLASS = VM - LI1   UPDATE = B   MODEL = FF   HESF = D   NF =       2
  0 NIT=   40 NFV=  138 NFG=    0 NDC=    0 NCG=   0 F= .504D-13 G= .828D-05
FF =   .5038712822D-13
X  =   .1000000098D+01   .1000000177D+01
```

(in a PC computer). Batch files are written in the UFO control language. This language is described in section 4.1. Here we note that a certain experience with the UFO control language can be obtained by using the demo-files `PROB01.UFO`,..., `PROB20.UFO`. These demo-files contain 20 test problems described in chapter 7. We can solve them by using the statements `UFOGO PROB01`,..., `UFOGO PROB20`.

Besides the batch mode, we can use the full dialogue mode. The full dialogue mode is started if we use the statement `UFOGO` (without a batch input file specification). Full dialogue modes (text and graphic) are described in sections 4.3 and 4.4. An example which demonstrates the text dialogue mode applied to the Rosenbrock function is given in Appendix A.

Besides basic output files, the UFO system produces additional files which can contain some useful information. A list of the most important UFO files follows:

`P.UFO` - Batch input file.
`P.TMP` - Temporary file containing a control program ancestor generated in the first pass of the UFO macroprocessor.
`P.FOR` - Control program generated in the second pass of the UFO macroprocessor (UNIX versions use the file `P.F` in this case).
`P.OUT` - Text input file.
`P.DAT` - Stored values of problem variables.
`P.DIM` - Dimensions of basic problem vectors and matrices.
`P.SIF` - Messages of the SIF decoder.
`P.I` - Template given by the SIF decoder.

1.3. The UFO versions for PC computers

There are two UFO versions for PC computers. The MS DOS version requires the Microsoft Fortran Power Station compiler version 1.0 and it uses the DOS graphic system for the graphic dialogue (Section 4.4) and the graphic screen output (Section 5.3). The Windows (98, NT, 2000) version requires the Digital Visual Fortran compiler version 5 and higher but it does not contain graphical possibilities at present. Of course, the UFO system can also be used for PC computers with other compilers. In this case, the UNIX version of the UFO system has to be applied with procedures `GENER`, `COMPIL` and `UFOGO` modified for the compiler used. The PC versions are distributed by using the files `UFODOS.ZIP` and `UFOWIN.ZIP`, which contain templates `*.I`, sample input files `*.UFO`, sample output files `*.OUT`, procedures `*.BAT`, programs `*.EXE` and other important files together with the subdirectory `F32`, which contains libraries `*.LIB`. The installation of the UFO system is carried out by putting the `*.ZIP` file into the directory `UFO` and by using the routine `PKUNZIP`. If the connection to the CUTE test environment is required, then the subdirectory `UFO` has to be created and the `*.SIF` files from the CUTE collection (page `http://www.cse.clrc.ac.uk/Activity/CUTE`) have to be copied into this subdirectory.

The PC versions of the UFO system are provided with the special UFO environment, which makes the use of the UFO system easy. The UFO environment is called by using the statement `UFO` (program `UFO.EXE`). It is controlled by using the "pull-down" menu. The main menu is activated by pressing key <F10>. The UFO environment contains a source program editor whose control is similar to the Word Star editor and, therefore, to the most commonly used source program editors under the MS DOS system. All significant statements of the source program editor are available from the UFO environment menu.

Since the UFO environment menu is clearly understood we do not describe it (the description is given in [142]). We only show the usual way for operating input files. When the batch mode input file is prepared by using the source program editor we press key <F10> and find the command Run! in the UFO environment menu. This command starts the UFO preprocessor and its action corresponds to the statement `UFOGO` (with the input file which is loaded in the source program editor). An easier possibility

is pressing keys <Alt-1>. Similarly, pressing keys <Alt-9> has the same effect as the statement `GENER` (with the input file which is loaded in the source program editor). Furthermore, if the control program `P.FOR` is loaded in the source program editor, pressing keys <Alt-3> has the same effect as the statement `COMPIL` and pressing keys <Alt-5> causes an exit from the UFO environment.

1.4. The UFO versions for UNIX workstations

The UNIX versions of the UFO system are distributed by using the files `ufo.tar.gz`, `for.tar.gz`, `bel.tar.gz` and the procedure `instal`. File `ufo.tar.gz` contains templates `*.I`, sample input files `*.UFO`, sample output files `*.OUT`, procedures and other important files. File `for.tar.gz` contains the UFO source modules `*.F`. File `bel.tar.gz` contains the BEL (UFO preprocessor) source modules `*.F`. The installation of the UFO system is carried out by putting these files into the directory `ufo` and by using the statements `instal SGI`, `instal DIG`, `instal SUN` and `instal HP`, for Silicon Graphic, Digital, Sun and Hewlett Packard workstations, respectively (the Fortran 77 compiler is assumed). If the connection to the CUTE test environment is required, then the subdirectory `ufo` has to be created and the `*.SIF` files from the CUTE collection (page `http://www.cse.clrc.ac.uk/Activity/CUTE`) have to be copied into this subdirectory. The UNIX versions of the UFO system do not contain graphical possibilities at present.

1.5. Suggestions for beginners

If we want to solve a particular optimization problem, then the best way for understanding the UFO system is to find a similar problem in the list of sample problems (Chapter 7). This sample problem can be solved typing `UFOGO PROBxx` (xx is the number of the sample problem). After solving the sample problem, we can modify the batch input file `PROBxx.BAT` to describe our problem. Basic suggestions concerning the description of optimization problems are given in Chapter 2 and the most important features of the UFO control language are described in Chapter 4. Optimization methods need not be selected by user, the system automatically chooses a suitable method. On the other hand, if the user is familiar with optimization methods, then the method can be selected by using suggestions given in Chapter 3. Information about output possibilities can be found in Chapters 5 and special tools of the UFO system are described in Chapter 6.

2. Problems solved using the UFO system

The most general problem which can be solved by using the UFO system is a minimization of an objective function $F : R^n \to R$ over a set $X \subset R^n$. The objective function can have several forms determined using the macrovariable $MODEL:

$MODEL='FF' - General optimization. In this case

$$F(x) = \pm f^F(x)$$

where $f^F : R^n \to R$ is a real valued, so-called model function

$MODEL='FL' - Linear optimization. In this case

$$F(x) = \pm \ (f^F + \sum_{i=1}^{n} g_i^F x_i)$$

where f^F, g_i^F, $1 \le i \le n$, are real coefficients.

$MODEL='FQ' - Quadratic optimization. In this case

$$F(x) = \pm \ (f^F + \sum_{i=1}^{n} (g_i^F + \frac{1}{2} \sum_{j=1}^{n} h_{ij}^F x_j) x_i)$$

where f^F, $g_i^F, 1 \le i \le n,$, $h_{ij}^F, 1 \le i \le n, 1 \le j \le n$, are real coefficients.

$MODEL='AF' - Sum of function minimization. In this case

$$F(x) = \sum_{k=1}^{n_A} f_k^A(x)$$

where $f_k^A : R^n \to R, \ 1 \le k \le n_A$, are real valued, so-called approximating functions.

$MODEL='AQ' - Sum of square minimization. In this case

$$F(x) = \sum_{k=1}^{n_A} |f_k^A(x)|^2$$

where $f_k^A : R^n \to R, \ 1 \le k \le n_A$, are real valued, so-called approximating functions.

$MODEL='AP' - Sum of power minimization. In this case

$$F(x) = \sum_{k=1}^{n_A} |f_k^A(x)|^r$$

where $f_k^A : R^n \to R, \ 1 \le k \le n_A$, are real valued, so-called approximating functions and $1 < r < \infty$ is a real exponent.

$MODEL='AM' - Minimization of maximum (minimax). In this case

$$F(x) = \max_{1 \le k \le n_A} |f_k^A(x)|$$

where $f_k^A : R^n \to R, \ 1 \le k \le n_A$, are real valued, so-called approximating functions.

$MODEL='DF' - Minimization of the general integral criterion with respect to the state equations. In this case

$$F(x) = \int_{t_A^{min}}^{t_A^{max}} f^A(x, y_A(x,t_A), t_A)\, dt_A + f^F(x, y_A(x, t_A^{max}), t_A^{max})$$

and

$$\frac{dy_A(x,t_A)}{dt_A} = f^E(x, y_A(x,t_A), t_A),\ y^A(x, t_A^{min}) = f^Y(x)$$

where $f^A : R^{n+n_E+1} \to R$ is a real valued, smooth, so-called subintegral function, $f^F : R^{n+n_E+1} \to R$ is a real valued, smooth, so-called terminal function, $f^E : R^{n+n_E+1} \to R^{n_E}$ is a real valued, smooth, so-called state function and $f^Y : R^n \to R^{n_E}$ is a real valued, smooth, so-called initial function.

$MODEL='DQ' - Minimization of the sum of square integral criterion with respect to the state equations. In this case

$$F(x) = \frac{1}{2}\int_{t_A^{min}}^{t_A^{max}} \sum_{i=1}^{n_E} w_i^E(t_A)(y_i^A(x,t_A) - y_i^E(t_A))^2\, dt_A + \frac{1}{2}\sum_{i=1}^{n_E} w_i^E(y_i^A(x, t_A^{max}) - y_i^E)^2$$

and

$$\frac{dy_A(x,t_A)}{dt_A} = f^E(x, y_A(x,t_A), t_A),\ y^A(x, t_A^{min}) = f^Y(x)$$

where $f^E : R^{n+n_E+1} \to R^{n_E}$ is a real valued, smooth, so-called state function and $f^Y : R^n \to R^{n_E}$ is a real valued, smooth, so-called initial function.

$MODEL='DE' - Solving an initial value problem for a system of ordinary differential equations. In this case

$$\frac{dy_A(t_A)}{dt_A} = f^E(y_A(t_A), t_A),\ y^A(t_A^{min}) = y_A^{min}$$

where $f^E : R^{n_E+1} \to R^{n_E}$ is a real valued, smooth, so-called state function.

$MODEL='NE' - Solving a system of nonlinear functional equations

$$f_k^A(x) = 0, \qquad 1 \le k \le n_A$$

where $n_A = n$ ($MODEL='NE' is equivalent to $MODEL='AQ' if $n_A = n$).

The model function $f^F : R^n \to R$ can have several types of Hessian matrices specified by the macrovariable $HESF:

$HESF='D' - Dense Hessian matrix.
$HESF='S' - Sparse Hessian matrix with a general pattern.
$HESF='N' - Hessian matrix is not used.

The default option is $HESF='D'. The approximating functions $f_k^A : R^n \to R,\ 1 \le k \le n_A$, can have several types of Jacobian matrices specified by the macrovariable $JACA:

$JACA='D' - Dense Jacobian matrix.
$JACA='S' - Sparse Jacobian matrix with a general pattern.
$JACA='N' - Jacobian matrix is not used.

If the approximating functions are used then we can choose several types of the Hessian matrix represen-

tation. These types are again specified by the macrovariable $HESF:

$HESF='D'	- Dense Hessian matrix.
$HESF='S'	- Sparse Hessian matrix with a general pattern.
$HESF='B'	- Sparse Hessian matrix with a partitioned pattern
$HESF='N'	- Hessian matrix is not used.

If $JACA='D', then it must be either $HESF='D' or $HESF='N'. If $JACA='S', we can specify all types of Hessian matrices ($HESF='D', $HESF='S', $HESF='B', $HESF='N'). The representation $HESF='B' usually leads to more expensive matrix operations. Therefore, we recommend to prefer the choice $HESF='S' against the choice $HESF='B'.

The subintegral function, the terminal function, the state function and the initial function, which appeared in the case of dynamical systems optimization, are considered to be dense. Therefore we cannot use the specifications $HESF='S' or $HESF='B' in this case.

The set $X \subset R^n$ can be the whole R^n (unconstrained case) or defined by box constraints

$$\begin{array}{rcccll} x_i^L & \leq & x_i & & & \text{if} \quad i \in I_1 \\ & & x_i & \leq & x_i^U & \text{if} \quad i \in I_2 \\ x_i^L & \leq & x_i & \leq & x_i^U & \text{if} \quad i \in I_3 \\ x_i^L & = & x_i & & & \text{if} \quad i \in I_5 \end{array}$$

where $I_1 \cup I_2 \cup I_3 \cup I_5 \subset \{i \in N : 1 \leq i \leq n\}$, or by general linear constraints

$$\begin{array}{rcccll} c_k^L & \leq & \sum_{i=1}^{n} g_{ki}^C x_i & & & \text{if} \quad k \in L_1 \\ & & \sum_{i=1}^{n} g_{ki}^C x_i & \leq & c_k^U & \text{if} \quad k \in L_2 \\ c_k^L & \leq & \sum_{i=1}^{n} g_{ki}^C x_i & \leq & c_k^U & \text{if} \quad k \in L_3 \\ c_k^L & = & \sum_{i=1}^{n} g_{ki}^C x_i & & & \text{if} \quad k \in L_5 \end{array}$$

where $g_{ki}^C,\ 1 \leq k \leq n_C,\ 1 \leq i \leq n$, are real coefficients and $L_1 \cup L_2 \cup L_3 \cup L_5 \subset \{k \in N : 1 \leq k \leq n_C\}$, or by general nonlinear constraints

$$\begin{array}{rcccll} c_k^L & \leq & f_k^C(x) & & & \text{if} \quad k \in N_1 \\ & & f_k^C(x) & \leq & c_k^U & \text{if} \quad k \in N_2 \\ c_k^L & \leq & f_k^C(x) & \leq & c_k^U & \text{if} \quad k \in N_3 \\ c_k^L & = & f_k^C(x) & & & \text{if} \quad k \in N_5 \end{array}$$

where $f_k^C : R^n \to R,\ 1 \leq k \leq n_C$, are real valued, smooth, so-called constraint functions and $N_1 \cup N_2 \cup N_3 \cup N_5 \subset \{k \in N : 1 \leq k \leq n_C\}$. The constraint functions $f_k^C : R^n \to R,\ 1 \leq k \leq n_C$, can have several types of Jacobian matrices specified by the macrovariable $JACC:

$JACC='D' - Dense Jacobian matrix.
$JACC='S' - Sparse Jacobian matrix with a general pattern.

If $JACC='D', then it must be $HESF='D' or $HESF='N'. If $JACC='S', then it must be $HESF='S' or $HESF='N'.

There are several limitations in the current version of the UFO system:

1. Minimization of maximum (minimax) and nonsmooth optimization are not implemented in the sparse case.
2. Minimization of dynamical systems is not implemented in the sparse case.
3. Usually the UFO system serves for local optimization. Global optimization can be used only for relatively small ($n \leq 20$) dense problems that are unconstrained or contain box constraints.

These limitations will be consecutively removed in subsequent versions of the UFO system.

In the rest of this report we will use the notation NF, NA, NC instead of n, n_A, n_C and X, FF(X), GF(X), HF(X), FA(KA;X), GA(KA;X), FC(KC;X), GC(KC;X) instead of x, $f^F(x)$, $g^F(x)$, $h^F(x)$, $f_k^A(x)$, $g_k^A(x)$, $f_k^C(x)$, $g_k^C(x)$. This new notation corresponds to that of the variables and of the fields in the UFO system.

2.1. Specification of variables

First we must specify the number of variables using the statement $NF=number_of_variables. If there are no box constraints we set $KBF=0. In the opposite case we set $KBF=1 or $KBF=2. If $KBF=1 or $KBF=2, then

$$\begin{array}{lcll} \text{X(I)} & - & \text{unbounded} & \text{, if IX(I)} = 0 \\ \text{XL(I)} & \leq & \text{X(I)} & \text{, if IX(I)} = 1 \\ & & \text{X(I)} \leq \text{XU(I)} & \text{, if IX(I)} = 2 \\ \text{XL(I)} & \leq & \text{X(I)} \leq \text{XU(I)} & \text{, if IX(I)} = 3 \\ \text{X(I)} & - & \text{constant} & \text{, if IX(I)} = 5 \end{array}$$

where 1≤ I≤ NF. The option $KBF=2 must be chosen if IX(I)=3 for at least one index 1≤ I≤ NF. Then two different fields XL(I) and XU(I), 1≤ I≤ NF are declared. In the opposite case we set $KBF=1 and only one common field XL(I)=XU(I), 1 ≤ I≤ NF is declared.

The initial values of variables X(I), 1≤ I≤ NF, types of box constraints IX(I), 1≤ I≤ NF, and lower and upper bounds XL(I) and XU(I), 1≤ I≤ NF, can be specified using macrovariable $INPUT. The default values are IX(I)=0 and XL(I)=XU(I)=0, 1≤ I≤ NF. For example:

$KBF=2; $NF=4
$SET(INPUT)
X(1)=x_1
X(2)=x_2; IX(2)=1; XL(1)=x_1^L
X(3)=x_3; IX(3)=3; XL(3)=x_3^L; XU(3)=x_3^U
X(4)=x_4; IX(4)=5
$ENDSET

The UFO system allows us to use a scaling of variables (for instance if the values of variables differ very much in their magnitude). We set:

$NORMF=1 - Scaling parameters XN(I), 1≤ I≤ NF, are determined automatically so that X(I)/XN(I)=1, 1≤I≤NF, for the initial values of variables.

$NORMF=2 - Scaling parameters must be specified by the user by means of the macrovariable $INPUT.

The scaling of variables is recommended only in exceptional cases since it increases the computational time and storage requirements. The scaling of variables is suppressed if $NORMF=0 (this value is default). The scaling of variables is not permitted in the case of general constraints (if KBC>0).

2.2. Specification of the model function (dense problems)

If the macrovariable $MODEL is not specified or if $MODEL='FF', then the objective function is defined by the formula

F(X) = + FF(X) if $IEXT = 0 (minimization)

or

F(X) = − FF(X) if $IEXT = 1 (maximization)

Option $IEXT=0 is default.

The model function FF(X) must be defined by the user either directly in the full dialogue mode, or by using corresponding macrovariables in the batch (or mixed) mode. The value of the model function is specified by using the macrovariable $FMODELF:

```
$SET(FMODELF)
   FF = value FF(X)
   (for given values of variables X(I), 1≤ I≤ NF)
$ENDSET
```

The first derivatives of the model function are specified by using the macrovariable $GMODELF:

```
$SET(GMODELF)
   GF(1) = derivative ∂FF(X)/ ∂X(1)
   GF(2) = derivative ∂FF(X)/ ∂X(2)
   ——
   GF(NF) = derivative ∂FF(X)/ ∂X(NF)
   (for given values of variables X(I), 1≤I≤NF)
$ENDSET
```

The second derivatives of the model function are specified by using the macrovariable $HMODELF. If $HESF='D', the Hessian matrix is assumed to be dense and we specify only its upper half:

```
$SET(HMODELF)
   HF(1) = derivative ∂²FF(X)/ ∂X(1)²
   HF(2) = derivative ∂²FF(X)/ ∂X(1)∂X(2)
   HF(3) = derivative ∂²FF(X)/ ∂X(2)²
   HF(4) = derivative ∂²FF(X)/ ∂X(1) ∂X(3)
   HF(5) = derivative ∂²FF(X)/ ∂X(2) ∂X(3)
   HF(6) = derivative ∂²FF(X)/ ∂X(3)²
   ——
   HF(NF*(NF+1)/2) = derivative ∂²FF(X)/ ∂X(NF)²
  (for given values of variables X(I), 1≤I≤NF)
$ENDSET
```

If the macrovariables $GMODELF or $HMODELF are not defined, we suppose that the first or the second derivatives of the model function are not given analytically. In this case, they are computed numerically by using the UFO system routines whenever it is required. If it is advantageous to compute the first derivatives of the model function FF(X) together with its value, we can replace the set of models

$FMODELF, $GMODELF by the common model $FGMODELF. Similarly we can replace the set of models $FMODELF, $GMODELF, $HMODELF by the common model $FGHMODELF.

To improve the efficiency of the computation, we can specify additional information about the model function FF(X). The first piece of information, useful for an automatic choice of the optimization method, is the computational complexity specified by the macrovariable $KCF:

$KCF=1 - Evaluation of the model function FF(X) is very easy (it requires $O(n)$ simple operations at most).

$KCF=2 - Evaluation of the model function FF(X) is of medium complexity (it at least requires $O(n)$ complicated operations and $O(n^2)$ simple operations at most).

$KCF=3 - Evaluation of the model function FF(X) is extremely difficult (it at least requires $O(n^2)$ complicated or $O(n^3)$ simple operations).

The option $KCF=2 is default. An additional useful piece of information is the analytical complexity (differentiability and conditioning), which is specified by the macrovariable $KSF:

$KSF=1 - The model function FF(X) is smooth and well-conditioned.

$KSF=2 - The model function FF(X) is smooth but ill-conditioned.

$KSF=3 - The model function FF(X) is nonsmooth.

The option $KSF=1 is default. Other specifications which can improve the computational efficiency and robustness of optimization methods are a lower bound of the objective function values and an upper bound of the stepsize. Both these values depend on the definition of the objective function and can be specified by the statements $FMIN=lower_bound (for the objective function) and $XMAX=upper_bound (for the stepsize). We recommend a definition of $FMIN whenever it is possible and a definition of $XMAX whenever the objective function contains exponentials.

If $MODEL='FL', we suppose the model function to be linear of the form

$$\mathrm{FF(X)} = \mathrm{FF} + \sum_{\mathrm{I=1}}^{\mathrm{NF}} \mathrm{GF(I)} * \mathrm{X(I)}$$

In this case we need not specify the value and the first derivatives of the model function by the macrovariables $FMODELF and $GMODELF as in the general case. Instead, we must specify the coefficients FF (constant value) and GF(I), 1≤ I≤ NF, (constant gradient) using the macrovariable $INPUT:

```
$ADD(INPUT)
   FF = constant value
   GF(1) = constant derivative ∂FF(X)/∂X(1)
   GF(2) = constant derivative ∂FF(X)/∂X(2)
   ——
   GF(NF) = constant derivative ∂FF(X)/∂X(NF)
$ENDADD
```

If $MODEL='FL', we usually assume that either box constraints or general linear constraints are given. In this case the optimization problem is the linear programming problem.

If $MODEL='FQ', we suppose the model function to be quadratic of the form

$$\mathrm{FF(X)} = \mathrm{FF} + \sum_{\mathrm{I=1}}^{\mathrm{NF}} \mathrm{GF(I)} * \mathrm{X(I)} + \frac{1}{2} \sum_{\mathrm{I=1}}^{\mathrm{NF}} \sum_{\mathrm{J=1}}^{\mathrm{NF}} \mathrm{HF(K)} * \mathrm{X(I)} * \mathrm{X(J)}$$

where K=MAX(I,J)*(MAX(I,J)-1)/2+MIN(I,J). In this case we need not specify the value, the first derivatives and the second derivatives of the model function by the macrovariables $FMODELF, $GMODELF and $HMODELF as in the general case. The coefficients FF (constant value) and GF(I), 1≤ I≤ NF, (constant gradient) are specified in the same way as in the linear case. The coefficients HF(K), 1≤ K≤ NF*(NF+1)/2, (the constant Hessian matrix) must be specified using the macrovariable $INPUT. If $HESF='D', the Hessian matrix is assumed to be dense and we specify only its upper half:

$ADD(INPUT)
HF(1) = constant derivative $\partial^2 FF(X)/\partial X(1)^2$
HF(2) = constant derivative $\partial^2 FF(X)/\partial X(1)\partial\ X(2)$
HF(3) = constant derivative $\partial^2 FF(X)/\partial X(2)^2$
HF(4) = constant derivative $\partial^2 FF(X)/\partial X(1)\partial\ X(3)$
HF(5) = constant derivative $\partial^2 FF(X)/\partial X(2)\partial\ X(3)$
HF(6) = constant derivative $\partial^2 FF(X)/\partial X(3)^2$
—
HF(NF*(NF+1)/2) = constant derivative $\partial^2 FF(X)/\partial X(NF)^2$
$ENDADD

If $MODEL='FQ', we usually assume that either box constraints or general constraints are given. In this case the optimization problem is the quadratic programming problem.

If the model function is linear or quadratic, then the options $KCF and $KSF need not be defined since they are not used.

2.3. Specification of the model function (sparse problems)

The UFO system contains optimization methods which take into account the sparsity pattern of the Hessian matrix HF. This possibility decreases the computational time and storage requirements for large-scale optimization problems. In this case we use the option $HESF='S' which means that the sparsity pattern is specified. All other specifications remain the same as in the case of dense problems. The sparsity pattern of the Hessian matrix is specified by using the macrovariable $INPUT. Two integer vectors IH and JH are used where IH(I), $1 \leq$ I$\leq$ NF+1, are pointers and JH(K), $1 \leq$ K$\leq$ M, are indices of nonzero elements. Only the upper half of the Hessian matrix is assumed and the nonzero elements are ordered in rows. The number of nonzero elements must be specified using the statement $M=number_of_elements. The number of nonzero elements could be greater than is required (twice say) since it is used for the declaration of working fields. For example, if we have the Hessian matrix

$$HF = \begin{pmatrix} h^F_{11}, & h^F_{12}, & h^F_{13}, & 0, & h^F_{15} \\ h^F_{21}, & h^F_{22}, & 0, & h^F_{24}, & 0 \\ h^F_{31}, & 0, & h^F_{33}, & 0, & h^F_{35}, \\ 0, & h^F_{42}, & 0, & h^F_{44}, & 0, \\ h^F_{51}, & 0, & h^F_{53}, & 0, & h^F_{55} \end{pmatrix}$$

then we have to set:

$NF=5
$M=20 (the minimum required value is M=10)
$ADD(INPUT)
IH(1)=1; IH(2)=5; IH(3)=7
IH(4)=9; IH(5)=10; IH(6)=11
JH(1)=1; JH(2)=2; JH(3)=3; JH(4)=5; JH(5)=2
JH(6)=4; JH(7)=3; JH(8)=5; JH(9)=4; JH(10)=5
$ENDADD

All diagonal elements of the sparse Hessian matrix are assumed to be nonzero.

As in the case of the dense problem, the second derivatives of the model function can be specified by using the macrovariable $HMODELF. If $HESF='S', only nonzero elements of the upper half (including the diagonal) of the Hessian matrix are specified. For the above example the specification has the form:

$SET(HMODELF)
HF(1)=h^F_{11}; HF(2)=h^F_{12}; HF(3)=h^F_{13}; HF(4)=h^F_{15}
HF(5)=h^F_{22}; HF(6)=h^F_{24}; HF(7)=h^F_{33}; HF(8)=h^F_{35}
HF(9)=h^F_{44}; HF(10)=h^F_{55}
$ENDSET

If the model function is quadratic (i.e. if $MODEL='FQ') and if $HESF='S', then the coefficients HF(K), 1≤ K≤ M, (constant sparse Hessian matrix) must be specified by using the macrovariable $INPUT. If the matrix given in the above example is the constant sparse Hessian matrix, we use the following specification:

$ADD(INPUT)
HF(1)=h^F_{11}; HF(2)=h^F_{12}; HF(3)=h^F_{13}; HF(4)=h^F_{15}
HF(5)=h^F_{22}; HF(6)=h^F_{24}; HF(7)=h^F_{33}; HF(8)=h^F_{35}
HF(9)=h^F_{44}; HF(10)=h^F_{55}
$ENDADD

2.4. Objective functions for discrete approximation

If we set $MODEL='AF', then we suppose that the objective function F(X) has this form:

$$\mathrm{F(X)} = \sum_{\mathrm{KA=1}}^{\mathrm{NA}} \mathrm{FA(KA;X)} \qquad \text{if KBA} = 0$$

or

$$\mathrm{F(X)} = \sum_{\mathrm{KA=1}}^{\mathrm{NA}} \mathrm{AW(KA)} * (\mathrm{FA(KA;X)} - \mathrm{AM(KA)}) \qquad \text{if KBA} = 1$$

where FA(KA;X), 1≤ KA≤ NA, are approximating functions. This form of the objective function is very useful in large-scale optimization when the approximating functions FA(KA;X), 1≤ KA≤ NA, are assumed to have sparse gradients.

If we set $MODEL='AP', then we suppose that the objective function F(X) has this form:

$$\mathrm{F(X)} = \frac{1}{\mathrm{R}} \sum_{\mathrm{KA=1}}^{\mathrm{NA}} |\mathrm{FA(KA;X)}| **\mathrm{R} \qquad \text{if KBA} = 0$$

or

$$\mathrm{F(X)} = \frac{1}{\mathrm{R}} \sum_{\mathrm{KA=1}}^{\mathrm{NA}} |\mathrm{AW(KA)} * (\mathrm{FA(KA;X)} - \mathrm{AM(KA)})| **\mathrm{R} \qquad \text{if KBA} = 1$$

where FA(KA;X), 1≤ KA≤ NA, are approximating functions, and R>1 is a real exponent. The value of the exponent is specified by the choice $REXP=R (default value is $REXP=2). Since the most used value of the exponent is R=2, and since the computations are the simplest and the most efficient for such a choice, we can use the specification $MODEL='AQ' in this case (minimization of the sum of squares). Moreover, $MODEL='AQ' is formally set whenever we choose $MODEL='AP' and $REXP=2.

If we set $MODEL='AM', then we suppose that the objective function F(X) has the form:

$$\mathrm{F(X)} = \max_{1\le \mathrm{KA}\le \mathrm{NA}} (+\mathrm{FA(KA;X)}) \quad \text{if } \$\mathrm{IEXT} = -1$$

$$\mathrm{F(X)} = \max_{1\le \mathrm{KA}\le \mathrm{NA}} (|\mathrm{FA(KA;X)}|) \quad \text{if } \$\mathrm{IEXT} = 0$$

$$F(X) = \max_{1 \le KA \le NA} (-FA(KA;X)) \quad \text{if \$IEXT} = +1$$

for \$KBA=0, or

$$F(X) = \max_{1 \le KA \le NA} (+AW(KA) * (FA(KA;X) - AM(KA))) \quad \text{if \$IEXT} = -1$$

$$F(X) = \max_{1 \le KA \le NA} (|AW(KA) * (FA(KA;X) - AM(KA))|) \quad \text{if \$IEXT} = 0$$

$$F(X) = \max_{1 \le KA \le NA} (-AW(KA) * (FA(KA;X) - AM(KA))) \quad \text{if \$IEXT} = +1$$

for \$KBA=1, where FA(KA;X), 1≤ KA≤ NA, are approximating functions. The default value is \$IEXT=0 (the minimax or the Chebyshev approximation).

The option \$KBA serves as a decision between a simple objective function and a more complicated one. The simple objective function uses no additional fields while the more complicated one uses two additional fields at most, AM and AW. Vector AM usually contains frequently used observations which can be included into the functions FA(KA;X), 1≤ KA≤ NA, in the case of the simple objective function. Observations AM(KA), 1≤ KA≤ NA, are specified by using the macrovariable \$INPUT. Their default values are AM(KA)=0, 1≤ KA≤ NA. Vector AW serves for possible scaling specified by the option \$NORMA:

\$NORMA=0 - No scaling is performed. In this case AW(KA)=1, 1≤ KA≤ NA.
\$NORMA=1 - Scaling parameters are determined automatically so that AW(KA)=|AM(KA)|, 1≤ KA≤ NA.
\$NORMA=2 - Scaling parameters must be specified by the user by means of the macrovariable \$INPUT.

The number of approximating functions NA must be specified, in all the above cases, by using the statement \$NA=number_of_functions.

2.5. Specification of the approximating functions (dense problems)

The approximating functions FA(KA;X), 1≤ KA≤ NA, must be defined by the user either directly in the full dialogue mode or by using corresponding macrovariables in the batch (or mixed) mode. The values of the approximating functions are specified by using the macrovariables \$FMODELA or \$FMODELAS:

```
$SET(FMODELA)
      FA = value FA(KA;X)
      (for a given index KA and given values of variables X(I), 1≤ I≤ NF)
$ENDSET
```

or

```
$SET(FMODELAS)
      AF(1) = value FA(1;X)
      AF(2) = value FA(2;X)
      ---
      AF(NA) = value FA(NA;X)
$ENDSET
```

The first derivatives of the approximating functions are specified by using the macrovariables \$GMODELA or \$GMODELAS:

$SET(GMODELA)
GA(1) = derivative ∂FA(KA;X)/∂X(1)
GA(2) = derivative ∂FA(KA;X)/∂X(2)
——
GA(NF) = derivative ∂FA(KA;X)/∂X(NF)
(for a given index KA and given values of variables X(I), 1$\leq$ I$\leq$ NF)
$ENDSET

or

$SET(GMODELAS)
AG(1) = derivative ∂FA(1;X)/∂X(1)
AG(2) = derivative ∂FA(1;X)/∂X(2)
——
AG(NF) = derivative ∂FA(1;X)/∂X(NF)
AG(NF+1) = derivative ∂FA(2;X)/∂X(1)
AG(NF+2) = derivative ∂FA(2;X)/∂X(2)
——
AG(NA*NF) = derivative ∂FA(NA;X)/∂X(NF)
$ENDSET

The second derivatives of the approximating functions are specified by using the macrovariables $HMODELA or $HMODELAS. If $JACA='D', the Hessian matrices are assumed to be dense and we specify only their upper half:

$SET(HMODELA)
HA(1) = derivative ∂^2FA(KA;X)/∂X(1)^2
HA(2) = derivative ∂^2FA(KA;X)/∂X(1)∂X(2)
HA(3) = derivative ∂^2FA(KA;X)/∂X(2)^2
HA(4) = derivative ∂^2FA(KA;X)/∂X(1)∂X(3)
HA(5) = derivative ∂^2FA(KA;X)/∂X(2)∂X(3)
HA(6) = derivative ∂^2FA(KA;X)/∂X(3)^2
——
HA(NF*(NF+1)/2) = derivative ∂^2FA(KA;X)/∂X(NF)^2
(for a given index KA and given values of variables X(I), 1$\leq$ I$\leq$ NF)
$ENDSET

or

$SET(HMODELAS)
AH(1) = derivative ∂^2FA(1;X)/∂X(1)^2
AH(2) = derivative ∂^2FA(1;X)/∂X(1)∂X(2)
AH(3) = derivative ∂^2FA(1;X)/∂X(2)^2
AH(4) = derivative ∂^2FA(1;X)/∂X(1)∂X(3)
AH(5) = derivative ∂^2FA(1;X)/∂X(2)∂X(3)
AH(6) = derivative ∂^2FA(1;X)/∂X(3)^2
——
AH(NF*(NF+1)/2) = derivative ∂^2FA(1;X)/∂X(NF)^2
AH(NF*(NF+1)/2+1) = derivative ∂^2FA(2;X)/∂X(1)^2
AH(NF*(NF+1)/2+2) = derivative ∂^2FA(2;X)/∂X(1)∂X(2)
AH(NF*(NF+1)/2+3) = derivative ∂^2FA(2;X)/∂X(2)^2
——
AH(NA*NF*(NF+1)/2) = derivative ∂^2FA(NA;X)/∂X(NF)^2
$ENDSET

If the macrovariables $GMODELA and $GMODELAS or $HMODELA and $HMODELAS are not defined, we suppose that the first or the second derivatives of the approximating functions are not given analytically. In this case, they are computed numerically by using the UFO system routines, whenever it is required. If it is advantageous to compute the first derivatives of the approximating functions FA(KA;X), 1≤ KA≤ NA, together with their values, we can replace the set of models $FMODELA, $GMODELA by the common model $FGMODELA and the set of models $FMODELAS, $GMODELAS by the common model $FGMODELAS. Similarly we can replace the set of models $FMODELA, $GMODELA, $HMODELA by the common model $FGHMODELA and the set of models $FMODELAS, $GMODELAS, $HMODELAS by the common model $FGHMODELAS.

To improve the efficiency of the computation, we can specify additional information about the approximating functions FA(KA;X), 1≤ KA≤ NA. The first piece of information, useful for an automatic choice of the optimization method, is the computational complexity specified by the macrovariable $KCA:

$KCA=1 - Evaluations of the approximating functions FA(KA;X), 1≤ KA≤ NA, are very easy (they require $O(n)$ simple operations at most).

$KCA=2 - Evaluations of the approximating functions FA(KA;X), 1≤ KA≤ NA, are of medium complexity (they at least require $O(n)$ complicated operations and $O(n^2)$ simple operations at most).

$KCA=3 - Evaluations of the approximating functions FA(KA;X), 1≤ KA≤ NA, are extremely difficult (they at least require $O(n^2)$ complicated or $O(n^3)$ simple operations).

The option $KCA=2 is default. An additional useful piece of information is the analytical complexity (conditioning) which is specified by the macrovariable $KSA:

$KSA=1 - The approximating functions FA(KA;X), 1≤ KA≤ NA, are smooth and well-conditioned.

$KSA=2 - The approximating functions FA(KA;X), 1≤ KA≤ NA, are smooth but ill-conditioned.

$KSA=3 - The approximating functions FA(KA;X), 1≤ KA≤ NA, are nonsmooth.

The option $KSA=1 is default.

If some of the approximating functions are linear and have the form

$$\mathrm{FA(KA;\,X)} = \sum_{\mathrm{I=1}}^{\mathrm{NF}} \mathrm{AG((KA-1)*NF+I)*X(I)}$$

we can specify them separately. Then the number of linear approximating functions must be specified by using the statement $NAL=number_of_linear_functions (default value is $NAL=0). We always suppose that the first NAL approximating functions are linear. Then the coefficients AG((KA-1)*NF+I), 1≤ KA≤ NAL, 1≤ I≤ NF, are specified using the macrovariable $INPUT, and the macrovariables $FMODELA or $FMODELAS, $GMODELA or $GMODELAS, $HMODELA or $HMODELAS are used only for the specification of the nonlinear approximating functions FA(KA;X), NAL<KA≤ NA.

2.6. Specification of the approximating functions (sparse problems)

The UFO system contains optimization methods which take into account the sparsity pattern of the Jacobian matrix AG. This possibility decreases the computational time and storage requirements for large-scale optimization problems. In this case we use the option $JACA='S' which means that the sparsity pattern is specified. All other specifications remain the same as in the case of dense problems. The sparsity pattern of the Jacobian matrix is specified by using the macrovariable $INPUT. Two integer vectors IAG and JAG are used where IAG(KA), 1≤ KA≤ NA+1, are pointers and JAG(K), 1≤ K≤ IAG(NA+1)-1, are indices of nonzero elements. Nonzero elements are ordered by the gradients of the approximating functions. The number of nonzero elements must be specified by using the statement $MA=number_of_elements. For example, if we have the gradients

$$\mathrm{GA}(1;\mathrm{X}) = [g^A_{11}, 0\ \ , 0\ \ , g^A_{14}],$$

$$\mathrm{GA}(2;\mathrm{X}) = [0\ \ , g^A_{22}, 0\ \ , g^A_{24}],$$

$$\mathrm{GA}(3;\mathrm{X}) = [0\ \ , 0\ \ , g^A_{33}, 0\ \ ],$$

$$\mathrm{GA}(4;\mathrm{X}) = [g^A_{41}, g^A_{42}, g^A_{43}, 0\ \ ],$$

$$\mathrm{GA}(5;\mathrm{X}) = [0\ \ , 0\ \ , g^A_{53}, g^A_{54}],$$

and the Jacobian matrix

$$\mathrm{AG}(\mathrm{X}) = \begin{pmatrix} g^A_{11} & ,0 & ,0 & ,g^A_{14} \\ 0 & ,g^A_{22} & ,0 & ,g^A_{24} \\ 0 & ,0 & ,g^A_{33} & ,0 \\ g^A_{41} & ,g^A_{42} & ,g^A_{43} & ,0 \\ 0 & ,0 & ,g^A_{53} & ,g^A_{54} \end{pmatrix}$$

then we have to set:

```
$NA=5
$MA=10
$ADD(INPUT)
      IAG(1)=1; IAG(2)=3; IAG(3)=5
      IAG(4)=6; IAG(5)=9; IAG(6)=11
      JAG(1)=1; JAG(2)=4; JAG(3)=2; JAG(4)=4; JAG(5)=3
      JAG(6)=1; JAG(7)=2; JAG(8)=3; JAG(9)=3; JAG(10)=4
$ENDADD
```

As in the case of the dense problem, the first derivatives of the approximating functions can be specified by using the macrovariables $GMODELA or $GMODELAS. If $JACA='S', only nonzero elements of the gradients are specified. For the above example the specifications have the form

```
$SET(GMODELA)
      IF (KA.EQ.1) THEN
            GA(1) = ∂FA(1;X)/∂X(1)
            GA(4) = ∂FA(1;X)/∂X(4)
      ELSE IF (KA.EQ.2) THEN
            GA(2) = ∂FA(2;X)/∂X(2)
            GA(4) = ∂FA(2;X)/∂X(4)
      ELSE IF (KA.EQ.3) THEN
            GA(3) = ∂FA(3;X)/∂X(3)
      ELSE IF (KA.EQ.4) THEN
            GA(1) = ∂FA(4;X)/∂X(1)
            GA(2) = ∂FA(4;X)/∂X(2)
            GA(3) = ∂FA(4;X)/∂X(3)
      ELSE
            GA(3) = ∂FA(5;X)/∂X(3)
            GA(4) = ∂FA(5;X)/∂X(4)
      ENDIF
$ENDSET
```

or

```
$SET(GMODELAS)
      AG(1) = ∂FA(1;X)/∂X(1)
      AG(2) = ∂FA(1;X)/∂X(4)
      AG(3) = ∂FA(2;X)/∂X(2)
      AG(4) = ∂FA(2;X)/∂X(4)
      AG(5) = ∂FA(3;X)/∂X(3)
      AG(6) = ∂FA(4;X)/∂X(1)
      AG(7) = ∂FA(4;X)/∂X(2)
      AG(8) = ∂FA(4;X)/∂X(3)
      AG(9) = ∂FA(5;X)/∂X(3)
      AG(10) = ∂FA(5;X)/∂X(4)
$ENDSET
```

As in the case of the dense problem, the second derivatives of the approximating functions can be specified by using the macrovariables $HMODELA or $HMODELAS. If $JACA='S', only nonzero elements of the Hessian matrices are specified. For the above example the specifications have the form

```
$SET(HMODELA)
      IF (KA.EQ.1) THEN
            HA(1) = ∂²FA(1;X)/∂X(1)²
            HA(2) = ∂²FA(1;X)/∂X(1)∂X(4)
            HA(3) = ∂²FA(1;X)/∂X(4)²
      ELSE IF (KA.EQ.2) THEN
            HA(1) = ∂²FA(2;X)/∂X(2)²
            HA(2) = ∂²FA(2;X)/∂X(2)∂X(4)
            HA(3) = ∂²FA(2;X)/∂X(4)²
      ELSE IF (KA.EQ.3) THEN
            HA(1) = ∂²FA(3;X)/∂X(3)²
      ELSE IF (KA.EQ.4) THEN
            HA(1) = ∂²FA(4;X)/∂X(1)²
            HA(2) = ∂²FA(4;X)/∂X(1)∂X(2)
            HA(3) = ∂²FA(4;X)/∂X(2)²
            HA(4) = ∂²FA(4;X)/∂X(1)∂X(3)
            HA(5) = ∂²FA(4;X)/∂X(2)∂X(3)
            HA(6) = ∂²FA(4;X)/∂X(3)²
      ELSE
            HA(1) = ∂²FA(5;X)/∂X(3)²
            HA(2) = ∂²FA(5;X)/∂X(3)∂X(4)
            HA(3) = ∂²FA(5;X)/∂X(4)²
      ENDIF
$ENDSET
```

or

```
$SET(HMODELAS)
      AH(1) = ∂²FA(1;X)/∂X(1)²
      AH(2) = ∂²FA(1;X)/∂X(1)∂X(4)
      AH(3) = ∂²FA(1;X)/∂X(4)²
      AH(4) = ∂²FA(2;X)/∂X(2)²
      AH(5) = ∂²FA(2;X)/∂X(2)∂X(4)
      AH(6) = ∂²FA(2;X)/∂X(4)²
      AH(7) = ∂²FA(3;X)/∂X(3)²
```

AH(8) = ∂^2FA(4;X)/∂X(1)^2
AH(9) = ∂^2FA(4;X)/∂X(1)∂X(2)
AH(10) = ∂^2FA(4;X)/∂X(2)^2
AH(11) = ∂^2FA(4;X)/∂X(1)∂X(3)
AH(12) = ∂^2FA(4;X)/∂X(2)∂X(3)
AH(13) = ∂^2FA(4;X)/∂X(3)^2
AH(14) = ∂^2FA(5;X)/∂X(3)^2
AH(15) = ∂^2FA(5;X)/∂X(3)∂X(4)
AH(16) = ∂^2FA(5;X)/∂X(4)^2
$ENDSET

Note that the dimensions of arrays HA or AH must be specified by the statement $MHA=dimension_of_HA or $MAH=dimension_of_AH.

If some of the approximating functions are linear (i.e. if $NAL>0) and if $JACA='S', then the coefficients AG(K), 1$\leq$ K$\leq$ IAG(NAL+1)$-$1 (constant part of the sparse Jacobian matrix), must be specified by using the macrovariable $INPUT. If the matrix given in the above example is the constant sparse Jacobian matrix, we use this specification:

$ADD(INPUT)
AG(1)=g_{11}^A; AG(2)=g_{14}^A; AG(3)=g_{22}^A; AG(4)=g_{24}^A
AG(5)=g_{33}^A; AG(6)=g_{41}^A; AG(7)=g_{42}^A; AG(8)=g_{43}^A
AG(9)=g_{53}^A; AG(10)=g_{54}^A
$ENDADD

There is another possibility which can be useful when all approximating functions are linear. It is based on the usage of special procedure UKMAI1 which serves for a direct input of individual Jacobian matrix elements. The procedure UKMAI1 is formally called by using the statement

CALL $UKMAI1(K,I,GAKI) or $SETAG(K,I,GAKI)

where K is an index of a given approximating function (a row of the Jacobian matrix), I is an index of a given variable (a column of the Jacobian matrix), and GAKI is the numerical value of the element ∂FA(K;X)/∂X(I). For the example given above we can write:

$ADD(INPUT)
$SETAG(1,1,$g_{11}^A$)
$SETAG(1,4,$g_{14}^A$)
$SETAG(2,2,$g_{22}^A$)
$SETAG(2,4,$g_{24}^A$)
$SETAG(3,3,$g_{33}^A$)
$SETAG(4,1,$g_{41}^A$)
$SETAG(4,2,$g_{42}^A$)
$SETAG(4,3,$g_{43}^A$)
$SETAG(5,3,$g_{53}^A$)
$SETAG(5,4,$g_{54}^A$)
$ENDADD

The main advantage of the last possibility is the fact that it is not necessary to specify the fields IAG and JAG beforehand.

If we use the option $JACA='S', then we can specify a form of the objective function sparse Hessian matrix. There are four possibilities:

$HESF='D' - Dense Hessian matrix.

$HESF='B' - Partitioned sparse Hessian matrix. This matrix is a sum of simple Hessian matrices which correspond to the individual approximating functions. Only nonzero blocks are stored.

$HESF='S' - General sparse Hessian matrix (the same as the model function Hessian matrix corresponding to the option $HESF='S').

$HESF='N' - Hessian matrix is not used.

This specification only serves for an internal realization of optimization methods and has no influence on the user's input. The default option is $HESF='D'.

2.7. Objective functions for optimization of dynamical systems

If we set $MODEL='DF', then we suppose that the objective function F(X) has this form:

$$\mathrm{F(X)} = \int_{\mathrm{TAMIN}}^{\mathrm{TAMAX}} \mathrm{FA(X, YA(TA), TA)}\, d\mathrm{TA} + \mathrm{FF(X, YA(TAMAX), TAMAX)}$$

where FA(X,YA(TA),TA) is a smooth subintegral function and FF(X,YA(TAMAX),TAMAX) is a smooth terminal function. At the same time

$$\frac{d\mathrm{YA(KE;TA)}}{d\mathrm{TA}} = \mathrm{FE(KE;X, YA(TA), TA),\ YA(KE;TAMIN) = FY(KE;X)}$$

where FE(KE;X,YA(TA),TA), 1≤ KE≤ NE, are smooth state functions and FY(KE;X), 1≤ KE≤ NE, are smooth initial functions.

If we set $MODEL='DQ', then we suppose the objective function F(X) has the form:

$$\mathrm{F(X)} = \frac{1}{2}\int_{\mathrm{TAMIN}}^{\mathrm{TAMAX}} \sum_{\mathrm{KE=1}}^{\mathrm{NE}} \mathrm{WE(KE;TA)} * (\mathrm{YA(KE;TA)} - \mathrm{YE(KE;TA)})^2\, d\mathrm{TA}$$

$$+\frac{1}{2}\sum_{\mathrm{KE=1}}^{\mathrm{NE}} \mathrm{EW(KE)} * (\mathrm{YA(KE;TAMAX)} - \mathrm{EY(KE)})^2$$

At the same time

$$\frac{d\mathrm{YA(KE;TA)}}{d\mathrm{TA}} = \mathrm{FE(KE;X, YA(TA), TA),\ YA(KE;TAMIN) = FY(KE;X)}$$

where FE(KE;X,YA(TA),TA), 1≤ KE≤ NE, are smooth state functions and FY(KE;X), 1≤ KE≤ NE, are smooth initial functions.

If we set $MODEL='NO', then we consider that the initial value problem

$$\frac{d\mathrm{YA(KE;TA)}}{d\mathrm{TA}} = \mathrm{FE(KE;YA(TA), TA),\ YA(KE;TAMIN)}$$

is given where FE(KE;YA(TA),TA), 1≤ KE≤ NE, are smooth state functions.

In all the above cases, the statement $NE=number_of_differential_equations must be used for the specification of number of differential equations NE.

2.8. Specification of the state functions

The state functions FE(KE;X,YA(TA),TA), 1≤ KE≤ NE, must be defined by the user either directly in the full dialogue mode or by using corresponding macrovariables in the batch (or mixed) mode. The values of the state functions are specified by using the macrovariables $FMODELE or $FMODELES:

```
$SET(FMODELE)
        FE = value FE(KE;X,YA(TA),TA)
        (for a given index KE, a given vector of variables X,
        a given vector of state variables YA(TA) and a given time TA)
$ENDSET
```

or

```
$SET(FMODELES)
        EF(1) = value FE(1;X,YA(TA),TA)
        EF(2) = value FE(2;X,YA(TA),TA)
        ---
        EF(NE) = value FE(NE;X,YA(TA),TA)
$ENDSET
```

The first derivatives of the state functions according to the variables are specified by using the macrovariables $GMODELE or $GMODELES:

```
$SET(GMODELE)
        GE(1) = derivative ∂FE(KE;X,YA(TA),TA)/∂X(1)
        GE(2) = derivative ∂FE(KE;X,YA(TA),TA)/∂X(2)
        ---
        GE(NF) = derivative ∂FE(KE;X,YA(TA),TA)/∂X(NF)
        (for a given index KE, a given vector of variables X,
        a given vector of state variables YA(TA) and a given time TA)
$ENDSET
```

or

```
$SET(GMODELES)
        EG(1) = derivative ∂FE(1;X,YA(TA),TA)/∂X(1)
        EG(2) = derivative ∂FE(1;X,YA(TA),TA)/∂X(2)
        ---
        EG(NF) = derivative ∂FE(1;X,YA(TA),TA)/∂X(NF)
        EG(NF+1) = derivative ∂FE(2;X,YA(TA),TA)/∂X(1)
        EG(NF+2) = derivative ∂FE(2;X,YA(TA),TA)/∂X(2)
        ---
        EG(NE*NF) = derivative ∂FE(NE;X,YA(TA),TA)/∂X(NF)
$ENDSET
```

The first derivatives of the state functions according to the state variables are specified by using the macrovariables $DMODELE or $DMODELES:

```
$SET(DMODELE)
        DE(1) = derivative ∂FE(KE;X,YA(TA),TA)/∂YA(1)
        DE(2) = derivative ∂FE(KE;X,YA(TA),TA)/∂YA(2)
        ---
        DE(NE) = derivative ∂FE(KE;X,YA(TA),TA)/∂YA(NE)
        (for a given index KE, a given vector of variables X,
```

a given vector of state variables YA(TA) and a given time TA)
$ENDSET

or

$SET(DMODELES)
ED(1) = derivative ∂FE(1;X,YA(TA),TA)/∂YA(1)
ED(2) = derivative ∂FE(1;X,YA(TA),TA)/∂YA(2)
——
ED(NE) = derivative ∂FE(1;X,YA(TA),TA)/∂YA(NE)
ED(NE+1) = derivative ∂FE(2;X,YA(TA),TA)/∂YA(1)
ED(NE+2) = derivative ∂FE(2;X,YA(TA),TA)/∂YA(2)
——
ED(NE*NE) = derivative ∂FE(NE;X,YA(TA),TA)/∂YA(NE)
$ENDSET

If it is advantageous to compute the first derivatives of the state functions FE(KE;X,YA(TA),TA), $1 \leq$ KE$\leq$ NE, together with their values, we can replace the set of models $FMODELE, $GMODELE, $DMODELE by the common model $FGDMODELE and the set of models $FMODELES, $GMODELES, $DMODELES by the common model $FGDMODELES. Partially we can replace the models $FMODELE, $GMODELE or $FMODELE, $DMODELE or $GMODELE, $DMODELE by the common models $FGMODELE or $FDMODELE or $GDMODELE, respectively. Similarly we can replace the models $FMODELES, $GMODELES or $FMODELES, $DMODELES or $GMODELES, $DMODELES by the common models $FGMODELES or $FDMODELES or $GDMODELES, respectively.

If $MODEL='DQ', we have to define the functions WE(KE;TA) and YE(KE;TA), $1 \leq$ KE$\leq$ NE, for a given index KE and a given time TA. These functions can be specified by using the macrovariable $FMODELE together with the state function FE(KE;X,YA(TA),TA):

$SET(FMODELE)
FE = value FE(KE;X,YA(TA),TA)
WE = value WE(KE;TA)
YE = value YE(KE;TA)
(for a given index KE, a given vector of variables X,
a given vector of state variables YA(TA) and a given time TA)
$ENDSET

The default values WE(KE;TA)=1 and YE(KE;TA)=0 cannot be specified, they are supposed automatically.

2.9. Specification of the initial functions

The initial functions FY(KE;X), $1 \leq$ KE$\leq$ NE, must be defined by the user either directly in the full dialogue mode or by using corresponding macrovariables in the batch (or mixed) mode. The values of the initial functions are specified by using the macrovariables $FMODELY or $FMODELYS:

$SET(FMODELY)
FE = value FY(KE;X)
(for a given index KE and a given vector of variables X)
$ENDSET

or

$SET(FMODELYS)
EF(1) = value FY(1;X)
EF(2) = value FY(2;X)

```
        ——
        EF(NE) = value FY(NE;X)
$ENDSET
```

The first derivatives of the initial functions according to the variables are specified by using the macrovariables $GMODELY or $GMODELYS:

```
$SET(GMODELY)
        GE(1) = derivative ∂FY(KE;X)/∂X(1)
        GE(2) = derivative ∂FY(KE;X)/∂X(2)
        ——
        GE(NF) = derivative ∂FY(KE;X)/∂X(NF)
        (for a given index KE and a given vector of variables X)
$ENDSET
```

or

```
$SET(GMODELYS)
        EG(1) = derivative ∂FY(1;X)/∂X(1)
        EG(2) = derivative ∂FY(1;X)/∂X(2)
        ——
        EG(NF) = derivative ∂FY(1;X)/∂X(NF)
        EG(NF+1) = derivative ∂FY(2;X)/∂X(1)
        EG(NF+2) = derivative ∂FY(2;X)/∂X(2)
        ——
        EG(NE*NF) = derivative ∂FY(NE;X)/∂X(NF)
$ENDSET
```

If it is advantageous to compute the first derivatives of the initial functions FY(KE;X), $1 \leq$ KE$\leq$ NE, together with their values, we can replace the set of models $FMODELY, $GMODELY by the common model $FGMODELY and the set of models $FMODELYS, $GMODELYS by the common model $FGMODELYS.

If the initial values YA(KE;TAMIN), $1 \leq$ KE$\leq$ NE, do not depend on the variables X(I), $1 \leq$ I$\leq$ NF, they can be specified by using the macrovariable $INPUT:

```
$ADD(INPUT)
        YA(1) = initial value YA(1,TAMIN)
        YA(2) = initial value YA(2,TAMIN)
        ——
        YA(NE) = initial value YA(NE,TAMIN)
$ENDADD
```

2.10. Specification of the subintegral function

If $MODEL='DF', the subintegral function FA(X,YA(TA),TA) must be defined by the user either directly in the full dialogue mode, or by using corresponding macrovariables in the batch (or mixed) mode. The value of the subintegral function is specified by using the macrovariable $FMODELA:

```
$SET(FMODELA)
        FA = value FA(X,YA(TA),TA)
        (for a given vector of variables X, a given vector of state variables YA(TA)
        and a given time TA)
$ENDSET
```

The first derivatives of the subintegral function according to the variables are specified by using the macrovariable $GMODELA:

```
$SET(GMODELA)
        GA(1) = derivative ∂FA(X,YA(TA),TA)/∂X(1)
        GA(2) = derivative ∂FA(X,YA(TA),TA)/∂X(2)
        ——
        GA(NF) = derivative ∂FA(X,YA(TA),TA)/∂X(NF)
        (for a given vector of variables X, a given vector of state variables YA(TA)
        and a given time TA)
$ENDSET
```

The first derivatives of the subintegral function according to the state variables are specified by using the macrovariable $DMODELA:

```
$SET(DMODELA)
        DA(1) = derivative ∂FA(X,YA(TA),TA)/∂YA(1)
        DA(2) = derivative ∂FA(X,YA(TA),TA)/∂YA(2)
        ——
        DA(NE) = derivative ∂FA(X,YA(TA),TA)/∂YA(NE)
        (for a given vector of variables X, a given vector of state variables YA(TA)
        and a given time TA)
$ENDSET
```

If it is advantageous to compute the first derivatives of the subintegral function FA(X,YA(TA),TA) together with its value, we can replace the set of models $FMODELA, $GMODELA, $DMODELA by the common model $FGDMODELA. Partially we can replace the models $FMODELA, $GMODELA or $FMODELA, $DMODELA or $GMODELA, $DMODELA by the common models $FGMODELA or $FDMODELA or $GDMODELA, respectively.

If $MODEL='DQ' and the objective function contains an integral part, we have to set $MODELA='YES' and define the functions WE(KE;TA) and YE(KE;TA), $1 \leq KE \leq NE$, by using the macrovariable $FMODELE.

2.11. Specification of the terminal function

If $MODEL='DF', the terminal function FF(X,YA(TAMAX),TAMAX) must be defined by the user either directly in the full dialogue mode, or by using corresponding macrovariables in the batch (or mixed) mode. The value of the terminal function is specified by using the macrovariable $FMODELF:

```
$SET(FMODELF)
        FF = value FF(X,YA(TAMAX),TAMAX)
        (for a given vector of variables X, a given vector of state variables YA(TAMAX)
        and a given time TAMAX)
$ENDSET
```

The first derivatives of the terminal function according to the variables are specified by using the macrovariable $GMODELF:

```
$SET(GMODELF)
        GF(1) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂X(1)
        GF(2) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂X(2)
        ——
        GF(NF) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂X(NF)
```

(for a given vector of variables X, a given vector of state variables YA(TAMAX)
and a given time TAMAX)
$ENDSET

The first derivatives of the terminal function according to the state variables are specified by using the macrovariable $DMODELF:

$SET(DMODELF)
DF(1) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂YA(1)
DF(2) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂YA(2)
——
DF(NE) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂YA(NE)
(for a given vector of variables X, a given vector of state variables YA(TAMAX)
and a given time TAMAX)
$ENDSET

If it is advantageous to compute the first derivatives of the terminal function FF(X,YA(TAMAX),TAMAX) together with its value, we can replace the set of models $FMODELF, $GMODELF, $DMODELF by the common model $FGDMODELF. Partially we can replace the models $FMODELF, $GMODELF or $FMODELF, $DMODELF or $GMODELF, $DMODELF by the common models $FGMODELF or $FDMODELF or $GDMODELF, respectively.

If $MODEL='DQ' and the objective function contains a terminal part, we have to set $MODELF='YES' and define the coefficients EW(KE) and EY(KE), 1≤ KE≤ NE, by using the macrovariable $INPUT:

$ADD(INPUT)
EW(1) = value EW(1); EY(1) = value EY(1)
EW(2) = value EW(2); EY(2) = value EY(2)
——
EW(NE) = value EW(NE); EY(NE) = value EY(NE)
$ENDADD

2.12. Optimization with general constraints

If there are no general constraints we set $KBC=0. In the opposite case we set $KBC=1 or $KBC=2. If $KBC=1 or $KBC=2, then

FC(KC;X) - unbounded		, if IC(KC) = 0
CL(KC) ≤	FC(KC;X)	, if IC(KC) = 1
	FC(KC;X) ≤ CU(KC)	, if IC(KC) = 2
CL(KC) ≤	FC(KC;X) ≤ CU(KC)	, if IC(KC) = 3
CL(KC) =	FC(KC;X) = CU(KC)	, if IC(KC) = 5

where 1≤ KC≤ NC. The option $KBC=2 must be chosen if IC(KC)=3 for at least one index 1≤ KC≤ NC. Then two different fields XL(K) and XU(KC), 1≤ KC≤ NC are declared. In the opposite case we set $KBC=1 and only one common field XL(KC)=XU(KC), 1≤ KC≤ NC is declared. The number of constraints NC must be specified by using the statement $NC=number_of_functions.

The types of general constraints IC(KC), 1≤ KC≤ NC, and lower and upper bounds XL(KC) and XU(KC), 1≤ KC≤ NC, can be specified by using the macrovariable $INPUT. The default values are IC(KC)=3 and XL(KC)=XU(KC)=0, 1≤ KC≤ NC. For example:

$KBF=2; $NC=3

$ADD(INPUT)
IC(1)=1; CL(1)=c_1^L
IC(2)=1; CL(2)=c_2^L
IC(3)=3; CL(3)=c_3^L; CU(3)=c_3^L
$ENDADD

2.13. Specification of the constraint functions (dense problems)

The constraint functions FC(KC;X), 1≤ KC≤ NC, must be defined by the user either directly in the full dialogue mode, or by using corresponding macrovariables in the batch (or mixed) mode. The values of the constraint functions are specified by using the macrovariables $FMODELC or $FMODELCS:

$SET(FMODELC)
FC = value FC(KC;X)
(for a given index KC and given values of variables X(I), 1≤ I≤ NF)
$ENDSET

or

$SET(FMODELCS)
CF(1) = value FC(1;X)
CF(2) = value FC(2;X)
——
CF(NC) = value FC(NC;X)
$ENDSET

The first derivatives of the constraint functions are specified by using the macrovariables $GMODELC or $GMODELCS:

$SET(GMODELC)
GC(1) = derivative ∂FC(KC;X)/∂X(1)
GC(2) = derivative ∂FC(KC;X)/∂X(2)
——
GC(NF) = derivative ∂FC(KC;X)/∂X(NF)
(for a given index KC and given values of variables X(I), 1≤ I≤ NF)
$ENDSET

or

$SET(GMODELCS)
CG(1) = derivative ∂FC(1;X)/∂X(1)
CG(2) = derivative ∂FC(1;X)/∂X(2)
——
CG(NF) = derivative ∂FC(1;X)/∂X(NF)
CG(NF+1) = derivative ∂FC(2;X)/∂X(1)
CG(NF+2) = derivative ∂FC(2;X)/∂X(2)
——
CG(NC*NF) = derivative ∂FC(NC;X)/∂X(NF)
$ENDSET

The second derivatives of the constraint functions are specified by using the macrovariables $HMODELC or $HMODELCS. If $JACC='D', the Hessian matrices are assumed to be dense and we only specify their upper half:

$SET(HMODELC)

HC(1) = derivative $\partial^2 FC(KC;X)/\partial X(1)^2$

HC(2) = derivative $\partial^2 FC(KC;X)/\partial X(1)\partial X(2)$

HC(3) = derivative $\partial^2 FC(KC;X)/\partial X(2)^2$

HC(4) = derivative $\partial^2 FC(KC;X)/\partial X(1)\partial X(3)$

HC(5) = derivative $\partial^2 FC(KC;X)/\partial X(2)\partial X(3)$

HC(6) = derivative $\partial^2 FC(KC;X)/\partial X(3)^2$

——

HC(NF*(NF+1)/2) = derivative $\partial^2 FC(KC;X)/\partial X(NF)^2$

(for a given index KC and given values of variables X(I), 1≤ I≤ NF)

$ENDSET

or

$SET(HMODELCS)

CH(1) = derivative $\partial^2 FC(1;X)/\partial X(1)^2$

CH(2) = derivative $\partial^2 FC(1;X)/\partial X(1)\partial X(2)$

CH(3) = derivative $\partial^2 FC(1;X)/\partial X(2)^2$

CH(4) = derivative $\partial^2 FC(1;X)/\partial X(1)\partial X(3)$

CH(5) = derivative $\partial^2 FC(1;X)/\partial X(2)\partial X(3)$

CH(6) = derivative $\partial^2 FC(1;X)/\partial X(3)^2$

——

CH(NF*(NF+1)/2) = derivative $\partial^2 FC(1;X)/\partial X(NF)^2$

CH(NF*(NF+1)/2+1) = derivative $\partial^2 FC(2;X)/\partial X(1)^2$

CH(NF*(NF+1)/2+2) = derivative $\partial^2 FC(2;X)/\partial X(1)\partial X(2)$

CH(NF*(NF+1)/2+3) = derivative $\partial^2 FC(2;X)/\partial X(2)^2$

——

CH(NC*NF*(NF+1)/2) = derivative $\partial^2 FC(NC;X)/\partial X(NF)^2$

$ENDSET

If the macrovariables $GMODELC and $GMODELCS or $HMODELC and $HMODELCS are not defined, we suppose that the first or the second derivatives of the constraint functions are not given analytically. In this case, they are computed numerically, by using the UFO system routines whenever it is required. If it is advantageous to compute the first derivatives of the constraint functions FC(KC;X), 1≤ KC≤ NC, together with their values, we can replace the set of models $FMODELC, $GMODELC by the common model $FGMODELC and the set of models $FMODELCS, $GMODELCS by the common model $FGMODELCS. Similarly we can replace the set of models $FMODELC, $GMODELC, $HMODELC by the common model $FGHMODELC and the set of models $FMODELCS, $GMODELCS, $HMODELCS by the common model $FGHMODELCS.

To improve the efficiency of the computation, we can specify some additional information about the constraint functions FC(KC;X), 1≤ KC≤ NC. The first piece of information, useful for an automatic choice of the optimization method, is the computational complexity specified by the macrovariable $KCC:

$KCC= 1 - Evaluations of the constraint functions FC(KC;X), 1≤ KC≤ NC, are very easy (they require $O(n)$ simple operations at most).

$KCC= 2 - Evaluations of the constraint functions FC(KC;X), 1≤ KC≤ NC, are of medium complexity (they at least require $O(n)$ complicated operations and $O(n^2)$ simple operations at most).

$KCC= 3 - Evaluations of the constraint functions FC(KC;X), 1≤ KC≤ NC, are extremely difficult (they at least require $O(n^2)$ complicated or $O(n^3)$ simple operations).

The option $KCC=2 is default.

If some of the constraint functions are linear and have the form

$$\mathrm{FC(KC;X)} = \sum_{\mathrm{I=1}}^{\mathrm{NF}} \mathrm{CG((KC-1)*NF+I)*X(I)}$$

we can specify them separately. Then the number of linear constraint functions must be specified by using the statement \$NCL=number_of_linear_functions (default value is \$NCL=0). We always suppose that the first NCL constraint functions are linear. Then the coefficients CG((KC-1)*NF+I), 1$\leq$ KC$\leq$ NCL, 1$\leq$ I$\leq$ NF, are specified by using the macrovariable \$INPUT and the macrovariables \$FMODELC or \$FMODELCS, \$GMODELC or \$GMODELCS, \$HMODELC or \$HMODELCS are used only for the specification of the nonlinear constraint functions FC(KC;X), NCL<KC$\leq$ NC.

2.14. Specification of the constraint functions (sparse problems)

The UFO system contains optimization methods which take into account the sparsity pattern of the Jacobian matrix CG. This possibility decreases the computational time and storage requirements for large-scale optimization problems. In this case, we use option \$JACC='S' which means that the sparsity pattern is specified. All other specifications remain the same as in the case of dense problems. The sparsity pattern of the Jacobian matrix is specified by using the macrovariable \$INPUT. Two integer vectors ICG and JCG are used where ICG(KC), 1$\leq$ KC$\leq$ NC+1, are pointers and JCG(K), 1$\leq$ K$\leq$ ICG(NC+1)-1, are indices of nonzero elements. Nonzero elements are ordered by the gradients of the constraint functions. The number of nonzero elements must be specified by using the statement \$MC=number_of_elements. The number of nonzero elements could be greater than is needed (twice say) since it is used for the declaration of working fields. For example, if we have the gradients

$$\mathrm{GC(1;X)} = [g_{11}^C, 0\ \ , 0\ \ , g_{14}^C],$$

$$\mathrm{GC(2;X)} = [0\ \ , g_{22}^C, 0\ \ , g_{24}^C],$$

$$\mathrm{GC(3;X)} = [0\ \ , 0\ \ , g_{33}^C, 0\ \ ],$$

$$\mathrm{GC(4;X)} = [g_{41}^C, g_{42}^C, g_{43}^C, 0\ \ ],$$

$$\mathrm{GC(5;X)} = [0\ \ , 0\ \ , g_{53}^C, g_{54}^C],$$

and the Jacobian matrix

$$\mathrm{CG(X)} = \begin{pmatrix} g_{11}^C & ,0 & ,0 & ,g_{14}^C \\ 0 & ,g_{22}^C & ,0 & ,g_{24}^C \\ 0 & ,0 & ,g_{33}^C & ,0 \\ g_{41}^C & ,g_{42}^C & ,g_{43}^C & ,0 \\ 0 & ,0 & ,g_{53}^C & ,g_{54}^C \end{pmatrix}$$

then we have to set:

```
$NC=5
$MC=20 (the minimum required value is MC=10)
$ADD(INPUT)
        ICG(1)=1; ICG(2)=3; ICG(3)=5
        ICG(4)=6; ICG(5)=9; ICG(6)=11
        JCG(1)=1; JCG(2)=4; JCG(3)=2; JCG(4)=4; JCG(5)=3
        JCG(6)=1; JCG(7)=2; JCG(8)=3; JCG(9)=3; JCG(10)=4
$ENDADD
```

As in the case of the dense problem, the first derivatives of the constraint functions can be specified by using the macrovariables $GMODELC or $GMODELCS. If $JACC='S', only the nonzero elements of the gradients are specified. For the above example the specification has the form:

```
$SET(GMODELC)
      IF (KC.EQ.1) THEN
            GC(1) = ∂FC(1;X)/∂X(1)
            GC(4) = ∂FC(1;X)/∂X(4)
      ELSE IF (KC.EQ.2) THEN
            GC(2) = ∂FC(2;X)/∂X(2)
            GC(4) = ∂FC(2;X)/∂X(4)
      ELSE IF (KC.EQ.3) THEN
            GC(3) = ∂FC(3;X)/∂X(3)
      ELSE IF (KC.EQ.4) THEN
            GC(1) = ∂FC(4;X)/∂X(1)
            GC(2) = ∂FC(4;X)/∂X(2)
            GC(3) = ∂FC(4;X)/∂X(3)
      ELSE
            GC(3) = ∂FC(5;X)/∂X(3)
            GC(4) = ∂FC(5;X)/∂X(4)
      ENDIF
$ENDSET
```

or

```
$SET(GMODELCS)
      CG(1) = ∂FC(1;X)/∂X(1)
      CG(2) = ∂FC(1;X)/∂X(4)
      CG(3) = ∂FC(2;X)/∂X(2)
      CG(4) = ∂FC(2;X)/∂X(4)
      CG(5) = ∂FC(3;X)/∂X(3)
      CG(6) = ∂FC(4;X)/∂X(1)
      CG(7) = ∂FC(4;X)/∂X(2)
      CG(8) = ∂FC(4;X)/∂X(3)
      CG(9) = ∂FC(5;X)/∂X(3)
      CG(10) = ∂FC(5;X)/∂X(4)
$ENDSET
```

As in the case of the dense problem, the second derivatives of the approximating functions can be specified by using the macrovariables $HMODELC or $HMODELCS. If $JACC='S', only nonzero elements of the Hessian matrices are specified. For the above example the specifications have the form

```
$SET(HMODELC)
      IF (KC.EQ.1) THEN
            HC(1) = ∂²FC(1;X)/∂X(1)²
            HC(2) = ∂²FC(1;X)/∂X(1)∂X(4)
            HC(3) = ∂²FC(1;X)/∂X(4)²
      ELSE IF (KC.EQ.2) THEN
            HC(1) = ∂²FC(2;X)/∂X(2)²
            HC(2) = ∂²FC(2;X)/∂X(2)∂X(4)
            HC(3) = ∂²FC(2;X)/∂X(4)²
      ELSE IF (KC.EQ.3) THEN
            HC(1) = ∂²FC(3;X)/∂X(3)²
```

```
      ELSE IF (KC.EQ.4) THEN
            HC(1) = ∂²FC(4;X)/∂X(1)²
            HC(2) = ∂²FC(4;X)/∂X(1)∂X(2)
            HC(3) = ∂²FC(4;X)/∂X(2)²
            HC(4) = ∂²FC(4;X)/∂X(1)∂X(3)
            HC(5) = ∂²FC(4;X)/∂X(2)∂X(3)
            HC(6) = ∂²FC(4;X)/∂X(3)²
      ELSE
            HC(1) = ∂²FC(5;X)/∂X(3)²
            HC(2) = ∂²FC(5;X)/∂X(3)∂X(4)
            HC(3) = ∂²FC(5;X)/∂X(4)²
      ENDIF
$ENDSET
```

or

```
$SET(HMODELCS)
      CH(1) = ∂²FC(1;X)/∂X(1)²
      CH(2) = ∂²FC(1;X)/∂X(1)∂X(4)
      CH(3) = ∂²FC(1;X)/∂X(4)²
      CH(4) = ∂²FC(2;X)/∂X(2)²
      CH(5) = ∂²FC(2;X)/∂X(2)∂X(4)
      CH(6) = ∂²FC(2;X)/∂X(4)²
      CH(7) = ∂²FC(3;X)/∂X(3)²
      CH(8) = ∂²FC(4;X)/∂X(1)²
      CH(9) = ∂²FC(4;X)/∂X(1)∂X(2)
      CH(10) = ∂²FC(4;X)/∂X(2)²
      CH(11) = ∂²FC(4;X)/∂X(1)∂X(3)
      CH(12) = ∂²FC(4;X)/∂X(2)∂X(3)
      CH(13) = ∂²FC(4;X)/∂X(3)²
      CH(14) = ∂²FC(5;X)/∂X(3)²
      CH(15) = ∂²FC(5;X)/∂X(3)∂X(4)
      CH(16) = ∂²FC(5;X)/∂X(4)²
$ENDSET
```

Note that the dimensions of arrays HC or CH must be specified by the statement $MHC=dimension_of_HC or $MCH=dimension_of_CH.

If some of the constraint functions are linear (i.e. if $NCL>0) and if $JACC='S', then the coefficients CG(K), $1 \leq K \leq$ ICG(NCL+1)-1 (constant part of the sparse Jacobian matrix), must be specified by using the macrovariable $INPUT. If the matrix given in the above example is the constant sparse Jacobian matrix, we use this specification:

```
$ADD(INPUT)
      CG(1)=g11^C; CG(2)=g14^C; CG(3)=g22^C; CG(4)=g24^C
      CG(5)=g33^C; CG(6)=g41^C; CG(7)=g42^C; CG(8)=g43^C
      CG(9)=g53^C; CG(10)=g54^C
$ENDADD
```

There is another possibility which can be useful when all constraint functions are linear. It is based on the usage of a special procedure UKMCI1 which serves for a direct input of individual Jacobian matrix elements. The procedure UKMCI1 is formally called by using the statement

```
CALL $UKMCI1(K,I,GCKI)     or     $SETCG(K,I,GCKI)
```

where K is an index of a given constraint function (a row of the Jacobian matrix), I is an index of a given variable (a column of the Jacobian matrix), and GCKI is a numerical value of the element ∂FC(K;X)/∂X(I). For the example given above we can write:

$ADD(INPUT)
    $SETCG(1,1,$g^C_{11}$)
    $SETCG(1,4,$g^C_{14}$)
    $SETCG(2,2,$g^C_{22}$)
    $SETCG(2,4,$g^C_{24}$)
    $SETCG(3,3,$g^C_{33}$)
    $SETCG(4,1,$g^C_{41}$)
    $SETCG(4,2,$g^C_{42}$)
    $SETCG(4,3,$g^C_{43}$)
    $SETCG(5,3,$g^C_{53}$)
    $SETCG(5,4,$g^C_{54}$)
$ENDADD

The main advantage of the last possibility is the fact that it is not necessary to specify the fields ICG and JCG beforehand. If the number of the constraints are very large, then we can use a slightly more complicated procedure UKMCI2 which uses dynamic structures and therefore works more quickly. The procedure UKMCI2 is formally called by using the statement

CALL $UKMCI2(K,I,GCKI)

where K is an index of a given constraint function (a row of the Jacobian matrix), I is an index of a given variable (a column of the Jacobian matrix), GCKI is the numerical value of the element ∂FC(K;X)/∂X(I) and LCG is an auxiliary working field.

2.15. Additional specifications concerning optimization problems

Useful specifications, which can improve the computational efficiency and robustness of the optimization methods, are a lower bound for the objective function value and an upper bound for the stepsize. Both of these values depend on the definition of the objective function and can be specified by the statements $FMIN=lower_bound (for the objective function value) and $XMAX=upper_bound (for the stepsize). We recommend a definition of $FMIN whenever it is possible, and a definition of $XMAX whenever the objective function contains the exponential functions. If the objective function is a sum of powers (or a sum of squares), then automatically $FMIN=0. The default option for the maximum stepsize is $XMAX=1000.

If there are no general constraints and if the number of variables is not greater than 20, then we can use global optimization methods. A decision between local and global optimization is effected by means of macrovariable $EXTREM:

$EXTREM='L' - A local extremum is found, which usually contains the starting point in its region of attractivity.
$EXTREM='G' - All extrema in the given region are found and a global extremum is determined.

The default option is $EXTREM='L'. If $EXTREM='G', we cannot use the common models $FGMODELF and $FGHMODELF for a common specification of the value , the gradient and the Hessian matrix of the model function. Similarly we cannot use the models $FGMODELA or $FGMODELAS and $FGHMODELA or $FGHMODELAS for a common specification of the approximating functions.

The global optimization is performed over a bounded region specified by lower and upper bounds XL(I) and XU(I), 1$\leq$ I$\leq$ NF. If these bounds are not specified (using the macrovariable $INPUT), they are computed from the initial values of variables and from the given maximum stepsize, so that XL(I)=X(I)$-$XMAX and XU(I)=X(I)+XMAX, 1$\leq$ I$\leq$ NF. The maximum stepsize is specified, as in

the case given above, using the statement $XMAX=maximum_stepsize. The default option is again $XMAX=1000.

Additional useful specifications, concerning the solution precision, are bounds used in termination criteria. These bounds can be specified by the macrovariables $TOLX, $TOLF, $TOLB, $TOLG, $TOLC and MIC, MIT, MFV:

$TOLX - lower bound for a relative change of variables
$TOLF - lower bound for a relative change of function values
$TOLB - lower bound for the objective function value
$TOLG - lower bound for the objective function gradient norm
$TOLC - lower bound for the violated constraint functions

$MIC - maximum number of penalty function changes
$MIT - maximum number of iterations
$MFV - maximum number of function evaluations

The default values are $TOLX='1.0D-8', $TOLF='1.0D-16', $TOLB='−1.0D60', $TOLG='1.0D-6', $TOLC='1.0D-6' and MIC=5, MIT=500, MFV=1000.

3. Optimization methods in the UFO system

The UFO system has a modular structure. All optimization methods can be set up using the individual simple modules. For example, the sequential quadratic programming variable metric methods for nonlinearly constrained optimization problems are set up by using the modules for an objective function evaluation, penalty function definition, direction determination, quadratic programming solution, stepsize selection, and variable metric update. The optimization methods contained in the UFO system can be roughly divided into two groups. The first group contains methods for unconstrained and linearly constrained optimization problems, while the second group contains methods for general nonlinear programming problems. Methods for general nonlinear programming problems, i.e. for problems with nonlinear constraints, are classified by their realization form which is determined by using the macrovariable $FORM:

$FORM='SQ' - Sequential (or recursive) quadratic programming methods for general dense problems.

$FORM='SE' - Inexact sequential (or recursive) quadratic programming methods for sparse equality constrained problems.

$FORM='SI' - Inexact interior point methods for sparse equality and inequality constrained problems.

Sections 3.1 - 3.20 concern methods for unconstrained and linearly constrained problems. These methods do not use the macrovariable $FORM for a classification. Methods for general nonlinear programming problems are described in Sections 3.21 - 3.23. The basic parts of optimization methods are described in Sections 3.24 - 3.28. Section 3.29 is devoted to global optimization methods.

Methods for unconstrained and linearly constrained problems contained in the UFO system can be partitioned into several classes which are specified by using the macrovariable $CLASS:

$CLASS='HM' - Heuristic methods for small-size problems. This class contains the pattern search method and the simplex method.

$CLASS='CD' - Conjugate direction methods which use no matrices. This class contains conjugate direction methods and variable metric methods with limited storage based on the Strang recursions.

$CLASS='VM' - Variable metric methods which use an approximation of the Hessian matrix which is updated in each iteration.

$CLASS='VL' - Variable metric methods with limited storage based on compact variable metric updates.

$CLASS='VR' - Variable metric methods with limited storage based on reduced Hessians.

$CLASS='MN' - Modified Newton methods which use the Hessian matrix computed either analytically or numerically.

$CLASS='TN' - Truncated Newton methods based on the difference approximation of directional derivatives.

$CLASS='GN' - Modified Gauss-Newton methods for nonlinear least squares problems which use the normal equation matrix as an approximation of the Hessian matrix. These methods are also realized by using the Jacobian matrix representation.

$CLASS='QN' - Quasi-Newton methods for nonlinear least squares problems and nonlinear equations.

$CLASS='QL' - Quasi-Newton methods with limited storage for sparse nonlinear least squares problems and sparse nonlinear equations.

$CLASS='QB' - Quasi-Newton and Brent methods for nonlinear equations.

$CLASS='LP' - Simplex type methods for linear programming problems.

$CLASS='LI' - Interior point methods for linear programming problems.

$CLASS='QP' - Simplex type methods for quadratic programming problems.

$CLASS='QI' - Interior point methods for quadratic programming problems.

$CLASS='BM' - Proximal bundle methods for nonsmooth optimization.

$CLASS='BN' - Bundle-Newton methods for nonsmooth optimization.

$CLASS='VB' - Variable metric bundle methods for nonsmooth optimization.

The individual methods from the above classes can be chosen by using additional specifications. The most important ones, concerning direction determination and stepsize selection, are the type of the method, the kind of the matrix decomposition and the number of the method. The type of the method is specified by the macrovariable $TYPE:

$TYPE='L' - Line search methods.
$TYPE='G' - General trust region methods.
$TYPE='T' - Special trust region methods for nonlinear least squares problems.
$TYPE='M' - Modified Marquardt methods for nonlinear least squares problems.
$TYPE='F' - SQP filter methods for nonlinear programing problems.
$TYPE='P' - Pattern search method of Hooke and Jeeves.
$TYPE='S' - Simplex method of Nelder and Mead.

The kind of the matrix decomposition is specified by the macrovariable $DECOMP:

$DECOMP='M' - The symmetric matrix is used as an input for the direction determination.
$DECOMP='G' - The LDL^T decomposition without permutations is used as an input for the direction determination. This decomposition is usually obtained by the Gill-Murray algorithm [50].
$DECOMP='S' - The LDL^T decomposition with permutations is used as an input for the direction determination. This decomposition is usually obtained by the Schnabel-Eskow algorithm [130].
$DECOMP='B' - The block LDL^T decomposition with permutations is used as an input for the direction determination. This decomposition is usually obtained by the Bunch-Parlett algorithm [15].
$DECOMP='I' - The inverse of a symmetric matrix is used as an input for the direction determination.
$DECOMP='R' - the $R^T R$ decomposition without permutations is used as an input for the direction determination. This decomposition is usually obtained by the recursive QR factorization [72].
$DECOMP='C' - The $R^T R$ decomposition with permutations is used as an input for the direction determination. This decomposition is usually obtained by an application of the rank revealing algorithm [18].
$DECOMP='A' - The rectangular matrix is used as an input for the direction determination.
$DECOMP='Q' - The QR decomposition of a rectangular matrix without permutations is used as an input for the direction determination. This decomposition is usually obtained by using the Householder reflection with the explicitly stored orthogonal matrix Q.
$DECOMP='E' - The general square matrix is used as an input for the direction determination in the case NA=NF (system of nonlinear equations).

If $FORM='SE', we have additional possibilities for a representation of matrices in the direction determination:

$DECOMP='K' - The indefinite Karush-Kuhn-Tucker matrix is used as an input for the direction determination.
$DECOMP='Z' - The null space representation based on orthogonal projection is used as an input for the direction determination.
$DECOMP='G' - The range space representation based on the Schur complement is used as an input for the direction determination.

If $FORM='SI', we have the following possibility for a representation of matrices in the direction determination:

$DECOMP='I' - The interior point Karush-Kuhn-Tucker matrix is used as an input for the direction determination.

The macrovariable $DECOMP is also used for the selection of conjugate direction methods. In this case it does not concern the kind of matrix decomposition.

The serial number of the method is specified by the macrovariable $NUMBER. It determines an individual realization of the direction determination.

Additional information about specifications $TYPE, $DECOMP, $NUMBER is given in Section 3.25.

All options used for the method selection have default values, which follows from the knowledge bases coded in the individual templates. Therefore they need not be specified by the user. The possibilities we describe can be of service to users who are familiar with optimization methods.

Almost all optimization methods have different realizations for three different representations of the objective function. If $HESF='D', dense variants can be used for either unconstrained problems or box constrained problems or linearly constrained problems (with dense linear constraints specified by $JACC='D'). If $HESF='S', sparse variants can be used for either unconstrained problems or box constrained problems or linearly constrained problems (with sparse linear constraints specified by $JACC='S'). If $JACA='S' and $HESF='B', partitioned variants can be used for either unconstrained problems or box constrained problems. Partitioned variants of optimization methods are usually less efficient due to the more expensive matrix operations. Therefore we recommend preferring sparse variants to the partitioned ones.

3.1. Heuristic methods

Heuristic (or comparative) methods are specified by the statement $CLASS='HM'. These methods can only be used for small-size problems (with 10 variables at most). The main advantage of the heuristic methods is that they do not require continuity of the objective function.

The individual heuristic methods are specified by the macrovariable $TYPE:

$TYPE='P' - Pattern search method of Hooke and Jeeves [64].
$TYPE='S' - Simplex method of Nelder and Mead [114].

The default value is $TYPE='P'.

3.2. Conjugate direction methods

Conjugate direction methods are specified by the statement $CLASS='CD'. These methods are very efficient for large problems with computationally simple objective functions ($KCF=1 or $KCA=1). The main advantage of conjugate direction methods is that no matrices are used (implicitly $HESF='N'). This fact highly decreases storage requirements.

The individual conjugate direction methods are specified by the macrovariable $DECOMP:

$DECOMP='C' - Conjugate gradient methods. These methods are the simplest ones of all conjugate direction methods and they require the fewest storage requirements. However, they usually consume a greater number of function evaluations then other conjugate direction methods.

$DECOMP='V' - Variable metric methods with limited storage based on the Strang recursions. These methods allow us to prescribe storage requirements using the number of VM steps (the number of necessary used vectors is approximately twice as great a number of VM steps). The number of VM steps is specified by the macrovariable $MF. Variable metric methods with limited storage usually consume fewer function evaluations then conjugate gradient methods.

There are two families of conjugate gradient methods implemented in the UFO system:

$NUMBER=1 - Basic conjugate gradient methods described in [81]. The individual methods are specified by using the macrovariables $MET, $MET1 and $MET2.

$NUMBER=2 - Generalized conjugate gradient methods introduced in [67]. The individual methods are specified by using the macrovariable $MET1.

If $MET=0, then the steepest descent method is used. If $MET=1, the Fletcher-Reeves method [41] is used. If $MET=2, the Polak-Ribiere method [120] is used. If $MET=3, the Hestenes-Stiefel method [62] is used. The macrovariable $MET1 specifies the restart procedure as it is described in [81]. If $MET1=1, a restarted CG method with a positive parameter is used. If $MET1=2, a bounded CG method with a positive parameter is used. If $MET1=3, a bounded CG method with a positive lower bound is used. If $MET1=4, a CG method with the Powell restart is used. If $MET1=5, a CG method with the test on conjugacy is used. If $MET1=6, a CG method with the test on orthogonality is used. The macrovariable $MET2 specifies the scaling parameter as is described in [81] ($MET2=1 for suppressed scaling and $MET2=2 for scaling in each iteration).

Similarly, the UFO system contains two variable metric methods with limited storage:

$NUMBER=1 - The BFGS method with limited storage described in [115]. The default number of VM steps is $MF=5.

$NUMBER=2 - The extended BFGS method with limited storage described in [68]. The default number of VM steps is $MF=3.

Both these methods are realized by using various scaling techniques [76] specified by the macrovariable $MET1. If $MET1=1, the scaling is suppressed. If $MET1=2, the scalar scaling is used. If $MET1=3, the diagonal scaling is used. If $MET1=4, the scalar and diagonal scalings are used simultaneously.

Possible specifications (type-decomp-number) for the conjugate direction methods in the unconstrained case are these:

L-C-1, L-V-1,
L-C-2, L-V-2.

The default choice is L-C-1. Conjugate direction methods can also be used for sparse linear constraints when $JACC='S'.

3.3. Variable metric methods

Variable metric methods are specified by the statement $CLASS='VM'. These methods are most commonly used for either unconstrained or linearly constrained optimizations. Variable metric methods use a symmetric (usually positive definite) matrix which is updated in every iteration in such a way that it approximates the Hessian matrix of the objective function as precisely as possible. In the UFO system, the variable metric methods are realized in three different forms (for $HESF='D', $HESF='S' and $HESF='B') depending on the Hessian matrix specification.

There are two families of variable metric methods for dense problems ($HESF='D') which are distinguished using the macrovariable $UPDATE:

$UPDATE='B' - The Broyden family [12]. Variable metric methods from this family are the most commonly used ones since they are very robust and efficient.

$UPDATE='D' - The Davidon family [26]. Variable metric methods from this family are similar to the previous ones. The only difference is that projections into the new subspace are computed. This guarantees the quadratic termination property even in the case of an imperfect line search.

The default value is $UPDATE='B'.

Individual variable metric methods are specified by using the macrovariables $MET, $MET1, $MET2 and $MET3. The macrovariable $MET determines the variable metric update. If $MET=1, the BFGS

method [12], [36], [54], [132] is used. If $MET=2, the DFP method [25], [40] is used. If $MET=3, the Hoshino method [65] is used. If $MET=4, the safeguarded rank-one method [80] is used. If $MET=5, the optimally conditioned method [26] is used. If $MET=6, the rank-one based method [80] from the preconvex part of the Broyden family is used. If $MET=7, the variationally derived method [83] from the preconvex part of the Broyden family is used. If $MET=8, the heuristic method [86] is used. If $MET=9, the method [155] derived from the matrix decomposition is used. If $MET=10, the method [156] which minimizes the angle between the direction vector and the negative gradient is used. If $MET=11, the method [86] which minimizes the norm of the direction vector is used. If $MET=12, the least prior deviation method [108] is used. The default value is $MET=1. If we specify $DECOMP='M', we can only use the values $MET=1,2,3,4.

The macrovariable $MET1 determines the Oren (scaling) parameter [117]. If $MET1=1, no scaling is used. If $MET1=2, the initial scaling [133] is used. If $MET1=3, the controlled scaling [83] is used. If $MET1=4, the simple controlled scaling [91] is used. If $MET1=5, the scaling in each iteration is used. The default value is $MET1=3. The scaling parameter is determined by using heuristic rules given in [86].

The macrovariable $MET2 determines the value of the Biggs (nonquadratic model) parameter [4]. If $MET2=1, the unit value is used. If $MET2=2, the Spedicato value [134] is used. If $MET2=3, the modified Spedicato value [86] is used. If $MET2=4, the value determined from the homogeneous model [86] is used. If $MET2=5, the value determined from the cubic model [5] is used. The default value is $MET2=2.

The macrovariable $MET3 determines the Powell correction [124]. If $MET3=1, the Powell correction is suppressed (the strong update elimination). If $MET3=2, the Powell correction is suppressed (the weak update elimination). If $MET3=3, the Powell correction is applied. The default value is $MET3=1.

Possible specifications (type-decomposition-number) for dense variable metric methods in the unconstrained case are these:

L-G-1,	L-S-1,	L-B-1,	L-I-1,	L-M-1,
				L-M-3,
G-G-1,	G-S-1,	G-B-1,		G-M-1,
G-G-2,	G-S-2,	G-B-2,		G-M-2,
				G-M-3,
				G-M-4,
				G-M-5,
				G-M-7.

The default choice is L-I-1. In both the box constrained and the linearly constrained cases we cannot use specifications with $DECOMP='B'.

If the Hessian matrix is sparse with a general pattern ($HESF='S'), the sparse variable metric methods, which preserve this pattern, are used. If $DECOMP='M', the individual variable metric updates (or families) are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [104]. This update can only be used if $DECOMP='M'.

$UPDATE='G' - The fractioned Marwil projection update [148]. This update can only be used if $DECOMP='M' and $NUMBER=3.

$UPDATE='T' - The fractioned Toint projection update (the best method given in [148]). This update can only be used if $DECOMP='M' and $NUMBER=3.

$UPDATE='B' - The partitioned variable metric updates from the Broyden family [58]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default value is $UPDATE='M'.

Fractioned updates with specifications $UPDATE='G' or $UPDATE='T' can only be used in the unconstrained case. If $UPDATE='B', the particular update is specified by using the macrovariable $MET. If $MET=1, the BFGS method is used. If $MET=2, the DFP method is used. If $MET=3, the

Hoshino method is used. If $MET=4, the safeguarded rank-one method is used. The default value is $MET=1.

If $DECOMP='G', less efficient sparse product form updates from the Broyden family are used. In this case, the particular update is specified by using the macrovariable $MET. If $MET=1, the BFGS method is used. If $MET=2, the DFP method is used. If $MET=3, the Hoshino method is used. The default value is $MET=1.

Possible specifications (type-decomposition-number) for sparse variable metric methods in the unconstrained case are these:

L-G-1,	L-M-1,
	L-M-3,
G-G-1,	G-M-1,
	G-M-2,
	G-M-3,
	G-M-4,
	G-M-5,
	G-M-7.

The default choice is L-M-3. In the box constrained case, only the choice $DECOMP='M' is permitted. The fractioned updates ($UPDATE='T' and $UPDATE='G') can only be used if $DECOMP='M' and $NUMBER=3.

If the Hessian matrix is sparse with a partitioned pattern ($HESF='B'), only the partitioned variable metric updates, specified by the choice $UPDATE='B', can be used. These updates are the same as in the case when the Hessian matrix is sparse with a general pattern, but the partitioned realization is usually less efficient than the general one due to the more expensive matrix operations.

Possible specifications (type-decomposition-number) for partitioned variable metric methods in the unconstrained case are these:

L-M-3,
G-M-3.

The default choice is L-M-3.

3.4. Variable metric methods with limited storage based on compact variable metric updates

Variable metric methods with limited storage based on compact variable metric updates are specified by the statement $CLASS='VL'. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited storage based on compact variable metric updates use several small-size matrices which are updated in every iteration in such a way that their product approximates the Hessian matrix as precisely as possible [17].

Individual variable metric methods with limited storage are specified by using the macrovariables $MET and $MET1. The macrovariable $MET determines the variable metric update. If $MET=1, the BFGS method [12], [36], [54], [132] is used. If $MET=4, the safeguarded rank-one method [80] is used. The macrovariable $MET1 determines the scaling technique. If $MET1=1, scaling is suppressed. If $MET1=2, the scalar scaling is used.

Possible specifications (type-decomposition-number) for variable metric methods based on compact variable metric updates with limited storage are these:

L-I-1,	
	L-M-3,
	G-M-3,
	G-M-4,
	G-M-5,

3.5. Variable metric methods with limited storage based on reduced Hessians

Variable metric methods with limited storage based on reduced Hessians are specified by the statement $CLASS='VR'. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited storage based on reduced Hessians use a small-size matrix which is updated in every iteration in such a way that it approximates the reduced Hessian matrix as precisely as possible [53].

Individual variable metric methods with limited storage are specified by using the macrovariables $MET, $MET1, and $MET2. The macrovariable $MET determines the variable metric update. If $MET=1, the BFGS method [12], [36], [54], [132] is used. If $MET=2, the DFP method [25], [40] is used. If $MET=3, the Hoshino method [65] is used. If $MET=4, the safeguarded rank-one method [80] is used. If $MET=5, the optimally conditioned method [26] is used. If $MET=6, the rank-one based method [80] from the preconvex part of the Broyden family is used. If $MET=7, the variationally derived method [83] from the preconvex part of the Broyden family is used. If $MET=8, the heuristic method [86] is used. If $MET=9, the method [155] derived from the matrix decomposition is used. If $MET=10, the method [156] which minimizes the angle between the direction vector and the negative gradient is used. The default value is $MET=1.

The macrovariable $MET1 determines the Oren (scaling) parameter [117]. If $MET1=1, no scaling is used. If $MET1=2, the initial scaling [133] is used. If $MET1=3, the controlled scaling [83] is used. If $MET1=4, the simple controlled scaling [91] is used. If $MET1=5, the scaling in each iteration is used. The default value is $MET1=3. The scaling parameter is determined by using heuristic rules given in [86].

The macrovariable $MET2 determines the value of the Biggs (nonquadratic model) parameter [4]. If $MET2=1, the unit value is used. If $MET2=2, the Spedicato value [134] is used. If $MET2=3, the modified Spedicato value [86] is used. If $MET2=4, the value determined from the homogeneous model [86] is used. If $MET2=5, the value determined from the cubic model [5] is used. The default value is $MET2=2.

The macrovariable $MET3 determines the Powell correction [124]. If $MET3=1, the Powell correction is suppressed (the strong update elimination). If $MET3=2, the Powell correction is suppressed (the weak update elimination). If $MET3=3, the Powell correction is applied.

Possible specifications (type-decomposition-number) for variable metric methods with limited storage based on reduced Hessians are these:

L-R-1.

3.6. Modified Newton methods

Modified Newton methods are specified by the statement $CLASS='MN'. These methods use the Hessian matrix of the objective function which is computed either analytically or numerically. The UFO system performs a numerical computation of the Hessian matrix automatically whenever the macrovariable $HMODELF (or $FGHMODELF) is not defined. Modified Newton methods are realized in three different forms (for $HESF='D', $HESF='S' and $HESF='B') depending on the Hessian matrix specification. Even if the modified Newton methods can be realized as line search methods ($TYPE='L'), it is more advantageous to realize them as trust region methods ($TYPE='G').

If the Hessian matrix is dense ($HESF='D'), all second derivatives have to be given analytically or they are computed numerically by using differences of gradients. Possible specifications (type-decomposition-number) for dense modified Newton methods in the unconstrained case are these:

L-G-1,	L-S-1,	L-B-1,	L-M-1,
L-G-2,	L-S-2,	L-B-2,	L-M-2,
			L-M-3,
G-G-1,	G-S-1,	G-B-1,	G-M-1,
G-G-2,	G-S-2,	G-B-2,	G-M-2,
			G-M-3,
			G-M-4,
			G-M-5,
			G-M-7.

The default choice is G-M-7. In both the box constrained and the linearly constrained cases we cannot use specifications with $DECOMP='S' and $DECOMP='B'. The choice L-G-1 differs from the choice L-G-2. The last one corresponds to the combination of both the Newton and the conjugate gradient methods.

If the Hessian matrix is sparse with a general pattern ($HESF='S'), we have two possibilities. If $MODEL='FF', only the structurally nonzero second order derivatives have to be given analytically by using the prescribed pattern. The numerical computation of the second derivatives is based on the fact that a substantially lower number of differences has to be used in comparison with the dense case. The determination of suitable differences is a combinatorial problem equivalent to a graph coloring problem [19], [20]. If $MODEL='AF' or $MODEL='AQ' or $MODEL='AP', only the nonzero second derivatives of the approximating functions have to be given analytically by using the prescribed pattern. The numerical computation of the second derivatives is based on the fact that the approximating functions depend on a minor number of variables so that the number of differences is substantially lower in comparison to the dense case.

If $MODEL='AQ' (sum of squares), the combination [89] of both the modified Newton and the modified Gauss-Newton methods can be used. This choice is possible by using the macrovariable $MET. If $MET=1, the modified Newton method is used. If $MET=2, then the combined method is used. The default value is $MET=2.

Possible specifications (type-decomposition-number) for sparse modified Newton methods in the unconstrained case are these:

L-G-1,	L-M-1,
	L-M-3,
G-G-1,	G-M-1,
	G-M-2,
	G-M-3,
	G-M-4,
	G-M-5,
	G-M-7.

The default choice is G-M-3. In the box constrained case, only the choice $DECOMP='M' is permitted.

If the Hessian matrix is sparse with a partitioned pattern ($HESF='B'), a computation of the second order derivatives is the same as in the case when the Hessian matrix is sparse with a general pattern, but the partitioned realization is usually less efficient than the general one due to the more expensive matrix operations.

If $MODEL='AQ' (sum of squares), the combination of both the modified Newton and the modified Gauss-Newton methods can be used. This choice is possible by using the macrovariable $MET like the dense case. Possible specifications (type-decomposition-number) for partitioned modified Newton methods in the unconstrained case are these:

L-M-3,
G-M-3.

The default choice is G-M-3.

3.7. Truncated Newton methods

Truncated Newton methods are specified by the statement $CLASS='TN'. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. Truncated Newton methods are very efficient for large problems with computationally simple objective functions ($KCF=1 or $KCA=1). The main advantage of truncated Newton methods is that no matrices are used (implicitly $HESF='N'). This fact highly decreases storage requirements.

Truncated Newton methods are implemented either as line search methods or as trust region methods and are based on the conjugate gradient subalgorithm. Possible specifications (type-decomposition-number) for truncated Newton methods are these:

L-M-3,
G-M-3,
G-M-4,
G-M-5.

The default choice is G-M-3.

3.8. Modified Gauss-Newton methods for nonlinear least squares and nonlinear equations

Modified Gauss-Newton methods are specified by the statement $CLASS='GN'. These methods are special optimization methods for either nonlinear least squares ($MODEL='AQ') or nonlinear least powers ($MODEL='AP') problems. Modified Gauss-Newton methods are based on the fact that the first term in the Hessian matrix expression, the so-called normal equation matrix, depending on the first derivatives of the approximating functions only is a good approximation of the whole Hessian matrix. The second term in the Hessian matrix expression can be approximated by using the variable metric updates.

Modified Gauss-Newton methods are realized in four different forms (for $HESF='D', $HESF='S', $HESF='B', $HESF='N') depending on the Hessian matrix specification. Although the modified Gauss-Newton methods can be realized as the line search methods ($TYPE='L'), it is more advantageous to realize them as the trust region methods ($TYPE='G').

If the Hessian matrix is specified to be dense ($HESF='D'), then the normal equation matrix is also dense. In this case, we can use hybrid methods with dense updates:

$UPDATE='N' - No update is used. The method utilizes the normal equation matrix (the first part of the Hessian matrix expression).

$UPDATE='S' - The Dennis structured approach [30] is used. The second part of the Hessian matrix is approximated by using modified variable metric updates. This part is added to the normal equation matrix if the conditions for leaving the modified Gauss-Newton method are satisfied.

$UPDATE='F' - The Fletcher hybrid approach [3], [42] is used. The Hessian matrix is approximated either by the normal equation matrix or by the matrix obtained by using the variable metric updates. The decision between the two cases is based on the rate of the function value decrease and on the normal equation matrix conditioning.

$UPDATE='B' - A variable metric update from the Broyden class is applied either to the normal equation matrix or to the previous approximation of the Hessian matrix if conditions for leaving the modified Gauss-Newton method are satisfied [89].

The default value is $UPDATE='N'.

Individual variable metric updates from the above families are specified by using the macrovariable $MET. If $MET=1, the BFGS method is used. If $MET=2, the DFP method is used. If $MET=3, the

Hoshino method is used. If $MET=4, the original (unsafeguarded) rank-one method is used. The value $MET=4 is only allowed if $UPDATE='S' and is the default in this case. The value $MET=1 is the default in the other cases.

Variable metric updates ($UPDATE=F or $UPDATE='B') can be realized either as simple updates (normal equation matrix is updated) or as cumulative updates (previous approximation of the Hessian matrix is updated), as is described in [89]. A decision between these possibilities is mediated by the macrovariable $MOT1. If $MOT1=0, the cumulative update is used. If $MOT1=1, the simple update is used.

In the dense case, the modified Gauss-Newton methods can be realized with additional special matrix decompositions which cannot be used in other cases. If $DECOMP='R', the recursive QR decomposition [120] is used with an additional correction of the upper triangular matrix R. If $DECOMP='C', this matrix R can moreover be changed by using the rank revealing algorithm [18] which can improve its conditioning. Possible specifications (type-decomposition-number) for dense modified Gauss-Newton methods in the unconstrained case are these:

L-G-1 ,	L-S-1,	L-B-1,	L-R-1,	L-C-1,	L-M-1,
					L-M-3,
G-G-1,	G-S-1,	G-B-1,	G-R-1,	G-C-1,	G-M-1,
G-G-2,	G-S-2,	G-B-2,	G-R-2,	G-C-2,	G-M-2,
					G-M-3,
					G-M-4,
					G-M-5,
					G-M-7,
T-G-1,	T-S-1,		T-R-1,	T-C-1,	T-M-1,
T-G-2,					
	T-S-7,			T-C-7,	T-M-7,
					M-M-1.

The default choice is G-M-7. In both the box constrained and the linearly constrained cases we cannot use specifications $DECOMP='S', $DECOMP='R', $DECOMP='C'. If $DECOMP='S' or $DECOMP='C', then variable metric updates cannot be used ($UPDATE='N'). The specification $UPDATE='S' can only be used if $DECOMP='M'.

If the Hessian matrix is specified to be sparse with a general pattern ($HESF='S'), the normal equation matrix has the same structure. In this case, we can use hybrid methods with sparse updates:

$UPDATE='N' - No update is used. The method utilizes the normal equation matrix (the first part of the Hessian matrix expression).

$UPDATE='S' - The Dennis structured approach [30] is used. The second part of the Hessian matrix is approximated by using modified variable metric updates. This part is added to the normal equation matrix if conditions for leaving the modified Gauss-Newton method are satisfied.

$UPDATE='D' - The Brown-Dennis structured approach [14] is used. The Hessian matrices of approximating functions are approximated by using variable metric updates. These matrices serve for approximating the second part of the Hessian matrix which is added to the normal equation matrix if conditions for leaving the modified Gauss-Newton method are satisfied.

$UPDATE='B' - A variable metric update from the Broyden class is applied either to the normal equation matrix or to the previous approximation of the Hessian matrix if conditions for leaving the modified Gauss-Newton method are satisfied [89].

$UPDATE='M' - A sparse update based on the Marwil projection is applied either to the normal equation matrix or to the previous approximation of the Hessian matrix if conditions for leaving the modified Gauss-Newton method are satisfied [89].

The default value is $UPDATE='N'.

Individual variable metric updates from the above families are specified by using the macrovariable $MET as in the dense case. The value $MET=4 is only allowed if either $UPDATE='S' or $UPDATE='D', and is the default in this case. The value $MET=1 is the default in the other cases excepting the case $UPDATE='M' in which the macrovariable $MET is not utilized.

Variable metric updates ($UPDATE=M or $UPDATE='B') can be realized either as simple updates (normal equation matrix is updated) or as cumulative updates (previous approximation of the Hessian matrix is updated). A decision between these possibilities is mediated by the macrovariable $MOT1 similarly as in the dense case.

If $UPDATE='D', we can use several switches for utilizing variable metric updates specified by the macrovariable $MOT2. If $MOT2=0, the Fletcher and Xu switch [42] is used. If $MOT2=1, a modification of the Fletcher and Xu switch is used. If $MOT2=2, the Denis and Welsch switch [33] is used. If $MOT2=3, the Ramsin and Wedin switch [126] is used. The default value is $MOT2=0.

Possible specifications (type-decomposition-number) for sparse Gauss-Newton methods in the unconstrained case are these:

L-G-1,	L-M-1,
	L-M-3,
G-G-1,	G-M-1,
G-G-2,	G-M-2,
	G-M-3,
	G-M-4,
	G-M-5,
	G-M-7,
T-G-1,	T-M-1,
	T-M-7,
	M-M-1.

The default choice is G-M-3. In the box constrained case, only the choice $DECOMP='M' is permitted.

If the Hessian matrix is specified to be sparse with a partitioned pattern ($HESF='B'), the normal equation matrix has the same structure. If that is the case, then we can use hybrid methods with partitioned updates $UPDATE='N', $UPDATE='S', $UPDATE='D', $UPDATE='F', $UPDATE='B', whose details have already been explained above. Note that the partitioned realization is usually less efficient than the general one due to the more expensive matrix operations.

Possible specifications (type-decomposition-number) for partitioned Gauss-Newton methods are these:

L-M-3,
G-M-3.

The default choice is G-M-3.

If the Hessian matrix is not specified ($HESF='N'), the normal equation matrix is not used. The Jacobian matrix, defining a linear least squares problem, is utilized in each iteration instead. Such so-called normal equation free Gauss-Newton methods are realized in two different forms (for $JACA='D' and $JACA='S') depending on the Jacobian matrix specification.

If the Jacobian matrix is specified to be dense ($JACA='D'), then we cannot use hybrid methods with variable metric updates (only the specification $UPDATE='NO is permitted). Moreover, dense, normal equation free Gauss-Newton methods can only be used in the unconstrained case.

Possible specifications (type-decomposition-number) for dense, normal equation free, Gauss-Newton methods are these:

L-Q-1,	L-A-1,	L-E-1,
	L-A-3,	L-E-3,
	L-A-4,	L-E-4,
		L-E-5,
G-Q-1,	G-A-1,	G-E-1,
G-Q-2,		G-E-2,
	G-A-3,	G-E-3,
	G-A-4,	G-E-4,
		G-E-5,
	G-A-7.	

The default choice is G-A-3 for least squares problems and G-E-3 for systems of nonlinear equations. The specification $DECOMP='E' can only be used if NA=NF (system of nonlinear equations).

If the Jacobian matrix is specified to be sparse ($JACA='S'), we can use hybrid methods with simple variable metric updates:

$UPDATE='N' - No update is used. The method utilizes the original Jacobian matrix.
$UPDATE='V' - The simple factorized BFGS update [89] is used. The second order information is approximated by the unsymmetric rank-one update of the Jacobian matrix.
$UPDATE='R' - The simple factorized rank-one update [89] is used. The second order information is approximated by the addition of a dense row to the Jacobian matrix.

If $UPDATE='V' or $UPDATE='R', we can use several switches for utilizing variable metric updates, specified by the macrovariable $MOT2 as in the case of the specification $HESF='S' described above. The default value is $MOT2=0.

The main advantage of sparse, normal equation free, Gauss-Newton methods consists in the fact that the normal equation matrix is dense if the sparse Jacobian matrix has at least one dense row. If this is the case, then the classical Gauss-Newton methods cannot be used. On the other hand, the normal equation matrix often has a lower number of nonzero elements than the Jacobian one. Consequently, the classical Gauss-Newton methods are more efficient in this case.

Possible specifications (type-decomposition-number) for sparse, normal equation free, Gauss-Newton methods are these:

L-A-1,	L-E-1,
L-A-3,	L-E-3,
L-A-4,	L-E-4,
	L-E-5,
G-A-1,	G-E-1,
	G-E-2,
G-A-3,	G-E-3,
G-A-4,	G-E-4,
	G-E-5,
G-A-7.	

The default choice is G-A-3 for least squares problems and G-E-3 for systems of nonlinear equations. The specification $DECOMP='E' can only be used if NA=NF (system of nonlinear equations). The choice L-E-1 differs from the choice L-E-2. The last one corresponds to the incomplete LU decomposition.

3.9. Quasi-Newton methods for nonlinear least squares and nonlinear equations

Quasi-Newton methods are specified by the statement $CLASS='QN'. These methods are special optimization methods for nonlinear least squares ($MODEL='AQ') problems including systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Quasi-Newton methods use a rectangular matrix which is updated in every

iteration in such a way that it approximates the Jacobian matrix as precisely as possible. In the UFO system, the quasi-Newton methods are realized in two different forms (for $JACA='D' and $JACA='S') depending on the Jacobian matrix specification.

There are two possibilities for dense problems ($JACA='D') which are distinguished by using the macrovariable $UPDATE:

$UPDATE='N' - No update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='B' - The Broyden family [13] of rank-one updates is used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

When $UPDATE='B', the individual quasi-Newton methods are specified by using the macrovariable $MET. If $MET=1, the first (good) Broyden update [13] is used. If $MET=2, the second Broyden update [13] is used. If $MET=3, the second Greenstadt update [136] is used. If $MET=4, the first Greenstadt update [136] is used. If $MET=5, the first Todd OC update [69] is used. If $MET=6, the first Todd OCX update [69] is used. If $MET=7, the second Todd OC update [69] is used. If $MET=8, the second Todd OCX update [69] is used. The default value is $MET=1. Dense quasi-Newton methods can only be used in the unconstrained case.

Possible specifications (type-decomposition-number) for dense quasi-Newton methods are these:

L-Q-1,	L-A-1,	L-E-1,
	L-A-3,	L-E-3,
	L-A-4,	L-E-4,
		L-E-5,
G-Q-1,	G-A-1,	G-E-1,
G-Q-2,		G-E-2,
	G-A-3,	G-E-3,
	G-A-4,	G-E-4,
		G-E-5,
	G-A-7.	

The default choice is G-Q-3. The specification $DECOMP='E' can only be used if NA=NF (system of nonlinear equations).

If the Jacobian matrix is sparse with a general pattern ($JACA='S'), there are two possibilities for computing an approximation of the Jacobian matrix by the differences. These possibilities are distinguished by using the macrovariable $NUMDER:

$NUMDER=1 - Derivatives of individual approximating functions are computed.

$NUMDER=2 - The Coleman-More [21] graph coloring algorithm is used.

Moreover, various sparse quasi-Newton updates which preserve the pattern of the Jacobian matrix can be used.

If $NUMDER=1, there are three choices of the quasi-Newton updates which are specified by the macrovariable $UPDATE:

$UPDATE='N' - No update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='B' - Sparse quasi-Newton updates are used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

$UPDATE='S' - Modified Newton methods such as the row scaling update are used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

If $NUMDER=2, there are four choices of the quasi-Newton updates which are specified by the macrovariable $UPDATE:

$UPDATE='N' - No update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='B' - Sparse quasi-Newton updates [131] are used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

$UPDATE='S' - Modified Newton methods such as the row scaling update are used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

$UPDATE='C' - Cyclic column determination methods are used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

When $UPDATE='B', the individual quasi-Newton methods are specified by using the macrovariable $MET. If $MET=1, the Schubert update [131] is used. If $MET=2, the Bogle-Perkins update [10] is used. If $MET=3, the column update [105] is used. When $UPDATE='S' and $MET=0, the modified Newton method is used. When $UPDATE='S' and $MET=1, the row scaling update [105] is used. When $UPDATE='C' and $MET=0, the cyclic column determination method [74] is used. When $UPDATE='S' and $MET=1, the cyclic column determination method [74] is used followed by the Schubert update [131].

Possible specifications (type-decomposition-number) for sparse quasi-Newton methods are these:

L-A-1,	L-E-1,
L-A-3,	L-E-3,
L-A-4,	L-E-4,
	L-E-5,
G-A-1,	G-E-1,
	G-E-2,
G-A-3,	G-E-3,
G-A-4,	G-E-4,
	G-E-5,
G-A-7.	

The default choice is G-A-3 for the least squares problems and G-E-3 for systems of nonlinear equations. The specification $DECOMP='E' can only be used if NA=NF (system of nonlinear equations). The choice L-E-1 differs from the choice L-E-2. The latter corresponds to the incomplete LU decomposition.

3.10. Quasi-Newton methods with limited storage for nonlinear equations

Quasi-Newton methods with limited storage are specified by the statement $CLASS='QL'. The number of QN steps is specified by the macrovariable $MF (the default value is $MF=5). These methods are special methods for solving sparse systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore only the case NA=NF is permitted. Quasi-Newton methods with limited storage use an initial approximation of the sparse Jacobian matrix together with several small-size matrices which are updated in every iteration in such a way that their product approximates the Jacobian matrix as precisely as possible [17]. There are two possibilities which are distinguished by using the macrovariable $UPDATE:

$UPDATE='N' - No update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='B' - The Broyden good update of rank-one with limited storage [17] is used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

Possible specifications (type-decomposition-number) for quasi-Newton methods with limited storage are these:

L-A-3,	L-E-3,
L-A-4,	L-E-4,
	L-E-5,
G-A-3,	G-E-3,
G-A-4,	G-E-4,
	G-E-5,

The default choice is G-E-3.

Besides the quasi-Newton methods with limited storage, this class contains inverse column scaling methods which are chosen by using the specification $DECOMP='I'. There are two possibilities which are distinguished by using the macrovariable $UPDATE:

$UPDATE='N' - No update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='B' - The inverse column scaling update [106] is used in almost all iterations. Only after the restart is the Jacobian matrix approximated numerically by using differences.

Possible specifications (type-decomposition-number) for inverse column scaling methods are these:

L-I-1,
L-I-3.

If $NUMBER=1, then a complete LU decomposition is used. If $NUMBER=3, then a combination of direct and iterative methods is used. The default value is $NUMBER=3.

3.11. Truncated Newton methods for nonlinear equations

Truncated Newton methods are specified by the statement $CLASS='TN'. These methods are special methods for solving systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore only the case NA=NF is permitted. Truncated Newton methods differ from quasi-Newton methods in that the sparse Jacobian matrix multiplication is replaced by the numerical differentiation. These methods are very efficient for large problems with computationally simple functions in nonlinear equations ($KCA=1). The main advantage of the truncated Newton methods is that matrices are not used (implicitly $JACA='N'). This fact highly decreases storage requirements.

Truncated Newton methods are implemented either as the line search methods or as the trust region methods and are based on the smoothed CGS subalgorithm. This subalgorithm can be preconditioned by using the tridiagonal decomposition. This possibility is determined by the macrovariable $MOS2. If $MOS2=0, the tridiagonal decomposition is not used. If $MOS2=1, the tridiagonal decomposition is used before the iterative process. If $MOS2=2, the tridiagonal decomposition is used as a preconditioner. If $MOS2=3, both previous cases are assumed. The default value is $MOS2=0.

Possible specifications (type-decomposition-number) for truncated Newton methods are these:

L-E-3,
L-E-4,
L-E-5,
G-E-3,
G-E-4,
G-E-5.

The default choice is G-E-3.

3.12. Quasi-Newton and Brent methods for nonlinear equations

Quasi-Newton and Brent methods are specified by the statement $CLASS='QB'. These methods are special simple methods for solving dense systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore, only the case NA=NF is permitted. Individual methods are selected using the macrovariable $NUMBER:

$NUMBER=1 - The Brent method described in [?].
$NUMBER=3 - The simple Newton method (this method can also be used if the macrovariable $GMODELA is defined).

The default value is $NUMBER=3.

3.13. Simplex type methods for linear programming problems

Simplex type methods for linear programming problems are specified by the statement $CLASS='LP'. These methods are realized in two different forms (for $JACC='D' and $JACC='S') depending on the constraint Jacobian matrix specification.

If the constraint Jacobian matrix is dense ($JACC='D'), we can use two different linear programming methods based on the active set strategy:

$NUMBER=1 - Primal reduced gradient (null-space) method (like the method proposed in [49]), which is a special implementation of the steepest descent reduced gradient method.
$NUMBER=2 - Primal projected gradient (range-space) method which is a special implementation of the steepest descent projected gradient method.

Possible specifications (type-number) for dense linear programming methods are L-1 and L-2. The default choice is L-1.

If the constraint Jacobian matrix is sparse ($JACC='S'), we can use two different linear programming methods based on the active set strategy: one linear programming method :

$NUMBER=1 - Primal reduced gradient (null-space) simplex type method which is described in [147].

$NUMBER=2 - Primal projected steepest descent (range-space) method.

A possible specification (type-number) for sparse linear programming methods are L-1 and L-2. The default choice is L-2.

3.14. Interior point methods for linear programming problems

Interior point methods for linear programming problems are specified by using the statement $CLASS='LI'. These methods, based on an infeasible primal-dual predictor-corrector strategy, can be used only in the sparse case when $JACC='S'. Moreover, only the standard LP constraints $Ax = b$, $x \geq 0$ can be considered at present. Individual methods are chosen by using the macrovariable $MLP:

$MLP=1 - The first algorithm of Miao [107].
$MLP=1 - The second algorithm of Miao [107].
$MLP=3 - The Mizuno algorithm [109].

All these methods can be realized in three forms depending on the way of solving the linear generalized Karush-Kuhn-Tucker system:

$NUMBER=1 - Direct solution based on the Gill-Murray decomposition applied to the Schur complement.

$NUMBER=2 - Direct solution based on the Bunch-Parlett decomposition applied to the original Karush-Kuhn-Tucker system.

$NUMBER=3 - Iterative solution based on the conjugate gradient method applied to the Schur complement.

Possible specifications (type-number) for interior point methods are L-1, L-2 and L-3. The default choice is L-1.

3.15. Simplex type methods for quadratic programming problems

Simplex type methods for quadratic programming problems are specified by using the statement $CLASS='QP'. These methods are realized in two different forms (for $JACC='D' and $JACC='S') depending on the constraint Jacobian matrix specification.

If the constraint Jacobian matrix is dense ($JACC='D'), we can use three different quadratic programming methods based on the active set strategy:

$NUMBER=1 - Primal reduced gradient (null-space) method (like the method proposed in [51]) which is a special implementation of the Newton reduced gradient method.

$NUMBER=2 - Primal projected gradient (range-space) method (like the method proposed in [38]) which is a special implementation of the Newton projected gradient method.

$NUMBER=3 - Dual projected gradient (range-space) method (like the method proposed in [55]).

Possible specifications (type-number) for dense quadratic programming methods are L-1, L-2, and L-3. The default choice is L-1.

If the constraint Jacobian matrix is sparse ($JACC='S'), we can use two different quadratic programming methods based on the active set strategy:

$NUMBER=1 - Primal reduced gradient (null-space) simplex type method which is described in [147].

$NUMBER=2 - Primal projected conjugate gradient (range-space) method.

A possible specification (type-number) for sparse quadratic programming methods are L-1 and L-2. The default choice is L-2.

3.16. Interior point methods for quadratic programming problems

Interior point methods for quadratic programming problems are specified by using the statement $CLASS='QI'. These primal-dual methods, based on the logarithmic barrier function and iterative solution of the indefinite Karush-Kuhn-Tucker system, can be used only in the sparse case when $JACC='S'. Interior point methods for quadratic programming problems are in fact the same as methods with the choices $TYPE='L' and $DECOMP='I' described in Section 3.23.

Two realizations are possible, which are specified by the macrovariable $NUMBER:

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [35] of the indefinite Karush-Kuhn-Tucker system is used.

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3. The macrovariable $MOS1 specifies the precision control. If $MOS1=0, the precision control is suppressed. If $MOS1=1, a precision guaranteeing descent direction is used together with the basic choice of the penalty parameter. The default value is $MOS1=0. The macrovariable $MOS2 specifies a preconditioning technique. If $MOS2=0, preconditioning is suppressed. If ABS($MOS2)=1, the indefinite preconditioner [96] based on a diagonal approximation of the Hessian matrix is used in the normal equation form. If $MOS2 is positive, a complete Gill-Murray decomposition is used. If $MOS2 is negative, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1. The macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method. If $MOS3=0, the residual smoothing is suppressed. If $MOS3=1, a simple one-dimensional residual smoothing is used. The default value is $MOS3=0.

The default value is $NUMBER='3'.

Possible specifications (type-decomposition-number) for inexact recursive quadratic programming methods for equality constrained nonlinear programming problems are L-1 and L-3. The default choice is L-3.

3.17. Proximal bundle methods for nonsmooth optimization

Proximal bundle methods for nonsmooth optimization problems are specified by the statement $CLASS ='BM'. These methods use a solution of the special quadratic programming subproblem derived from the cutting plane approach [152]. This subproblem is in fact the same as in the recursive quadratic programming methods for minimax problems. Proximal bundle methods are realized only for unconstrained or linearly constrained dense problems ($JACA='D'). The special quadratic programming subproblem can be solved by using the following methods:

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [77].
$NUMBER=2 - Primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

The special quadratic programming subproblem is defined in such a way that it has a diagonal Hessian matrix. There are several methods for computing the diagonal weight coefficients, which are selected by using the macrovariables $MOS and $MES2. If $MOS=1 and $MES2=1, the weights are updated using curvature of the one-dimensional quadratic function. If $MOS=1 and $MES2=2, the weights are updated using the minimum position estimate (suitable for polyhedral and nearly polyhedral functions). If $MOS=2, the weights are updated using the quasi-Newton condition.

Proximal bundle methods are only realized as line search methods in two modifications which are specified by the macrovariable $MEX. If $MEX=0, a convex version is assumed. If $MEX=1, a nonconvex version is assumed and we can define a measure of nonconvexity using the macrovariable $ETA5. The default value is $ETA5=0.25. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard, which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Proximal bundle methods are sensitive to the values of the above two parameters. Therefore, they should be carefully tuned.

Possible specifications (type-number) for proximal bundle methods are L-1 and L-2. The default choice is L-1. Proximal bundle methods can be used when $KSF=3 or $KSA=3. They can also be used for minimax problems as is shown in Section 3.20.

3.18. Bundle-Newton methods for nonsmooth optimization

Bundle-Newton methods for nonsmooth optimization problems are specified by the statement $CLASS ='BN'. These methods use a solution of the special quadratic programming subproblem derived from the cutting plane approach which contains second order information [94]. This subproblem is in fact the same as in recursive quadratic programming methods for minimax problems. Bundle-Newton methods are only realized for unconstrained or linearly constrained dense problems ($JACA='D'). The special quadratic programming subproblem can be solved by using the following methods:

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [77].
$NUMBER=2 - Primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

The special quadratic programming subproblem has a general (dense) Hessian matrix which is a bundle approximation of the second-order matrix of the original nonsmooth problem.

Bundle-Newton methods are only realized as line search methods. A nonconvex version is assumed and we can define a measure of nonconvexity using the macrovariable $ETA5. The default value is $ETA5=0.25. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard, which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Proximal bundle methods are sensitive to the values of the above two parameters. Therefore, they should be carefully tuned.

Possible specifications (type-number) for bundle-Newton methods are L-1 and L-2. The default choice is L-1. Bundle-Newton methods can be used when $KSF=3 or $KSA=3. They can also be used for minimax problems as is shown in Section 3.20.

3.19. Variable metric bundle methods for nonsmooth optimization

Variable metric bundle methods for nonsmooth optimization problems are specified by the statement $CLASS ='VB'. These methods are based on a special realization of the BFGS variable metric method. This realization uses special null steps and restarts. Stepsize selection is based on the polyhedral approximation obtained using bundles of points and subgradients. Variable metric bundle methods are realized only for unconstrained or linearly constrained dense problems ($JACA='D'). They need not solve any quadratic programming subproblem.

Variable metric bundle methods are only realized as line search methods in two modifications which are specified by the macrovariable $MEX. If $MEX=0, a convex version [97] is assumed. If $MEX=1, a nonconvex version [153] is assumed and we can define a measure of nonconvexity using the macrovariable $ETA5. The default value is $ETA5=0.25. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard, which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Variable metric bundle methods are sensitive to the values of the above two parameters. Therefore, they should be carefully tuned.

Possible specifications (type-number) for variable metric bundle methods are L-1 and L-2. The default choice is L-1. Variable metric bundle methods can be used when $KSF=3 or $KSA=3. They can also be used for minimax problems as is shown in Section 3.20.

3.20. Methods for minimax problems.

Minimax problems are specified by the choice $MODEL='AM'. These problems can be solved using six classes of methods:

$CLASS='BM' - Proximal bundle methods.
$CLASS='BN' - Bundle-Newton methods.
$CLASS='VB' - Variable metric bundle methods.

$CLASS='VM' - Recursive quadratic programming variable metric methods [100]. An approximation of the Lagrangian function Hessian matrix is updated in each iteration using the variable metric updates belonging to the Broyden family.

$CLASS='MN' - Recursive quadratic programming modified Newton methods. The Lagrangian function Hessian matrix is computed in each iteration either analytically or numerically.

$CLASS='LP' - Recursive linear programming methods.

The default value is $CLASS='VM'.

Even if the minimax problems can be solved by using bundle methods described in Sections 3.17 - 3.19, it is more efficient to use the recursive quadratic programming methods that utilize a special structure of the minimax problem. Recursive quadratic programming methods are realized in three different forms:

$TYPE='L' - Line search methods.

$TYPE='G' - General trust region methods.

$TYPE='C' - General trust region methods with second order corrections [43].

If $TYPE='L', the special line search method ($MES=5), described in [78], can be used.

The special quadratic programming subproblem, which is derived from the minimax problem, can be solved by using two different methods:

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [77].

$NUMBER=2 - Primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

Recursive quadratic programming variable metric methods use the same updates as methods with the choices $DECOMP='G' and $UPDATE='B' described in Section 3.3 (values $MET=1 - $MET=12 can be used). Similarly, recursive quadratic programming modified Newton methods correspond to the methods with the choice $DECOMP='G' described in Section 3.6 (the Gill-Murray decomposition is used).

Recursive linear programming methods are realized as trust region methods with box constrained subproblems. The special linear programming subproblem, which is derived from the minimax problem, is solved by a primal projected gradient (range-space) method which is a special implementation of the steepest descent method.

All of the above methods are only realized for dense unconstrained or linearly constrained problems. A possible specifications (type-number) for recursive quadratic programming methods are these:

L-1,
L-2,
G-1,
G-2,
C-1,
C-2.

The default choice is L-1. A possible specification (type-number) for recursive linear programming methods is G-1.

3.21. Recursive quadratic programming methods for dense general nonlinear programming problems

Recursive quadratic programming methods for dense general nonlinear programming problems are specified by the statement $FORM='SQ'. These methods belong to the two following classes:

$CLASS='VM' - Recursive quadratic programming variable metric methods. An approximation of the Lagrangian function Hessian matrix is updated in each iteration using variable metric updates.

$CLASS='MN' - Recursive quadratic programming modified Newton methods. The Lagrangian function Hessian matrix is computed in each iteration either analytically or numerically.

The default value is $CLASS='VM'. Variable metric methods are the same as in Section 3.3 with the choice $DECOMP='G' and $UPDATE='B' (values $MET=1 - $MET=12 can be used). Similarly, modified Newton methods are the same as in Section 3.6 with the choice $DECOMP='G' (the Gill-Murray decomposition is used).

Recursive quadratic programming methods for dense general nonlinear programming problems are realized as line search methods ($TYPE='L') with the l_1-exact penalty function. They are like the methods proposed in [124]. The special line search method ($MES=5) for l_1-exact penalty function can be used successfully. The quadratic programming subproblem can be solved by using two different methods:

$NUMBER=1 - Dual projected gradient (range-space) method (like the method proposed in [55]).
$NUMBER=2 - Primal projected gradient (range-space) method (like the method proposed in [38]) which is a special implementation of the Newton projected gradient method.

Possible specifications (type-number) for these methods are L-1 and L-2. The default choice is L-1.

3.22. Recursive quadratic programming methods for sparse equality constrained nonlinear programming problems

Recursive quadratic programming methods for sparse equality constrained nonlinear programming problems are specified by the statement $FORM='SE'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Recursive quadratic programming variable metric methods. An approximation of the Lagrangian function Hessian matrix is updated in each iteration using variable metric updates.
$CLASS='VL' - Recursive quadratic programming variable metric methods with limited storage based on compact representations of variable metric updates. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited storage use several small-size matrices which are updated in every iteration in such a way that their product approximates the Lagrangian function Hessian matrix as precisely as possible [17].
$CLASS='MN' - Inexact recursive quadratic programming modified Newton methods. The Lagrangian function Hessian matrix is computed in each iteration either analytically or numerically.

The default value is $CLASS='MN'.

If $CLASS='VM', the individual variable metric updates (or families) are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [104].
$UPDATE='B' - The partitioned variable metric updates from the Broyden family [58]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default value is $UPDATE='M'. If $UPDATE='B', the particular update is specified by using the macrovariable $MET. If $MET=1, the BFGS method is used. If $MET=2, the DFP method is used. If $MET=3, the Hoshino method is used. If $MET=4, the safeguarded rank-one method is used. The default value is $MET=1.

If $CLASS='VL', two variable metric updates with limited storage, belonging to the Broyden family, can be used. These updates are specified by using the macrovariable $MET. If $MET=1, then the

BFGS method is used. If $MET=4, then the safeguarded rank-one method is used. The default value is $MET=1.

Recursive quadratic programming methods for sparse equality constrained nonlinear programming problems are realized in three different ways which are specified by using the macrovariable $TYPE:

$TYPE='L' - Line search methods. These methods can use five different merit functions for the stepsize selection. Individual merit functions are determined by using the macrovariable $MEP. If $MEP=0, no merit function is used. If $MEP=1, the Powell l_1 exact penalty function is used. If $MEP=2, the l_2 augmented Lagrangian function is used. If $MEP=3, the l_1 augmented Lagrangian function is used. If $MEP=4, the Han l_1 exact penalty function is used. If $MEP=5, the Schittkowski augmented Lagrangian function is used. The default value is $MEP=2.

$TYPE='F' - SQP filter methods [44]. These methods are based on a special multicriterial decision and do not use any merit function.

$TYPE='G' - Trust region methods. These methods use two direction determination subproblems [34], [71], [98]. The vertical subproblem, solved by using the dog-leg method, serves for a sufficient decrease of constraint violations. The horizontal subproblem, solved by a special realization of the conjugate gradient method, serves for minimization of a quadratic approximation of a particular merit function. Individual merit functions are determined by using the macrovariable $MEP. If $MEP=0, no merit function is used. If $MEP=1, the Powell l_1 exact penalty function is used. If $MEP=2, the l_2 augmented Lagrangian function is used. If $MEP=3, the l_1 augmented Lagrangian function is used. If $MEP=4, the Han l_1 exact penalty function is used. If $MEP=5, the Schittkowski augmented Lagrangian function is used. The default value is $MEP=2.

The default value is $TYPE='L'.

If $TYPE='L' or $TYPE='F', the direction vector can be computed in three different ways, which are specified by using the macrovariable $DECOMP:

$DECOMP='K' - The direction vector is determined as a solution of the indefinite Karush-Kuhn-Tucker system [96].

$DECOMP='Z' - The direction vector is decomposed into two parts. The vertical part is computed directly from the constraint violation. The horizontal part, lying in the null-space, is computed iteratively by using a special realization of the conjugate gradient method. Instead of projecting into the null-space, either the augmented system or an orthogonal projection matrix, both determined from a range-space basis, are used [66].

$DECOMP='G' - The direction vector is determined directly from the Lagrangian multipliers, which are determined iteratively by using the conjugate gradient method in the range space using the Schur complement.

The default value is $DECOMP='K'.

If $DECOMP='K', five realizations are possible, which are specified by the macrovariable $NUMBER:

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [35] of the indefinite Karush-Kuhn-Tucker system is used.

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2, $MOS3 and $MOS4. The macrovariable $MOS1 specifies the precision control. If $MOS1=0, the precision control is suppressed. If $MOS1=1, a precision guaranteeing descent direction is used together with the basic choice of the penalty parameter. The default value is $MOS1=0. The macrovariable $MOS2 specifies a preconditioning technique. If $MOS2=0, preconditioning is suppressed. If ABS($MOS2)=1, the indefinite preconditioner [96] based on a diagonal approximation of the Hessian matrix is used in the normal equation form. If ABS($MOS2)=2, the indefinite preconditioner [96] based on a diagonal approximation of the Hessian matrix is used in the augmented system form. If ABS($MOS2)=3, the indefinite preconditioner [96] based on a diagonal perturbation of the Schur complement is used. If $MOS2 is positive, a complete Gill-Murray decomposition is used. If $MOS2 is negative, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1. The macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method. If $MOS3=0, the residual smoothing is suppressed. If $MOS3=1, a simple one-dimensional residual smoothing is used. The default value is $MOS3=0. The macrovariable $MOS4 specifies the choice of the initial direction. If $MOS4=0, the zero initial direction is used. If $MOS4=1, the vertical initial direction is used. The default value is $MOS4=0.

$NUMBER=4 - An inexact preconditioned conjugate residual (PCR) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact preconditioned conjugate residual method depends on specifications given by the macrovariables $MOS1 and $MOS2, which have the same meaning as in case $NUMBER=3. The default values are $MOS1=0 and $MOS2=1.

$NUMBER=5 - An inexact symmetric preconditioned quasi-minimum residual (PQMR) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact symmetric preconditioned quasi-minimum residual method depends on specifications given by the macrovariables $MOS1 and $MOS2, which have the same meaning as in case $NUMBER=3. The default values are $MOS1=0 and $MOS2=1.

$NUMBER=6 - An inexact nonsymmetric preconditioned conjugate gradient squared (PCGS) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact nonsymmetric preconditioned conjugate gradient squared method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3, which have the same meaning as in case $NUMBER=3. The default values are $MOS1=0, $MOS2=1 and $MOS3=0.

The default value is $NUMBER='3'.

If $DECOMP='Z', only one realization is possible, which is specified by the macrovariable $NUMBER:

$NUMBER=3 - An inexact null-space preconditioned conjugate gradient (NPCG) method for the determination of the horizontal direction is applied which uses a special determination of the required precision. A particular realization of the null-space preconditioned conjugate gradient method depends on the specifications given by the macrovariables $MOS1 and $MOS2. The macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter. If $MOS1=0, the precision control is suppressed. If $MOS1=1, a precision guaranteeing descent direction is used together with the basic choice of the penalty parameter. If $MOS1=2, a precision guaranteeing descent direction is used together with an extended choice of the penalty parameter, based on the condition of positive definitness. The default value is $MOS1=0. The macrovariable $MOS2 specifies a way for computing the preconditioner. If ABS($MOS2)=1, the preconditioner is computed by using the orthogonal projection matrix determined from a range-space basis. If ABS($MOS2)=2, the preconditioner is computed by using the augmented system determined from a range-space basis. If $MOS2 is positive, a diagonal approximation of the Hessian matrix is used. If $MOS2 is negative the unit approximation of the Hessian matrix is used. The default value is $MOS2=1.

If $DECOMP='G', two realizations are possible, which are specified by the macrovariable $NUMBER:

$NUMBER=3 - The sparse Gill-Murray decomposition of the Lagrangian function Hessian matrix followed by a range-space smoothed conjugate gradient (RSCG) method for a positive definite range space system is applied which uses a special determination of the required precision. The particular realization of the preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3. The macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter. If $MOS1=0, the precision control is suppressed. If $MOS1=1, a precision guaranteeing descent direction is used. The default value is $MOS1=1. The macrovariable $MOS2 specifies a preconditioning technique. If $MOS2=0, the preconditioning is suppressed. If ABS($MOS2)=1, the positive definite preconditioner [96] based on a diagonal approximation of the Hessian matrix is used. If ABS($MOS2)=2, the polynomial preconditioner [113] based on a decomposition of the normal equation is used. If $MOS2 is positive, a complete Gill-Murray decomposition is used. If $MOS2 is negative, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1. The macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method. If $MOS3=0, the residual smoothing is suppressed. If $MOS3=1, then a simple one-dimensional residual smoothing is used. The default value is $MOS3=1.

$NUMBER=4 - The sparse Bunch-Parlett decomposition of the Lagrangian function Hessian matrix followed by a range-space smoothed conjugate gradient (RSCG) method for an indefinite range space system is applied which uses a special determination of the precision required. The particular realization of the smoothed conjugate gradient method depends on specifications given by the macrovariables $MOS1 and $MOS3, which have the same meaning as in case $NUMBER=3. The default values are $MOS1=1 and $MOS3=1.

The default value is $NUMBER='3'.

If $TYPE='G', only the specifications $DECOMP='Z' and $NUMBER='3' are possible, which corresponds to the trust region conjugate gradient (TRCG) method. The macrovariable $MOS2 specifies a way for computing the projection step. If $MOS2=1, the projection step is computed by using the orthogonal projection matrix determined from a range-space basis. If $MOS2=2, the projection step is computed by using the augmented system determined from a range-space basis.

If $TYPE='L', the UFO system allows us to choose a second order correction for overcoming the Maratos effect and various Lagrange multipliers updates. This is affected by the macrovariables $MEP1

and $MEP2. The macrovariable $MEP1 specifies a second order correction. If $MEP1=1, the second order correction is suppressed. If $MEP1=2, the second order correction is determined as being a least squares solution of the shifted constraint system. The default value is $MEP1=1. The macrovariable $MEP2 specifies estimates of Lagrange multipliers at the beginning of each iteration. If $MEP2=1, the initial estimate is taken from the previous iteration. If $MEP2=2, the initial estimate is determined as being a least squares solution of the first part of the Karush-Kuhn-Tucker system. The default value is $MEP2=1.

Possible specifications (type-decomposition-number) for inexact recursive quadratic programming methods for equality constrained nonlinear programming problems are these:

L-K-1,		
L-K-3,	L-Z-3,	L-G-3,
L-K-4,		L-G-4,
L-K-5,		
L-K-6,		
F-K-1,		
	F-Z-2,	
F-K-3,	F-Z-3,	F-G-3,
F-K-4,		F-G-4,
F-K-5,		
F-K-6,		
	G-Z-3.	

The default choice is L-K-3. The choice $DECOMP='G' cannot be used for variable metric methods with limited storage ($CLASS='VL').

3.23. Interior point methods for sparse equality and inequality constrained nonlinear programming problems

Interior point methods for sparse equality and inequality constrained nonlinear programming problems are specified by the statement $FORM='SI'. These methods, which are intended for large problems, belong to the following class:

$CLASS='MN' - Inexact interior point modified Newton methods. The Lagrangian function Hessian matrix is computed in each iteration either analytically or numerically.

Interior point methods for sparse equality and inequality constrained nonlinear programming problems are realized in two different ways which are specified by using the macrovariable $TYPE:

$TYPE='L' - Line search methods. These methods can use five different merit functions for the stepsize selection. Individual merit functions are determined by using the macrovariable $MEP. If $MEP=0, no merit function is used. If $MEP=2, the l_2 augmented Lagrangian function is used. The default value is $MEP=0.

$TYPE='F' - SQP filter methods [44]. These methods are based on a special multicriterial decision and do not use any merit function.

The default value is $TYPE='L'.

The direction vector can be computed in the way, which is specified by using the macrovariable $DECOMP:

$DECOMP='I' - The direction vector is determined as a solution of the indefinite Karush-Kuhn-Tucker system [96].

Two realizations are possible, which are specified by the macrovariable $NUMBER:

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [35] of the indefinite Karush-Kuhn-Tucker system is used.

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3. The macrovariable $MOS1 specifies the precision control. If $MOS1=0, the precision control is suppressed. If $MOS1=1, a precision guaranteeing descent direction is used together with the basic choice of the penalty parameter. The default value is $MOS1=0. The macrovariable $MOS2 specifies a preconditioning technique. If $MOS2=0, preconditioning is suppressed. If ABS($MOS2)=1, the indefinite preconditioner [96] based on a diagonal approximation of the Hessian matrix is used in the normal equation form. If $MOS2 is positive, a complete Gill-Murray decomposition is used. If $MOS2 is negative, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1. The macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method. If $MOS3=0, the residual smoothing is suppressed. If $MOS3=1, a simple one-dimensional residual smoothing is used. The default value is $MOS3=0.

The default value is $NUMBER='3'.

Possible specifications (type-decomposition-number) for inexact recursive quadratic programming methods for equality constrained nonlinear programming problems are these:

L-I-1,
L-I-3.

The default choice is L-I-3.

3.24. Methods for initial value problems for ordinary differential equations

Methods for initial value problems for ordinary differential equations are specified by using the macrovariable $SOLVER. The UFO system contains five types of integration methods:

$SOLVER='DP5' - The Dormand and Prince method of the fifth order with a stepsize control for nonstiff problems.

$SOLVER='DP8' - The Dormand and Prince method of the eighth order with a stepsize control for nonstiff problems.

$SOLVER='EX1' - The extrapolation method with a stepsize control, based on the midpoint rule, for nonstiff problems.

$SOLVER='RD5' - The Radau method of the fifth order with a stepsize control for stiff problems.

$SOLVER='RS4' - The Rosenbrock method of the fourth order with a stepsize control for stiff problems.

The default value is $SOLVER='DP8'. These methods, described in [60], use a stepsize control based on a local truncation error.

A solution to the initial value problem for ordinary differential equations can be stored for subsequent processing. The extent of the data stored is determined by using the macrovariable $MED. If $MED=0, then no data are stored. If $MED=1, the data in all solution steps are stored. If $MED=2, the data in equidistant mesh points are stored. The number of mesh points is specified by using the statement $NA=number_of_mesh_points in the last case.

3.25. Methods for direction determination

Optimization methods, contained in the UFO system, are usually implemented in such a way that they use the same modules for direction determination. These modules, realized with different kinds of matrix decomposition, are distinguished by using the macrovariables $TYPE and $NUMBER. The

meaning of the specification $TYPE has been explained above. Now we will explain the specification $NUMBER.

If $TYPE='L', then line search methods are supposed. In this case, relatively simple procedures are used for direction determination. There are five possibilities:

$NUMBER=1 - Direct methods for solving linear systems based on various matrix decompositions. These decompositions are interesting, especially in the sparse case. The Gill-Murray decomposition [50] of the Hessian matrix is applied if $DECOMP='M' and $MOS2=0 or if $DECOMP='G'. The Schnabel-Eskow decomposition [130] of the Hessian matrix is used if $DECOMP='M' and $MOS2=1 or if $DECOMP='S'. The Choleski decomposition of the Hessian matrix is utilized if $DECOMP='R' or $DECOMP='C'. The Bunch-Parlett decomposition [15] of the Hessian matrix is applied if $DECOMP='B'. The inverse matrix is used if $DECOMP='I'. The orthogonal QR decomposition [149] of the Jacobian matrix is utilized if $DECOMP='A' or $DECOMP='Q'. The complete LU decomposition [27] of the Jacobian matrix is applied if $DECOMP='E'. Moreover, symbolic decomposition is always determined before the iterative process in the sparse case, so that only numerical computations with known factors are carried out in the subsequent iterations.

$NUMBER=2 - An alternative possibility to the previous case. The direct solution is combined with a conjugate gradient direction if the Hessian matrix is indefinite. This possibility can be advantageously used in connection with the modified Newton method.

$NUMBER=3 - Inexact iterative methods. The conjugate gradient method [29] for solving linear systems with the Hessian matrix is applied if $DECOMP='M'. The CGLS method [119] for solving linear least squares problems with the Jacobian matrix is used if $DECOMP='A'. The smoothed CGS method [145] for solving linear systems with the Jacobian matrix is utilized if $DECOMP='E'. The precision is specified by the macrovariable $MOS. If $MOS=1, simple strategy is used. If $MOS=2, the geometric decreasing strategy is used. If $MOS=3, the harmonic decreasing strategy is used. If $DECOMP='M' and $HESF='S', the conjugate gradient method can be preconditioned by using the incomplete Gill-Murray (IGM) decomposition. This possibility is specified by the macrovariable $MOS2. If $MOS2=0, preconditioning is suppressed. If $MOS2=1, the IGM decomposition is used. Similarly, if $DECOMP='E' and $JACA='S', the smoothed CGS method can be preconditioned by using either the incomplete LU (ILU) decomposition or the SSOR iteration. This possibility is specified by the macrovariable $MOS2. If $MOS2=0, preconditioning is suppressed. If $MOS2=1, the ILU decomposition is used. If $MOS2=2, the SSOR iteration is used.

$NUMBER=4 - Inexact iterative methods. The LSQR method [119] for solving linear least squares problems with the Jacobian matrix is applied if $DECOMP='A'. The GMRES method [129] for solving linear systems with the Jacobian matrix is used if $DECOMP='E'. The precision is specified by the macrovariable $MOS as in the previous case.

$NUMBER=5 - Inexact iterative methods. The smoothed BICGSTAB method [151] for solving linear systems with the sparse Jacobian matrix is used if $DECOMP='E'. The precision is specified by the macrovariable $MOS as in the previous case.

If the line search method is used then a descent property of the determined direction is tested. If

$$-s^T g \geq \varepsilon_0 \parallel s \parallel \parallel g \parallel$$

where $s^T g$ is the directional derivative, s is the direction, and g is the objective function gradient, then the direction is accepted. In the opposite case the optimization method is restarted. The value ε_0 is specified using the macrovariable $EPS0.

If $TYPE='G', then trust region methods are supposed. The initial trust region radius can be specified by the statement $XDEL=trust_region_radius, but the default automatically derived value is recommended. The trust region methods can be internally scaled. This way is very advantageous for nonlinear regression problems containing exponentials. The trust region scaling is specified by the macrovariable $MOS1. If $MOS1=1, no scaling is performed. If $MOS1=2, the scaling coefficients are derived from the normal equation matrix diagonal elements [85]. There are six possibilities:

$NUMBER=1 - So-called single dog-leg methods based on various matrix decompositions. These decompositions are interesting especially in the sparse case. The Gill-Murray decomposition [50] of the Hessian matrix is applied if $DECOMP='M' and $MOS2=0 or if $DECOMP='G'. The Schnabel-Eskow decomposition [130] is used if $DECOMP='M' and $MOS2=1 or if $DECOMP='S'. The Choleski decomposition of the Hessian matrix is utilized if $DECOMP='R' or $DECOMP='C'. The Bunch-Parlett decomposition [15] of the Hessian matrix is applied if $DECOMP='B'. The orthogonal QR decomposition [149] of the Jacobian matrix is utilized if $DECOMP='A' or $DECOMP='Q'. The complete LU decomposition [27] of the Jacobian matrix is applied if $DECOMP='E'. Moreover, symbolic decomposition is always determined before the iterative process in the sparse case, so that only numerical computations with known factors are carried out in the subsequent iterations. The individual dog-leg methods are specified by the macrovariable $MOS. If $MOS=1, the single dog-leg method [121] is used. If $MOS=2, the double dog-leg method [31] is used. If $MOS=3, the triple dog-leg method is used. If $MOS=4, the optimum dog-leg method [16] is used.

$NUMBER=2 - An alternative possibility to the previous case. The so-called multiple dog-leg methods (combinations of single dog-leg methods and conjugate gradient methods) [88] are supposed. The number of dog-leg steps is specified by the statement $MOS=number_of_steps.

$NUMBER=3 - Iterative trust region methods. The conjugate gradient trust region method [138] with the Hessian matrix is applied if $DECOMP='M'. The CGLS trust region method [84] with the Jacobian matrix is used if $DECOMP='A'. The smoothed CGS trust region method [93] with the Jacobian matrix is utilized if $DECOMP='E'. The precision is specified by the macrovariable $MOS. If $MOS=1, the simple strategy is used. If $MOS=2, the geometric decreasing strategy is used. If $MOS=3, the harmonic decreasing strategy is used. If $DECOMP='M' and $HESF='S', the conjugate gradient method can be preconditioned by using the incomplete Gill-Murray (IGM) decomposition. This possibility is specified by the macrovariable $MOS2. If $MOS2=0, preconditioning is suppressed. If $MOS2=1, the IGM decomposition is used. Similarly, if $DECOMP='E' and $JACA='S', the smoothed CGS method can be preconditioned by using either the incomplete LU (ILU) decomposition or the SSOR iteration. This possibility is specified by the macrovariable $MOS2. If $MOS2=0, preconditioning is suppressed. If $MOS2=1, the ILU decomposition is used. If $MOS2=2, the SSOR iteration is used.

$NUMBER=4 - Iterative trust region methods. The combined Lanczos and CG trust region method [88] with the Hessian matrix is applied if $DECOMP='M'. The LSQR trust region method [84] with the Jacobian matrix is used if $DECOMP='A'. The GMRES trust region method [93] with the Jacobian matrix is utilized if $DECOMP='E'. The precision is specified by the macrovariable $MOS as in the previous case. Iterative methods can be again preconditioned. This possibility is specified by the macrovariable $MOS2 as in the previous case.

$NUMBER=5 - Iterative trust region methods. The combined CG and Lanczos trust region method [88] with the Hessian matrix is applied if $DECOMP='M'. The smoothed BICGSTAB trust region method [93] with the Jacobian matrix is utilized if $DECOMP='E'. The precision is specified by the macrovariable $MOS as in the previous case. Iterative methods can be again preconditioned. This possibility is specified by the macrovariable $MOS2 as in the previous case.

$NUMBER=7 - An optimum locally constrained trust region method [112]. The Gill-Murray decomposition [50] of the Hessian matrix is applied if $DECOMP='M' and $MOS2=0. The Schnabel-Eskow decomposition [130] of the Hessian matrix is used if $DECOMP='M' and $MOS2=1 or if $DECOMP='S'. The special augmented Jacobian matrix is used if $DECOMP='A'.

If $TYPE='T', only the specifications $NUMBER=1, $NUMBER=2 and $NUMBER=7 can be used. These specifications have the same meaning as in the case $TYPE='G', but the implementation is simpler. If $NUMBER=7, the simplified optimum locally constrained trust region method [85] is used.

If $TYPE='M', only the specification $NUMBER=1 can be used. In this case a modified Marquardt method proposed by Fletcher [37] is applied.

3.26. Methods for stepsize selection

Stepsize selection is a very important part of optimization methods. The UFO system contains two types of stepsize selection procedures: line search methods and trust region methods. Line search methods are realized in two modifications specified by the macrovariable $SEARCH:

$SEARCH='B' - Basic line search methods based on various interpolation and extrapolation formulas.

$SEARCH='M' - Mixed line search methods which control the maximum stepsize like the trust region methods.

The choice of individual line search procedures is influenced by the order of directional derivatives being used. This order can be specified by the macrovariable $KDS. The value of the macrovariable $KDS is usually derived internally from the order of analytically supplied partial derivatives. If this order is zero, then always $KDS=0. In the opposite case, the value of the macrovariable $KDS can be specified by the user. If $KDS=0, only the function values are used during the line search. If $KDS=1, the function values and the first directional derivatives are used. If $KDS=2 then, in addition, the Hessian matrices or their approximations are computed during the line search (this case is very useful for a line search implementation of modified Gauss-Newton methods).

The particular interpolation and extrapolation rule is specified by the macrovariable $MES. If $KDS=0, we have the following possibilities:

$MES=1 - The uniformly increasing extrapolation or bisection interpolation is used.

$MES=2 - Two point quadratic extrapolation or interpolation is used.

$MES=3 - Three point quadratic extrapolation or interpolation is used.

$MES=4 - Three point cubic extrapolation or interpolation is used.

$MES=5 - Special extrapolation or interpolation is used based on the special form of the objective function.

If $KDS=1 or $KDS=2, the following possibilities, based on the first directional derivatives, can be used:

$MES=1 - The uniformly increasing extrapolation or bisection interpolation is used.

$MES=2 - Quadratic extrapolation or interpolation (with one directional derivative) is used.

$MES=3 - Quadratic extrapolation or interpolation (with two directional derivatives) is used.

$MES=4 - Cubic extrapolation or interpolation [25] is used.

$MES=5 - Conic extrapolation or interpolation [6] is used.

More detailed specifications concerning the line search selection can be chosen using macrovariables $MES1, $MES2, $MES3:

$MES1=1	- Constant extrapolation is used.
$MES1=2	- Extrapolation specified by the macrovariable $MES is used.
$MES1=3	- Extrapolation is suppressed.
$MES2=1	- Standard line search termination criterion is used.
$MES2=2	- Special termination criterion for nonconvex functions is used.
$MES2=3	- Line search is terminated after at least two function evaluations.
$MES3=1	- Safeguard against rounding errors is suppressed.
$MES3=2	- The first level of safeguard is used.
$MES3=3	- The second level of safeguard is used.

Another useful specification for the line search selection is a termination criterion which is determined by using the macrovariable $KTERS:

$KTERS<0	- The nonmonotone line search procedure proposed in [59] is used. The absolute value of the macrovariable $KTERS, which cannot be greater then 10, gives the number of nonmonotone steps.
$KTERS=1	- Perfect stepsize. The relative precision of the stepsize parameter is given by the value $EPS3.
$KTERS=2	- The Goldstein stepsize [56]. The termination precision is given by the value $EPS1.
$KTERS=3	- The Curry-Altman stepsize [24] (Wolfe conditions). The termination precision is given by the values $EPS1 and $EPS2.
$KTRES=4	- The extended Curry-Altman stepsize [39] (strict Wolfe conditions). The termination precision is given by the values $EPS1 and $EPS2.
$KTERS=5	- The Armijo stepsize [2]. The termination is given by the value $EPS1.
$KTERS=6	- The first stepsize. The stepsize selection is terminated after the first function evaluation.

The last useful specification for the line search methods is the initial stepsize choice which is determined by the macrovariable $INITS. The initial stepsize is usually computed by the rule

$$\alpha \ = \min(c_1, -c_2(\Delta F / s^T g))$$

where $s^T g$ is the initial directional derivative and $\Delta F = F - F_{min}$ or $\Delta F = F_{old} - F$ if the value of the macrovariable $INITS is positive or negative respectively. The absolute value of the macrovariable $INITS determines coefficients c_1 and c_2 If |INITS|=1, then $c_1 = 1$ and $c_2 = 0$. If |INITS|=2, then $c_1 = 1$ and $c_2 = 4$. If |INITS|=3, then $c_1 = 1$ and $c_2 = 2$. If |INITS|=4, then $c_1 = 0$ and $c_2 = 2$.

Trust region methods are also realized in two modifications specified by the macrovariable $SEARCH:

$SEARCH='B'	- The basic trust region methods with stepsize control based on the comparison of both the actual and the predicted function decreases.
$SEARCH='M'	- Mixed trust region methods which use interpolation formulas for stepsize reduction like the line search methods [116].

Trust region methods are also influenced by using the macrovariable $KTERS. If $KTERS<0, then nonmonotone trust region procedure proposed in [28] is used. The absolute value of the macrovariable $KTERS, which cannot be greater then 10, gives the number of nonmonotone steps.

3.27. Methods for numerical differentiation

The UFO system computes derivatives of the model function (of the approximating functions, of the constraint funcions) numerically whenever they are not given analytically. This is made possible by the macroprocessor which generates a corresponding part of the control program. The main problem of a

numerical differentiation is a difference determination which has to be chosen in such a way that the total influence of both the cancellation and the roundoff error is as small as possible. There are three possibilities in the UFO system which are distinguished by using the macrovatiable $MCG:

$MCG=0 - The simple difference determination described in [32] is used.
$MCG=1 - The optimum difference determination proposed in [52] is used.
$MCG=2 - The optimum difference determination proposed in [140] is used.

The default option is $MCG=2. The above possibilities are used for a computation of the model function first order derivatives. The others (second order derivatives or derivatives of the approximating functions and constraint functions) are always computed with the simple difference determination.

3.28. Methods for objective function evaluation in the case of dynamical systems optimization

If either $MODEL='DF' or $MODEL='DQ', the objective function is computed from the solution of an initial value problem for ordinary differential equations. The initial value problem is solved and the integral criterion is evaluated by using integration methods specified by the macrovariable $SOLVER as is described above. If the partial derivatives of all the functions used are given analytically, the gradient of the objective function is computed by integration methods. There are two possibilities specified by the macrovariable $SYSTEM:

$SYSTEM='F' - Forward integration using an augmented system of ordinary differential equations.
$SYSTEM='B' - Backward integration using the adjoint system of ordinary differential equations.

The default value is $SYSTEM='F'. In the case of modified Gauss-Newton methods ($CLASS='GN'), an approximation of the Hessian matrix is also computed by using forward integration of an augmented system.

3.29. Global optimization methods

Global optimization methods are used if $EXTREM='G' is specified. The global optimization methods use local optimization methods for finding local minima. Therefore the particular local optimization method has to be chosen by using the macrovariables $CLASS and $TYPE and others. Individual global optimization methods are specified by using the macrovariables $GCLASS and $GTYPE. The UFO system contains four classes of global optimization methods:

$GCLASS=1 - Random search methods. These methods are simple and robust, but less efficient.
$GCLASS=2 - Continuation methods. These methods use some penalty functions which are adjusted after reaching an arbitrary local minimum so that another local minimum is found.
$GCLASS=3 - Clustering methods. These methods are based on randomly generated sample points which are processed using clustering algorithms to determine attractivity regions (clusters) of the individual minima. The attractivity regions (clusters) obtained are not searched repeatedly.
$GCLASS=4 - Multi-level methods. Modern stochastic methods which involve a combination of sampling and local search techniques. These methods combine strong theoretical properties with an attractive computational behaviour. These methods are simpler but more efficient than the clustering methods.

If $GCLASS=1, we can choose four types of global optimization methods:

$GTYPE=1 - Single-start methods. Random points, uniformly distributed in a given region, are generated and a local minimization method is started from the point with the lowest function value.

$GTYPE=2 - Multi-start methods. Random points, uniformly distributed in a given region, are generated and local minimization is started from every point. The local minima obtained are compared and selected.

$GTYPE=3 - Modified multi-start methods. Random points, distributed in a given region uniformly, are generated and local minimization is started whenever a point is found which has a lower function value than that reached up to date.

$GTYPE=4 - Bayesian reduced multi-start methods [7]. Random samples of points are repeatedly generated. Every random sample is reduced and local minimization is started from all points belonging to the reduced sample. Obtained local minima are compared and selected. This process is repeated while the Bayesian termination criterion is not satisfied.

If $GCLASS=2, then we can choose three types of global optimization methods:

$GTYPE=1 - Tunneling function methods [73]. These methods consist of two phases: a local minimization phase and a tunneling phase. The starting point for the second phase is the local minimum. At the end of the tunneling phase a new point is found which has a function value equal or lower than the starting point.

$GTYPE=2 - Combined tunneling function and random search methods. In this case a random search is used in the tunneling phase if the minimization of a tunneling function has failed to find a new starting point.

$GTYPE=3 - Filled function methods [46], [47]. The idea of filled function methods is based on a filled function. This function has a maximum in the point of a known minimum of the objective function. On the other hand, this function does not have minimizers or saddle points in any basin of a higher minimizer of the objective function, but it does have a minimizer or a saddle point in a basin of a lower minimizer of the objective function.

If $GCLASS=3, we can choose two types of global optimization methods:

$GTYPE=1 - Density clustering method [8]. Density clustering refers to a class of clustering techniques by using nonparametric probability density estimates to form clusters. All unclustered points from a reduced sample, which are within the threshold distance from the seed point, are added to the cluster.

$GTYPE=2 - Single linkage clustering method [8]. In this case, the next two clusters to be merged are those for which the distance between the nearest points is the smallest. When this distance becomes larger than the threshold distance, the procedure is stopped. Starting with each point in a separate cluster, the points at distances smaller than the threshold distance are linked. A cluster is recognized as a set of points linked together.

If $GCLASS=4, we can choose three types of global optimization methods:

$GTYPE=1 - multi-level single linkage method [128]. In this case, the function values of the sample points are used in a very simple manner to obtain a very powerful method. The local search procedure is applied to every sample point, except if there is another sample point within the critical distance which has a smaller function value. Clusters can be constructed by associating a point with a local minimum, if there exists a chain of points linking it to that minimum. This is done so that the distance between each successive pair is, at most, equal to the critical distance and the function value is decreasing along the chain. A point in this way could be assigned to more than one minimum.

$GTYPE=2 - Multi-level mode analysis method [128]. This method is a generalization of the mode analysis method. The region is partitioned into cells. After the sample reduction, it is determined which cells contain enough points to be "full". For each full cell the function value of the cell is defined to be equal to the smallest function value of any of the sample points in the cell. Finally, for every full cell, local minimization is applied except if a cell has a neighbouring cell which is full and has a smaller function value.

$GTYPE=3 - Modified multi-level single linkage method. This is a multi level single linkage method with some modifications which are described in [128].

The number of points randomly generated in the given region can be specified by using the macrovariable $MNRND. The default value is usually 100+20*NF. Since it depends on the number of variables and for NF>20 it is too large, we recommend to use global optimization methods up to 20 variables only. If we use clustering or multi level single linkage methods ($GCLASS=3 or $GCLASS=4), we can specify additional parameters.

$MNLMIN - Maximum considered number of local minima. The default value is 50+20*NF.

$GAMA - Reduction of random sample (typically 0.1D0 - 0.2D0). A greater value of GAMA usually leads to a greater number of local minima, but it requires a greater amount of work.

$SIGMA - Parameter of cluster or single linkage termination (typically 1 - 8).

4. Input possibilities in the UFO system

The UFO system has many input possibilities including interactive dialogues. These input possibilities can be divided into three basic groups which are batch mode, text dialogue mode and graphic dialogue mode. Batch and dialogue modes can be combined. The basic means for the batch and combined modes is the UFO control language.

4.1. The UFO control language

The form of the control program can be determined by using the statements of the UFO control language. The UFO control language is based on the batch editing language (BEL) [141] that described in Appendix B. The UFO control language contains four types of instructions:

1. Standard Fortran 77 instructions which can be written in the free format.
2. Fortran 77 instructions containing macrovariables. These instructions get a final form after the first pass of the UFO preprocessor.
3. Substitutions and directives. These macroinstructions control the UFO preprocessor execution.
4. Special substitutions. These macroinstructions are special tools of the UFO control language that realize the most useful sets of single instructions.

Standard Fortran 77 instructions used in the UFO control language have some extensions and limitations. The main extension is the free format. The instructions may not have a limited length, they can be written everywhere in the input file and if they are written in the same line, the character ';' is used to separate the instructions. The continuation of an instruction is specified by character '& '. The main limitation concerns the application of instructions in the control program. Therefore, statement numbers greater than 9999 cannot be used, comments can be introduced by character '$*$' only and the only continuation character can be '& '. Also, it is recommended to use identifiers beginning with character 'W' which are not used in the UFO system

Macrovariables used in the UFO system begin with character '$' and are supposed to be of the type character. Their values are always in the form of a string of characters which can be sometimes interpreted as an integer or a real or a logical constant. The chief significance of the macrovariables is their use in substituting their values for their names in the Fortran 77 statements. In this case we place the macrovariable (beginning with '$') in the text, but if it is followed by a letter or digit we have to use brackets. For example if we write

```
$FLOAT W(100)
```

or

```
CALL UD$HESF$TYPE$DECOMP$NUMBER
```

or

```
X(1)=1.0$(P)0
```

and if the values of $FLOAT, $HESF, $TYPE, $DECOMP, $NUMBER and $P are 'REAL*8' (this is default), 'D', 'L', 'G' '1' and 'D' (this is default), we get REAL*8 W(100) or CALL UDDLG1, or X(1)=1.0D0 respectively, after the UFO preprocessor application. The values of macrovariables can be defined and changed by assignments or by special directives as will be shown later.

Substitutions and directives are very important for the UFO control language since they make the substitutions of texts, definitions and changes of macrovariables, branching, loops, etc., possible. We briefly describe the most useful of them. A more detailed description is given in Appendix B.

1. Assignment: The assignment of a string of characters for a macrovariable is specified by the macroinstruction $MACRO='value'. For example, we have to set $HESF='D', $TYPE='L', $DECOMP='G', $NUMBER=1 (the integers do not need to be substituted as strings) to obtain the result given above.

2. Insertion of a text: If we write

```
$SET(MACRO)     or     $ADD(MACRO)
   text                      text
$ENDSET                  $ENDADD
```

   then a given text (that can contain a large number of Fortran 77 statements) is inserted into the macrovariable $MACRO. The macroinstruction $SET is used for the definition of a new macrovariable. The macroinstruction $ADD appends a new text into the old macrovariable so that it can be used repeatedly.

3. Logical substitutions: The macrovariables $INT, $REAL, $LOG and $DEF have logical values. If we write $INT(MACRO) (or $REAL(MACRO) or $LOG(MACRO)), the resulting value is either .TRUE., if the value of the macrovariable $MACRO is an integer constant (or real constant or logical constant), or .FALSE. in the opposite case. If we write $DEF(MACRO), the value of $DEF is either .TRUE., if the macrovariable $MACRO was previously defined (by the substitution $MACRO='value' or by using macroinstructions $SET and $ADD), or .FALSE. in the opposite case. This possibility can be used for branching. If we use the directive $ERASE(MACRO), the previously defined macrovariable $MACRO becomes undefined (so that $DEF(MACRO)=.FALSE.).

4. List of items macrovariables: Values of macrovariables can be lists of items, i.e. they can have a more complicated form $MACRO='item_1 \item_2\...\item_n' where every item corresponds to one value. The list of items macrovariables use pointers which point out the current items. The current item can be obtained by the substitution $DATA(MACRO) which also moves the pointer to the next item. The directive $RESTORE(MACRO) returns the pointer to the first item.

5. Branching: This possibility is very similar to the branching in the Fortran 77 language:

```
$IF(condition)
   statements
$ELSEIF(condition)
   statements
$ELSE
   statements
$ENDIF
```

   Conditions can be logical constants .TRUE., .FALSE., or logical macrovariables $INT(MACRO), $REAL(MACRO), $LOG(MACRO)), $DEF(MACRO), or they can have a form of comparisons MACRO=MACRO1, MACRO='value' etc. (besides the relation =, we can also use other relations < or > or <= or >= or <>). Branching is used in the UFO preprocessor stage and has an influence on the form of the control program.

6. Loops: The basic looping directives have the following form (similarly as in the Fortran 77 or Pascal languages):

```
$DO(MACRO=INDEX1,INDEX2,INDEX3)
   statements
$ENDDO
```

   or

```
$REPEAT
   statements
$UNTIL(condition)
```

For example if we set $NF=2, $NC=3 and write

```
$DO(I=1,NF,1)
  $DO(J=1,NC,1)
    CALL $SETCG($I,$J,$I.0D0+$J.0D0)
  $ENDDO
$ENDDO
```

then the UFO preprocessor generates the sequence

```
$SETCG(1,1,1.0D0+1.0D0)
$SETCG(1,2,1.0D0+2.0D0)
$SETCG(1,3,1.0D0+3.0D0)
$SETCG(2,1,2.0D0+1.0D0)
$SETCG(2,2,2.0D0+2.0D0)
$SETCG(2,3,2.0D0+3.0D0)
```

Similarly, if we set $FLOAT='REAL*8' $N=20, $MACRO='X($N)\G($N)\H($N,$N)\.END.', and write

```
$REPEAT
   $I='DATA(MACRO)'
   $FLOAT $I
$UNTIL(I='.END.')
```

then the UFO preprocessor generates the sequence

```
REAL*8 X(20)
REAL*8 G(20)
REAL*8 H(20,20)
```

7. File substitutions: Suppose we have a file with a name file_ name.extension. Then we can include it into the control program by using the macroinstructions

```
$INCLUDE('file_name.extension')
```

or

```
$SUBST('file_name.extension')
```

The main difference between these possibilities is that the directive $INCLUDE includes a text without change (it has to be a regular Fortran 77 text with a fixed format) while the directive $SUBST substitutes a text executed consecutively by the UFO preprocessor (so that it can contain the macrovariables and macroinstructions and be written in the free format). Moreover, the directives $SUBST can be nested. This possibility is widely used for control program generation by using nested templates. If the included file has the name file_ name.I, we can use a simpler form without extension. For example, the file UZLINS.I can be substituted by using the macroinstruction $SUBST('UZLINS').

8. Special substitutions: Besides macroinstructions of the batch editing language BEL, the UFO control language contains special substitutions which realize sets of instructions and are useful for controlling the UFO preprocessor:

$BATCH	- Switch to the batch mode.
$DIALOGUE	- Switch to the default dialogue mode (text or graphic).
$TDIALOGUE	- Switch to the text dialogue mode.
$GDIALOGUE	- Switch to the graphic dialogue mode.
$GLOBAL	- Global declarations.
$INITIATION	- Initiation of the global variables.
$INPUT	- User supplied input.
$OUTPUT	- User supplied output.
$METHOD	- Generation of the optimization method.
$MODERASE	- Cancellation of the current model.
$METERASE	- Cancellation of the current method.
$VARERASE	- Clearing the common variables.
$TSTART	- Start of the time measurement.
$TSTOP	- Termination of the time measurement and print of the measured time.
$END	- End of the optimization block.
$STANDARD	- Standard optimization block: The macroinstruction $STANDARD substitutes the sequence of macroinstructions $GLOBAL, $INITIATION, $MODERASE, $INPUT, $METHOD, $OUTPUT, $TSTOP.

Moreover $UYTES1, $UYTES2, $UYTES3, $UOTES4, $UKMAI1 $UKMCI1 $UKMCI2 are simplified substitutions of subroutines UYTES1, UYTES2, UYTES3, UOTES4, UKMAI1, UKMCI1, UKMCI2 respectively and $SETAG, $SETCG are simplified calling statements (sections 2.6 and 2.14).

We have described the basic possibilities of the UFO control language that are sufficient for preparing the batch input file. More details are given in subsequent chapters and especially in Appendix B. The following example demonstrates the use of the UFO control language for the solution to three collections of optimization problems by two selected methods.

```
$REM ---------------  basic parameters  ---------------

$TOLX='1.0$P-10'; $TOLF='1.0$P-15'; $TOLG='1.0$P-5'; $MIT=800; $MFV=1200
$KOUT=0; $LOUT=1; $MOUT=1

$BATCH
$GLOBAL
$ADD(INTEGER,'\IAG($NA+1)\JAG($MA)')

$REM ---------------  the first method  ---------------

$CLASS='VM'; $TYPE='L'; $DECOMP='M'; $NUMBER=3; $UPDATE='B'

$REM ---------------  the first model   ---------------

$MODEL='AF'; $JACA='S'; $HESF='S'; $NF=100; $NA=500; $MA=2000; $M=9000
$SET(INPUT)
```

```
  CALL EIUB14(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT,IEXT,IERR)
  IF(IERR.NE.0) GO TO 7777
$ENDSET
$SET(FMODELA)
  CALL EAFU14(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU14(NF,KA,X,GA,NEXT)
$ENDSET

$REM ---------------  the first solver  ---------------

$INITIATION
$MODERASE
  CALL $UYTES1
  DO 7777 NEXT=1,15
  CALL $UYTES2
$INPUT
$METHOD
  CALL $UYTES3
  7777 CONTINUE

$REM ---------------  the second method ---------------

$METERASE
$CLASS='GN'; $TYPE='L'; $DECOMP='M'; $NUMBER=3; $UPDATE='D'

$REM ---------------  the second model  ---------------

$MODEL='AQ'; $JACA='S'; $HESF='S'; $NF=100; $NA=500; $MA=2000; $M=9000
$SET(INPUT)
  CALL EIUB15(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT-15,IEXT,IERR)
  IF(IERR.NE.0) GO TO 8888
$ENDSET
$SET(FMODELA)
  CALL EAFU15(NF,KA,X,FA,NEXT-15)
$ENDSET
$SET(GMODELA)
  CALL EAGU15(NF,KA,X,GA,NEXT-15)
$ENDSET

$REM ---------------  the second solver ---------------

$INITIATION
$MODERASE
  DO 8888 NEXT=16,37
  CALL $UYTES2
$INPUT
$METHOD
  CALL $UYTES3
  8888 CONTINUE
```

```
$REM ---------------  the third model   ---------------

$SET(INPUT)
  CALL EIUB18(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT-37,IEXT,IERR)
  IF(IERR.NE.0) GO TO 9999
$ENDSET
$SET(FMODELA)
  CALL EAFU18(NF,KA,X,FA,NEXT-37)
$ENDSET
$SET(GMODELA)
  CALL EAGU18(NF,KA,X,GA,NEXT-37)
$ENDSET

$REM ---------------  the third solver  ---------------

$INITIATION
$MODERASE
  DO 9999 NEXT=38,65
  CALL $UYTES2
$INPUT
$METHOD
  CALL $UYTES3
  9999 CONTINUE

$REM ---------------  the final action  ---------------

  CALL $UOTES4
$END
```

4.2. The batch mode

A switch to the batch mode is realized by using the special substitution $BATCH. If we want to process either the batch mode or the mixed mode we have to prepare a batch input file written in the UFO control language. This input file prescribes the structure of the control program. If a macrovariable is used, it has to be one defined previously. Therefore definitions of macrovariables usually lie at the beginning of the input file. Many macrovariables serve for defining a given optimization problem. The most important among them are the macrovariable $INPUT which determines initial input values (user supplied input) and macrovariables which define problem functions, specifically the model (or objective) function, approximating functions for nonlinear approximation, constrain functions for nonlinear programming, state functions, initial functions and the terminal function for optimization of dynamical systems. These functions are specified by using special macrovariables whose names consist of three parts. The first part can contain letters F, G, D, H or their combinations:

F - Function value.
G - Gradient with respect to basic variables.
D - Gradient with respect to state variables.
H - Hessian matrix with respect to basic variables.
FG - Function value and gradient with respect to basic variables.
FD - Function value and gradient with respect to state variables.
GD - Gradient with respect to basic variables and gradient with respect to state variables.
FGD - Function value, gradient with respect to basic variables and gradient with respect to state variables.

FGH - Function value, gradient with respect to basic variables and Hessian matrix with respect to basic variables.

The second part always has the form MODEL. The third part can contain letters F, A, C, E, Y and also an additional letter S:

F - The model function or the terminal function.
A - The selected approximating function.
AS - All approximating functions.
C - The selected constraint function.
CS - All constraint functions.
E - The selected state function.
ES - All state functions.
Y - The selected initial function.
YS - All initial functions.

The following combinations are possible:

$FMODELF	$FMODELA	$FMODELC	$FMODELE	$FMODELY
	$FMODELAS	$FMODELCS	$FMODELES	$FMODELYS
$GMODELF	$GMODELA	$GMODELC	$GMODELE	$GMODELY
	$GMODELAS	$GMODELCS	$GMODELES	$GMODELYS
$DMODELF	$DMODELA		$DMODELE	
			$DMODELES	
$HMODELF	$HMODELA	$HMODELC		
	$HMODELAS	$HMODELCS		
$FGMODELF	$FGMODELA	$FGMODELC	$FGMODELE	$FGMODELY
	$FGMODELAS	$FGMODELCS	$FGMODELES	$FGMODELYS
$FDMODELF	$FDMODELA		$FDMODELE	
			$FDMODELES	
$GDMODELF	$GDMODELA		$GDMODELE	
			$GDMODELES	
$FGDMODELF	$FGDMODELA		$FGDMODELE	
			$FGDMODELES	
$FGHMODELF	$FGHMODELA	$FGHMODELC		
	$FGHMODELAS	$FGHMODELCS		

The choice of a suitable way for problem function definitions is ambiguous and problem dependent. We can only give several remarks:

1. The basic and most general way is the use of different macrovariables for different quantities (values, gradients, Hessian matrices) together with an independent evaluation of individual functions (the last letter is different from S). This way saves the computer storage and frequently also the computational time.

2. Sometimes, evaluations of gradients require function values. In this case, it can be advantageous to compute values and gradients simultaneously. A similar consideration also holds for Hessian matrices.

3. Even if simultaneous evaluations of all approximating (constraint, state, initial) functions increase storage requirements, it can be advantageous if there are complicated computations common for all such functions, and also if a problem has a low dimension or a sparse structure. It is frequently advantageous for the evaluation of state and initial functions when the dynamical systems are optimized.

4. If the gradients of approximating (constraint, state, initial) functions are computed simultaneously (the last letter is equal to S), then also function values have to be computed simultaneously. Similarly if the Hessian matrices are computed simultaneously, then also function values and gradients have to be computed simultaneously.

A simple example of a batch input file was shown in section 1.2. We repeat it here with some explanations:

```
$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
```

By using the macrovariable $INPUT, we specify the initial values of variables $x_1 = -1.2$ and $x_2 = 1.0$. By using the macrovariable $FMODELF, we specify the model function value (the model function gradient is not specified, it will be computed numerically). The macrovariable $NF defines the number of variables and $NOUT is a print specification. The macroinstruction $BATCH switches the mode to the batch mode. The macroinstruction $STANDARD defines the standard form of the control program. Descriptions of more complicated problems are shown in chapter 5.

In the above example, a direct definition of a model function value is used. We can also use indirect specifications by means of the Fortran 77 subroutines or the files prepared beforehand. Suppose that the model function value is defined by using the subroutine **EFFU01** or is specified in the file **FVAL.FOR**. Then we can write:

```
$SET(FMODELF)
  CALL EFFU01(NF,X,FF,NEXT)
$ENDSET
```

or

```
$SET(FMODELF)
  $INCLUDE('FVAL.FOR')
$ENDSET
```

or

```
$SET(FMODELF)
  $SUBST('FVAL.FOR')
$ENDSET
```

The last possibility is useful if the model function value specification is written in a free format or it contains the BEL macroinstructions.

If we need to utilize user supplied subroutines, we can include them into the control program using the macrovariable $SUBROUTINES:

```
$SET(SUBROUTINES)
  user supplied subroutines
$ENDSET
```

In this case, some exceptions laid on the text of user supplied subroutines forced by the UFO preprocessor have to be satisfied. All comments have to begin with character '*', the continuation line has to begin with character '&', character '$' has to be replaced by '$$' and character ';' does not have to be present.

The batch input file should also contain optimization method selection. Fortunately, this selection is not critical since the optimization method can be chosen automatically by using knowledge bases

contained in the UFO system templates. Here we will only demonstrate some possibilities. The greatest influence on the optimization method selection have the following macrovariables:

$CLASS	- Class of optimization methods (heuristic, conjugate gradient, variable metric, variable metric with limited storage, modified Newton, truncated Newton, Gauss-Newton, quasi-Newton, quasi-Newton with limited storage, proximal bundle, bundle-Newton).
$TYPE	- Type of optimization methods (line search, trust region, SQP filter).
$DECOMP	- Type of matrix decomposition (original matrix, Choleski decomposition, inversion).
$NUMBER	- Individual methods for direction determination (various direct, various iterative).
$UPDATE	- Type of variable metric or quasi-Newton update.

A more detailed description of these choices together with other choices ($MET, $MET1, $MET2, $MET3, $MES, $MES1, $MES2, $MES3, $MOS, $MOS1, $MOS2, $MOS3) is given in section 3.

4.3. The text dialogue mode

A switch to the text dialogue mode is realized by using the special substitution $TDIALOGUE. This is equivalent to the substitution $DIALOGUE in the UNIX version of the UFO system. If this is the case, a sequence of questions appear on the screen in the text form. Each question, which is placed in its own frame, consists of the macrovariable description usually followed by the list of its possible values. The name of a macrovariable together with its default value is written on the top of the frame. We have two possibilities for an answer. First, the required value can be entered from the keyboard. Secondly, we can press ENTER to choose the default value. After the assignment of a value to the macrovariable, a new question immediately appears on the screen until the last one is exhausted. The dialogue mode can be terminated by entering character '!' from the keyboard. We demonstrate four questions as an example:

```
                    ? INPUT ( ) ?
 USER SUPPLIED INPUT:

      HERE THE STARTING POINT, BOUNDS FOR VARIABLES,
      TYPES OF CONSTRAINTS, THE STRUCTURE OF SPARSE PROBLEM,
      AND OTHER INPUT DATA HAVE TO BE SPECIFIED.
```

Here a user supplied input is expected. This is a text which should be entered from the keyboard.

```
                    ? MODEL (FF) ?
 TYPE OF OBJECTIVE FUNCTION
    FF - GENERAL FUNCTION
    FL - LINEAR FUNCTION
    FQ - QUADRATIC FUNCTION
    AF - SUM OF FUNCTIONS
    AQ - SUM OF SQUARES
    AP - SUM OF POWERS
    AM - MINIMAX
    DF - DIFFERENTIAL SYSTEM WITH GENERAL INTEGRAL CRITERION
    DQ - DIFFERENTIAL SYSTEM WITH INTEGRAL OF SQUARES
    NO - MODEL IS NOT SPECIFIED
```

Here an optimization model, i.e. a type of the objective function, is chosen. We have 10 possibilities, FF, FL, FQ, AF, AQ, AP, AM, DF, DQ, NO. The default value of the macrovariable $MODEL corresponding to the general objective function is FF. By pressing ENTER, the default value FF is accepted.

```
? NF (0) ?
NUMBER OF VARIABLES
```

Here the number of variables is expected. This is a positive integer. No default value is offered, i.e. we have to set a value. If this value is not a positive integer, the answer is ignored and the same question appears on the screen.

```
? FMIN (-1.0D 60) ?
LOWER BOUND FOR FUNCTION VALUE
```

Here a real constant is expected. By pressing ENTER the default value -1.0D 60 is accepted.

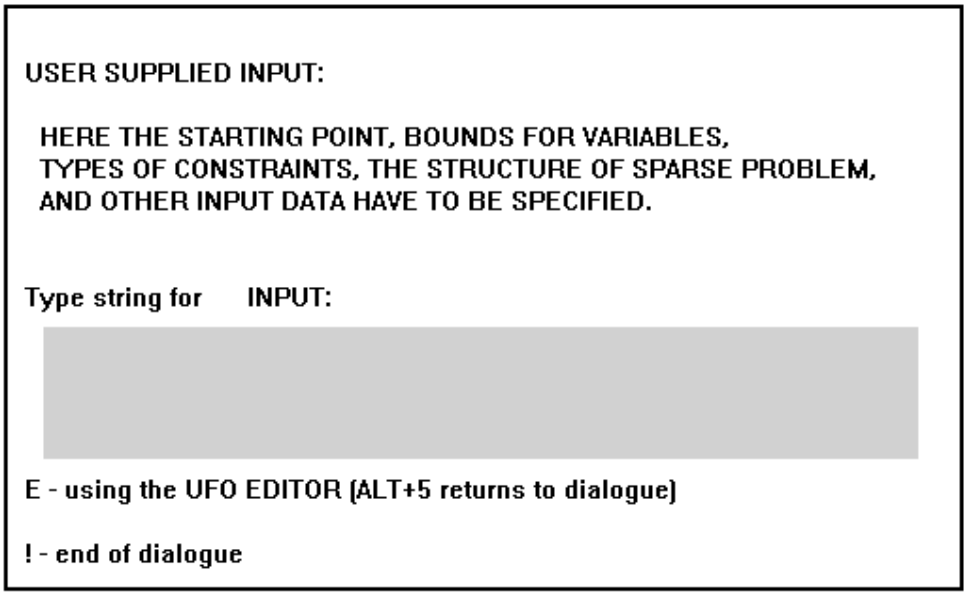
More details concerning a text dialogue mode are given in Appendix A, where a complete text dialogue concerning unconstrained minimization of the Rosenbrock function is shown.

4.4. The graphic dialogue mode

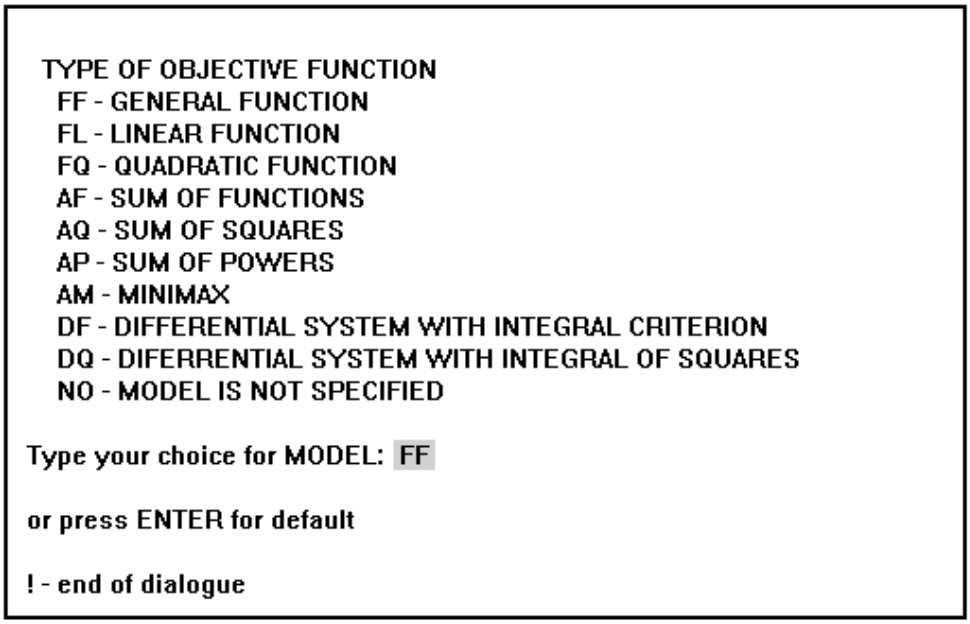
The graphic dialogue mode can be used only on PC computers under the MS DOS system. This possibility is not allowed on the UNIX workstations. A switch to the graphic dialogue mode is realized by using the special substitution $GDIALOGUE. This is an equivalent to the substitution $DIALOGUE in the PC version of the UFO system. If this is the case, a sequence of screens follows. Each screen realizes one question which is in fact the same as that in the text dialogue mode. Nevertheless, the graph dialogue mode has several advantages over the text one:

1. Information is better arranged on the screen.
2. The window for typing answers is in fact a simple editor. Therefore the text can be easily corrected and a movement controlled by arrows is possible.
3. Application of the special UFO editor is possible for realizing more complicated answers. The UFO editor works with multiple windows so that an answer can be set up from several sources. Therefore a convenient utility of the batch mode can also be used in the dialogue mode.

To compare text and graphic dialogue modes, we again demonstrate the above four questions:

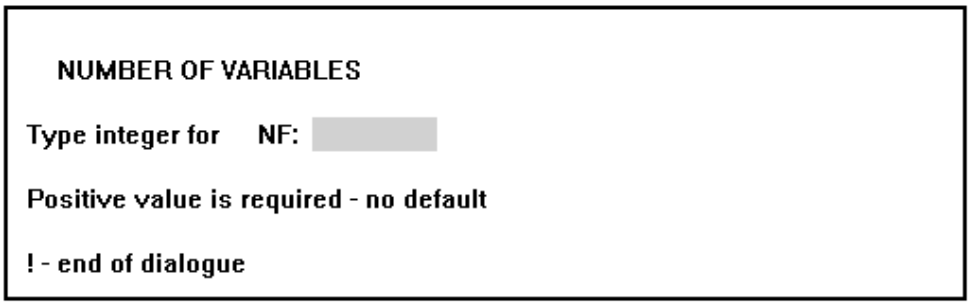


Here a user supplied input is expected. This is a text which should be written into the window displayed on the screen (followed by pressing ENTER). If this text is more complicated, we can use the UFO editor by typing character 'E' and pressing ENTER. The return from the UFO editor to the graphic dialogue is realized by pressing <alt-5> (section 1.3). The dialogue mode can be terminated by typing character '!' and pressing ENTER.

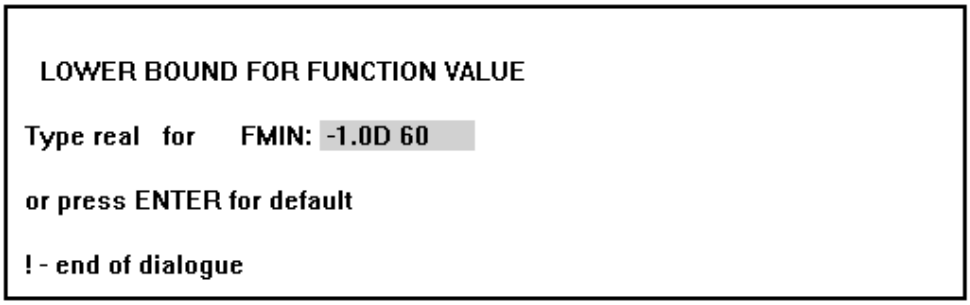


Here an optimization model, i.e. a type of the objective function, is chosen. We have 10 possibilities, FF, FL, FQ, AF, AQ, AP, AM, DF, DQ, NO. The default value of the macrovariable $MODEL, corresponding

to the general objective function, is FF. By pressing ENTER, the default value FF is accepted. If we want to choose a different possibility, it has to be written into the two-character window, followed by pressing ENTER. The dialogue mode can be terminated by typing character '!' and pressing ENTER.



Here the number of variables is expected. This is a positive integer. No default value is offered, i.e. we have to enter any value. If this value is not a positive integer, the answer is ignored and another answer is expected. The dialogue mode can be terminated by typing character '!' and pressing ENTER.



Here a real constant is expected. By pressing ENTER the default value -1.0D 60 is accepted. If we want to choose a different value, it has to be written into the twenty-character window, followed by pressing ENTER. The dialogue mode can be terminated by typing character '!' and pressing ENTER.

5. Output possibilities in the UFO system

The UFO system has many output possibilities including graphic pictures. These output possibilities can be divided into five basic groups.

5.1. Basic screen output

The basic screen output can be used only if $GRAPH='N' and $DISPLAY='N'. In this case, individual rows corresponding to the iterations and the final results are printed on the screen consequently. A print level of the screen output is determined by using the macrovariables $MOUT and $NOUT. The macrovariable $MOUT can have the following values:

$MOUT= 0 -	Screen output is suppressed.
$MOUT=± 1 -	Standard output. The final results appear on the screen.
$MOUT=± 2 -	Extended output. Additional information from every iteration appears on the screen.
$MOUT=± 3 -	Extended output. Additional final results of linear or quadratic programming subproblems appear on the screen.
$MOUT=± 4 -	Extended output. Additional information from every iteration of linear or quadratic programming subproblems appears on the screen.

If $MOUT>0, a standard line of the final results is printed, while if $MOUT<0 then a modified line of the final results, containing the termination criterion, is printed.

The macrovariable $NOUT can have the following values:

$NOUT= 0 -	Short final results (scalar variables) appear on the screen.
$NOUT= 1 -	Extended final results (vectors) appear on the screen.

5.2. Extended screen output

If we want to use an extended screen output, we have to set $DISPLAY='Y' (the default value is $DISPLAY='N'). This type of screen output consists of text pages which correspond to individual iterations and the final results. The final results are divided into several groups which can be displayed successively. We can change the displayed group by typing particular characters from the keyboard.

Change of the displayed group of the final results:

F - (function) :	Value of the objective function and statistics.
V - (variables) :	Values of variables if NF>0 (with their bounds if KBF>0).
A - (approximation) :	Values of approximating functions if NA>0 (with their prescribed values if KBA>0). Values of selected components of a solution of the set of ordinary differential equations at the prescribed mesh points if NE>0.
C - (constraints) :	Values of constraint functions if NC>0 (with their bounds if KBC>0).
D - (data) :	Data which specify the problem solved (sizes of problem and additional specifications).
O - (options) :	Options which specify the method used.

Exit:

Q - (quit) :	Exit from the extended screen output.

After typing each character we must use ENTER.

Besides these possibilities we can stop every iteration for scanning the iterative process. It is specified if we set $SCAN='Y' (the default value is $SCAN='N'). If $SCAN='N', the output of iterations is suppressed. Scanning of the iterative process can be terminated by using character '!' from the keyboard.

5.3. Graphic screen output

The graphic screen output can be used only on PC computers under the MS DOS system. This possibility is not allowed on the UNIX workstations. If we want to use a graphic screen output, we have to set $GRAPH='Y' (the default value is $GRAPH='N'). In this case, both iterations and the final results appear in the graphic mode. In general, the graphic screen output is a sequence of screens which can be examined successively in a required order. A change of the screen is carried out by using the menu given on the top of this screen. We have three possibilities. First, the character displayed as a capital at the menu item can immediately be typed from the keyboard. Secondly, we can use characters $\rightarrow$ and $\leftarrow$ which realize movement in the top menu. The underlined menu item is then selected by pressing ENTER. Finally, we can apply a mouse click to the menu item. In the subsequent graphic screen output description, we focus our attention to the first possibility without a loss of generality.

The graphic form of the final results can be specified in detail by using macrovariables $PATH ('N'- no, 'Y'- yes, 'E'- extended), $MAP ('N'- no, 'Y'- yes, 'E'- extended), $HIL ('N'- no, 'Y'- yes) and $ISO ('N'- no, 'Y'- yes). The final results are divided into several groups which can be displayed successively. We can change the displayed group by typing particular characters from the keyboard.

Change of the displayed group of the final results:

F - (function) :	Value of the objective function and statistics.
V - (variables) :	Values of variables if NF>0 (with their bounds if KBF>0).
A - (approximation) :	Values of approximating functions if NA>0 (with their prescribed values if KBA>0). Values of selected components of a solution of the set of ordinary differential equations at the prescribed mesh points if NE>0.
C - (constraints) :	Values of constraint functions if NC>0 (with their bounds if KBC>0).
D - (data) :	Data which specify the problem solved (sizes of problem and additional specifications).
O - (options) :	Options which specify the method used.
T - (path) :	Values of the objective function and selected variables (we can change these variables during the graphic output, if we have specified $PATH='E') in the last NPA iterations (only if $PATH='Y' or $PATH='E').

Exit:

Q - (quit) :	Exit from the graphic output.
X - (exit) :	Exit from the UFO system.

Besides these possibilities we can stop every iteration for scanning the iterative process. It is specified if we set $SCAN='Y' (the default value is $SCAN='N'). In every iteration, we can choose one of the possibilities F, V, A, C, D, O as in the case above. If we have chosen either V (variables) or A (approximation) or C (constraints), the intermediate results can be displayed graphicly by typing G (graph) from the keyboard. In all these cases we can execute a single iteration by typing SPACE merely. We can also execute all iterations until the k-th one by typing J (jump) and entering the number k. Finally, by typing U (automatic), all remaining iterations are executed without scanning.

Besides text representations in the graphic mode, which are essentially like the ones in the extended screen output (with the choice $DISPLAY='Y'), we can chose several types of graphic data representation.

a) Graphic picture:

If we have chosen either V (variables) or A (approximation) or C (constraints), the results can be displayed graphicly by typing G (graph) from the keyboard. A graphic picture appears on the screen in this case. It contains either values of variables with indices I, $1 \leq \mathrm{I} \leq \mathrm{NF}$, or values of the approximating functions with indices KA, $1 \leq \mathrm{KA} \leq \mathrm{NA}$, or values of the constraint functions with indices KC, $1 \leq \mathrm{KC} \leq \mathrm{NC}$. If we have chosen A (approximation) in the case of NE>0, the graphic picture contains a component (with the index VAR) of a solution of the set of ordinary differential equations at the mesh

points AT(KA), $1 \leq KA \leq NA$. We have to define the index VAR from the keyboard in this case. The graphic picture can be changed by typing the particular characters from the keyboard.

Change of representation:

V - (values) :	Values are drawn.
O - (ordinates) :	Values and ordinates from zero axis are drawn.
C - (curve) :	Values are connected by a curve.
M - (mixed) :	Curve and ordinates are drawn.

Change of graph (if either KBF>0 or KBA>0 or KBC>0):

F - (functions) :	Either values of variables X(I), $1 \leq I \leq NF$, or values of the approximating functions AF(KA), $1 \leq KA \leq NA$, or values of the constraint functions CF(KC), $1 \leq KC \leq NC$, are demonstrated.
A - (approximation) :	Either values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, or values of the approximating functions AF(KA) together with their prescribed values AM(KA) , $1 \leq KA \leq NA$, or values of the constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq KC \leq NC$, are demonstrated.
D - (differences) :	Either the differences between variables and their bounds or the differences between the approximating functions and their prescribed values or the differences between the constraint functions and their bounds are demonstrated.

Continuation (if either $NF > 200$ or $NA > 200$ or $NC > 200$):

P - (previous) :	Previous set of at most 200 values is drawn.
N - (next) :	Next set of at most 200 values is drawn.

Choice of the next displayed iteration (only if $SCAN='Y'):

J - (jump) :	The iterative process is stopped at the k-th iteration. Number k is read from the keyboard.
U - (automatic) :	All remaining iterations are executed without scanning.

New graph or return:

W - (new) :	This possibility can be used only if NE>0. Then a new component (with a new index VAR) of a solution of the set of ordinary differential equations is drawn. We have to define a new index VAR from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

If we have chosen F (function) as a group of final results, we can use additional graphic representations.

b) Two-dimensional orbit:

If NE>1, we can draw an orbit of two components of a solution of the set of ordinary differential equations by typing G (graph) from the keyboard. We have to define an index VAR for every selected component of a solution (according to the text appeared on the screen). The two-dimensional orbit can be changed by typing particular characters from the keyboard.

Change of the orbit:

V - (values) :	Values are drawn.
C - (curves) :	Values are connected by a curve.

New orbit or return:

W - (new) :	New components of a solution of the set of ordinary differential equations are drawn. We have to define new two indices from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

c) Three-dimensional orbit:

If NE>2, then we can draw an orbit of three components of a solution of the set of ordinary differential equations by typing I (picture) from the keyboard. We have to define an index VAR for every selected component of a solution (according to the text appeared on the screen). The three-dimensional orbit can be changed by typing particular characters from the keyboard.

Change of the orbit:

V - (values) :	Values are drawn.
C - (curves) :	Values are connected by a curve.
O - (rotate) :	Rotation of values or curves about a vertical axis by a subsequently entered angle Dfi.
T - (tilt) :	Tilting rotated values or curves by a subsequently entered angle Dtheta.
A - (axes) :	Drawing a picture with rotated and tilted axes.
S - (scale) :	Scaling of rotated and tilted values or curves to make full use of the screen.

New orbit or return:

W - (new) :	New components of a solution of the set of ordinary differential equations are drawn. We have to define new three indices from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

d) Coloured map of the objective function:

If we have specified either $MAP='Y' or $MAP='E' (default value is $MAP='N'), we can draw a coloured map of the objective function by typing M (map) from the keyboard. This picture can be changed by typing particular characters from the keyboard.

Change of the map:

L - (linear) :	Linear scale of the coloured map.
G - (logarithmic) :	Logarithmic scale of the coloured map.
R - (refinement) :	Refinement of the coloured map.
B - (back) :	Back refinement of the coloured map.
N - (inverse) :	Coloured map of the objective function negation.

Another type of picture, new map or return:

H - (hills) :	Drawing an objective function surface with respect to visibility (only if $HIL='Y' is specified).
S - (isolines) :	Drawing contours of the objective function (only if $ISO='Y' is specified).
W - (new) :	Selection of new variables and drawing a new coloured map.
Q - (quit) :	Return to the displayed group of final results.

If we set $MAP='Y', one picture for two variables is drawn. If we set $MAP='E', three pictures for all combinations of two from three variables are drawn. In both cases we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every variable used (according to the text appeared on the screen). Note that the choice $MAP='E' excludes the choices $HIL='Y' and $ISO='Y' so that the other pictures cannot be used.

e) Objective function surface:

If we have specified $HIL='Y' (default value is $HIL='N'), we can draw an objective function surface with respect to visibility by typing H (hills) from the keyboard. This picture can be changed by typing particular characters from the keyboard.

Change of the surface:

L - (linear) :	Linear scale of the surface.
G - (logarithmic) :	Logarithmic scale of the surface.

R - (refinement) : Refinement of the surface.
B - (back) : Back refinement of the surface.
O - (rotate) : Rotation of the surface about a vertical axis by a subsequently entered angle Dfi.
T - (tilt) : Tilting the rotated surface by a subsequently entered angle Dtheta.
F - (face) : Facing the rotated surface (drawing the rotated surface without tilting).
N - (inverse) : Surface of the objective function negation.

Another type of picture, new surface or return:

M - (map) : Drawing a coloured map of the objective function (only if $MAP='Y' is specified).
S - (isolines) : Drawing contours of the objective function (only if $ISO='Y' is specified).
W - (new) : Selection of new variables and drawing new surface.
Q - (quit) : Return to the displayed group of final results.

Before drawing the objective function surface we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every variable used (according to the text appeared on the screen).

f) Objective function contours:

If we have specified $ISO='Y' (default value is $ISO='N'), we can draw an objective function contours by typing S (isolines) from the keyboard. This picture can be changed by typing particular characters from the keyboard.

Change of contours:

L - (linear) : Linear scale of contours.
G - (logarithmic) : Logarithmic scale of contours.
R - (refinement) : Refinement of contours.
B - (back) : Back refinement of contours.
O - (colour) : Colouring of contours and used levels.
N - (inverse) : Inverse colouring of contours and used levels.

Another type of picture, new contours or return:

M - (map) : Drawing a coloured map of the objective function (only if $MAP='Y' is specified).
H - (hills) : Drawing an objective function surface with respect to visibility (only if $HIL='Y' is specified).
W - (new) : Selection of new variables and drawing a new surface.
Q - (quit) : Return to the displayed group of final results.

g) Graphic path of the objective function and selected variables:

If we have chosen T (path), we can display the values of the objective function as a function graph by typing G (graph) or draw the objective function contours with the path in the last NPA iterations by typing S (isolines). The graph can be changed in the same way as in a).

Change of contours:

L - (linear) : Linear scale of contours.
G - (logarithmic) : Logarithmic scale of contours.
R - (refinement) : Refinement of contours.
B - (back) : Back refinement of contours.
Z - (zoom) : Zoom of the path for the number of last iterations entered.

Another type of picture, new contours or return:

W - (new) : Selection of new variables and drawing new contours (only if we have specified $PATH='E').
Q - (quit) : Return to the displayed group of final results.

Before drawing the objective function contours we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every variable used (according to the text appeared on the screen).

5.4. Text file output

The UFO system contains a great number of text file output procedures which are controlled by using the macrovariables $KOUT, $KOUT1, $KOUT2, $KOUT3, and $LOUT. These text file output procedures are useful especially for debugging new optimization methods. The UFO system works with the output file `P.OUT`. The Fortran number of this output file defines the common variable IWR. The macrovariables $KOUT, $KOUT1, $KOUT2, $KOUT3 determines what is printed and the macrovariable $LOUT has an influence on the extent of the print.

The macrovariable $KOUT can have the following values:

$KOUT= 0 -	Text file output is suppressed (the file `P.OUT` is empty) .
$KOUT= ± 1 -	Standard output. The heading and the final results are printed together with selected information in each accepted iteration.
$KOUT= ± 2 -	Extended output. Additional information, obtained from stepsize selection, is printed.
$KOUT= ± 3 -	Extended output. Additional information, obtained from direction determination and variable metric update, is printed.
$KOUT= ± 4 -	Extended output. Additional information, obtained from linear constraint addition and deletion, is printed.
$KOUT= ± 5 -	Extended output. Additional information, obtained from numerical differentiation, is printed.

If $KOUT>0, a standard heading is printed while if $KOUT<0, an extended heading, containing problem specifications and optimization options, is printed.

The selection of iterations accepted for print is controlled by the contents of the macrovariables $KOUT1, $KOUT2, $KOUT3. If KOUT1≤ KOUT2, only the iterations whose numbers are between KOUT1 and KOUT2 are assumed, but the KOUT3−1 ones are always omitted (KOUT1 is a lower bound, KOUT2 is an upper bound and KOUT3 is a step). Similarly, if KOUT1>KOUT2, only the iterations whose numbers are smaller than KOUT2 or greater than KOUT1 are assumed, but the KOUT3−1 ones are always omitted. If $KOUT3=0, no iterations are assumed.

While the macrovariable $KOUT specifies which information is printed, the macrovariable $LOUT specifies how much information is printed:

$LOUT= 0 -	Basic output. The basic information (1 row if $KOUT=1) is printed in each accepted iteration.
$LOUT=± 1 -	Extended output. Additional scalars, together with the vector of variables, are printed.
$LOUT=± 2 -	Extended output. Additional vectors (usually gradients) are printed.
$LOUT=± 3 -	Extended output. Additional matrices (usually Hessian matrices) are printed.
$LOUT=± 4 -	The most extended output. All useful data are printed.

If $LOUT>0, the basic part of the information is printed. If $LOUT<0, a more extensive part of the information is printed.

The macrovariable $LOUT has an additional significance. If $KOUT=0 and $LOUT>0, a copy of the basic screen output is provided. If $KOUT=0 and $LOUT<0, paper saving print is assumed. In the last case, only several rows are printed for every solution. This type of output is useful for simultaneous tests of optimization methods.

To show a typical basic output which corresponds to the choices $KOUT=1, $KOUT3=0 and $LOUT=0 we propose the following results from unconstrained optimization:

```
UNCONSTRAINED MINIMIZATION USING UFO SYSTEM
-------------------------------------------
OPTIMIZATION SUBROUTINE : U1FDU1
DIRECTION DETERMINATION : UDDLI1
STEP SIZE DETERMINATION : US0L01
FUNCTION  DETERMINATION : UF1F01
GRADIENT  DETERMINATION : UF0GS2
H MATRIX  DETERMINATION :
VARIABLE METRIC  UPDATE : UUDBI1

PROBLEM
-------
NF = 2               KDF= 0 KSF= 1 KCF= 2 KBF= 0 ISNF= 1 NORMF= 0
NA = 0 NAL= 0 MAL= 0 KDA=-1 KSA= 0 KCA= 0 KBA= 0 ISNA= 0 NORMA= 0
NC = 0 NCL= 0 MCL= 0 KDC=-1 KSC= 0 KCC= 0 KBC= 0 ISNC= 0 NORMC= 0

FINAL RESULTS
-------------
FF= -.3072281498D+03
X = -.6228926480D+01  .4363683132D+01

TERMINATION: ITERM=4 GRAD TOL F=-.307D+03 G= .480D-06 D= .148D-07

STATISTICS
----------
NIT = 14                     NDEC = 0
NFV = 58  NAV = 0  NCV = 0   NRES = 6
NFG =  0  NAG = 0  NCG = 0   NREM = 0
NFH =  0  NAH = 0  NCH = 0   NADD = 0
```

Here the optimization subroutines used are listed on the top followed by problem specifications. After brief results, the termination causes are written. The termination cause ITERM=4 (GRAD TOL) corresponds to the attainment of the required gradient norm, F is the objective function value, G is the maximum absolute value of gradient elements and D is the maximum relative change of variables. The statistics contains the number of iterations NIT, the number of decompositions NDEC, the number of restarts NRES, the number of constraint deletions or additions NREM or NADD respectively, and a set of data concerns numbers (N) of model function (F) or approximating functions (A) or constraint functions (C) values (V) or gradients (G) or Hessian matrices (H) evaluations respectively.

5.5. User supplied output

The UFO system allows utilizing both the user supplied output subroutines and the post-processing subroutines. These subroutines can be inserted in the control program by using the macrovariable $OUTPUT:

```
$SET(OUTPUT)
      Calling the user supplied output subroutines.
      Calling the post-processing subroutines.
$ENDSET
```

The parameters of the user supplied output subroutines and the post-processing subroutines must satisfy the UFO conventions. For example, the vector of variables, the model function value and the model function gradient must be denoted X, FF and GF, respectively (see chapter 2).

5.6. Storing final results

If we set $OUTPUTDATA='Y', the final values of variables X(I), 1≤ I≤ NF, are stored in file `P.DAT`. Similarly, if we set $INPUTDATA='Y', the values of variables X(I), 1≤ I≤ NF, from file `P.DAT` are used as input data for a new optimization process.

5.7. Other output files

The UFO system uses two other output files `P.DIM` and `P.SIF` which contain additional information about the problem solved. File `P.DIM` shows us the problem dimension. It contains the numbers of variables, approximating functions, constraints and also numbers of nonzero elements in sparse structures. For example, if we apply the UFO system to the input file `TSIF21.UFO`, then the file `P.DIM` contains the following text:

```
PROBLEM: NEXT =       0
NUMBER OF VARIABLES:        NF  =     450
NUMBER OF CONSTRAINTS:      NC  =     360
NUMBER OF NONZERO ELEMENTS: MC  =    1576
NUMBER OF NONZERO ELEMENTS: MHC =      55
NUMBER OF NONZERO ELEMENTS: MCH =    4736
NUMBER OF NONZERO ELEMENTS: M   =    2779
```

File `P.SIF` contains information concerning SIF files of the CUTE collection (section 6.3). This file is generated by the SIF decoder. For example, if we apply the UFO system to the input file `TSIF21.UFO`, the file `P.SIF` contains the following text:

```
Problem name: BRITGAS
The objective function uses      1 nonlinear group

There are    360 nonlinear equality constraints

There are    426 variables bounded only from below
There are     24 variables bounded from below and above
```

5.8. Error messages

If we use the specification $MOUT>0 (basic screen output), then nonstandard terminations are indicated. The message consists of three parts: the name of a critical subroutine, the number of a message, and an explanation text. For example, if the number of iterations is exceeded, we obtain the following message:

```
  0  NIT= 500  NFV=2545  NFG= 0  NDC=1556  NCG= 0  F= .122D+06  G= .112D+05
 UYFUT1: (-2) MAXIMUM NUMBER OF ITERATIONS
```

Error messages are very useful especially in case the problem dimension is invalid. For example, if the number of nonzero elements in the Jacobian matrix is specified incorrectly, then we obtain the message:

```
  0  NIT=   0  NFV=   0  NFG= 0  NDC=   0  NCG= 0  F= .000D+00  G= .000D+00
 UZLMIN: (78) LACK OF SPACE : MA TOO SMALL
 ACTUAL VALUE:     298   -   DECLARED VALUE:     250
```

Here UZLMIN is a subroutine where an error was detected, 78 is the error number and MA TOO SMALL is the explanation. In this case, additional information (ACTUAL VALUE and DECLARED VALUE) is given.

The following table presents all UFO error messages (error numbers and explanations):

```
-1   -  MAXIMUM NUMBER OF FUNCTION EVALUATIONS : MFV TOO SMALL
-2   -  MAXIMUM NUMBER OF ITERATIONS : MIT TOO SMALL
-3   -  MAXIMUM NUMBER OF CYCLES : MIC TOO SMALL
 1   -  BAD DECOMPOSITION
 2   -  BAD INTERVAL IN THE OLC DIRECTION DETERMINATION
 3   -  MAXIMUM NUMBER OF STEPS IN THE OLC DIRECTION DETERMINATION
 4   -  BREAKDOWN IN THE ITERATIVE METHOD
 5   -  BREAKDOWN IN THE ITERATIVE METHOD
 6   -  MAXIMUM NUMBER OF REDUCTIONS
 7   -  NEGATIVE DIRECTIONAL DERIVATIVE
 8   -  BAD INTERVAL FOR INTERPOLATION
 9   -  BAD PREDICTION IN THE TRUST REGION METHOD
10   -  RESTART
11   -  FEASIBLE SOLUTION DOES NOT EXIST
12   -  BOUNDED SOLUTION DOES NOT EXIST
13   -  FEASIBLE SOLUTION DOES NOT EXIST
14   -  FEASIBLE TRUST REGION DOES NOT EXIST
15   -  INVALID SITUATION IN CONSTRAINT HANDLING
16   -  INVALID SITUATION IN CONSTRAINT HANDLING
17   -  LACK OF SPACE IN CONSTRAINT HANDLING : MMAX TOO SMALL
18   -  LACK OF SPACE IN CONSTRAINT HANDLING : MMAX TOO SMALL
19   -  BAD INPUT DATA
20   -  BAD INPUT DATA
21   -  UXSGFM: NAU IS DECLARED TOO SMALL
22   -  UXSGFM: NZ IS DECLARED TOO SMALL
23   -  UXSGFM: JACOBIAN MATRIX IS TOO UNSTABLE
24   -  UXSGFM: JACOBIAN MATRIX IS SINGULAR
25   -  UXSGFM: NZ IS TOO SMALL FOR THE FACTOR
26   -  UXSGFM: NAU IS TOO SMALL FOR THE FACTOR
27   -  UXSGFM: NZ IS TOO SMALL FOR DATA MANIPULATIONS AFTER FACTORIZATION
28   -  UXSGFM: COLUMN SCHEME FOR THE FACTOR IS NOT CREATED: LACK OF SPACE
29   -  BAD INPUT DATA
30   -  BAD INPUT DATA
31   -  UXSGUM: NAU IS DECLARED TOO SMALL
32   -  UXSGUM: JACOBIAN MATRIX IS SINGULAR
33   -  UXSGUM: FACTOR IS BADLY CONDITIONED
34   -  UXSGUM: LITTLE SPACE FOR L-UPDATES
35   -  UXSGUM: JACOBIAN MATRIX SINGULARITY IS FACED
36   -  UNBOUNDENESS IS FACED
37   -  UKLTS3: ROWS ARE NOT SPECIFIED
38   -  UKLTS3: COLUMNS ARE NOT SPECIFIED
39   -  UKLTS3: TYPE IS NOT SPECIFIED
40   -  UKLTS3: TYPC IS NOT DEFINED
41   -  LACK OF SPACE FOR THE CHOLESKI FACTOR : MMAX TOO SMALL
42   -  LACK OF SPACE FOR A SYMBOLIC FACTORIZATION : MMAX TOO SMALL
43   -  LACK OF SPACE FOR THE FILL-IN
44   -  LACK OF SPACE FOR NUMERICAL DIFFERENTIATION : M TOO SMALL
45   -  STRUCTURAL SINGULARITY DURING INCOMPLETE LU FACTORIZATION
46   -  INVALID STRUCTURE FOR INCOMPLETE LU FACTORIZATION
47   -  LACK OF SPACE IN NUMERICAL DIFFERENTIATION : NVAR TOO SMALL
48   -  LACK OF SPACE IN THE INCOMPLETE DECOMPOSITION : MMAX TOO SMALL
49   -  LACK OF SPACE IN THE SCHUR COMPLEMENT : MMAX TOO SMALL
```

```
50   -  LACK OF SPACE FOR THE FACTOR
51   -  INSUFFICIENT STORAGE FOR NONZERO SUBSCRIPTS
52   -  LACK OF SPACE IN THE FRONTAL SCHEME
53   -  ERROR IN THE FRONTAL SCHEME
54   -  LACK OF SPACE IN THE FRONTAL SCHEME
55   -  ERROR IN THE FRONTAL SCHEME
56   -  LACK OF SPACE IN THE INTEGER FIELD
57   -  LACK OF SPACE IN THE REAL FIELD
58   -  ZERO INDEX
59   -  DIMENSION ERROR
60   -  LACK OF SPACE IN THE WORKING FIELD
61   -  INVALID MATRIX ORDER
62   -  NUMBER OF NONZEROS SMALLER THAN ZERO
63   -  INVALID ENTRIES IN THE INPUT MATRIX
64   -  INCONSISTENT MEMORY
65   -  LACK OF SPACE IN THE INTEGER FIELD : MMAX TOO SMALL
66   -  LACK OF SPACE IN THE REAL FIELD : MMAX TOO SMALL
67   -  INVALID LU FACTORS
68   -  MAXIMUM INTEGER TOO SMALL
69   -  INVALID INPUTS
70   -  ZERO PIVOT WHEN DEFINITENESS IS DECLARED
71   -  CHANGE IN SIGN OF PIVOT ENCOUNTERED
72   -  SINGULARITY DETECTED
73   -  NONZERO ELEMENT IGNORED
74   -  PIVOT HAS DIFFERENT SIGN FROM THE PREVIOUS ONE
75   -  LACK OF SPACE : M TOO SMALL
76   -  LACK OF SPACE : MAH TOO SMALL
77   -  LACK OF SPACE : MCH TOO SMALL
78   -  LACK OF SPACE : MA TOO SMALL
79   -  LACK OF SPACE : MC TOO SMALL
80   -  LACK OF SPACE : NF TOO SMALL
81   -  LACK OF SPACE : NA TOO SMALL
82   -  LACK OF SPACE : NC TOO SMALL
83   -  SIMPLE BOUNDS ARE NOT PERMITTED
84   -  INEQUALITY CONSTRASINTS ARE NOT PERMITTED
85   -  TOO MANY DENSE ROWS : ND TOO SMALL
86   -  LACK OF SPACE : MHA TOO SMALL
87   -  LACK OF SPACE : MHC TOO SMALL
88   -  LINEAR DEPENDENCE OF ACTIVE CONSTRAINTS
89   -  INFEASIBLE SOLUTION
90   -  MAXIMUM NUMBER OF SIMPLEX ITERATIONS : MIS TOO SMALL
91   -  DIFFERENTIAL EQUATION IS UNSTABLE
92   -  MAXIMUM NUMBER OF INTEGRATION STEPS EXCEEDED
93   -  TOO SMALL INTEGRATION STEP
94   -  DIFFERENTIAL EQUATION IS STIFF
95   -  SINGULAR JACOBIAN IN IMPLICIT INTEGRATION METHOD
96   -  LACK OF SPACE IN DIFFERENTIAL EQUATION SOLVER
97   -  LACK OF SPACE : MMAX TOO SMALL
98   -  MAXIMUM NUMBER OF MINOR CYCLES : MIQ TOO SMALL                    '
99   -  MAXIMUM NUMBER OF MAJOR CYCLES : MAQ TOO SMALL                    '
```

6. Special tools of the UFO system

The UFO system contains special tools that facilitate the user's activity. There are tools for checking the correctness of optimization problems and for testing optimization methods.

6.1. Checking external subroutines

The values, gradients, Hessian matrices of the model function or the approximating functions or the constraint functions are specified by using the macrovariables $FMODELF, $GMODELF, $HMODELF or $FMODELA, $GMODELA, $HMODELA or $FMODELC, $GMODELC, $HMODELC, respectively. Sometimes the correctness of these models needs to be checked up. If this is the case, then both the analytical and the numerical differentiation can be compared. The checking of optimization problems can be specified by using the macrovariable $TEST. If $TEST='N', no checking is performed. If $TEST='Y', both the analytical and the numerical differentiation are executed before optimization is started (at the initial starting point) and the derivatives obtained are printed. Only the derivatives that are analytically specified (the first, the second) are checked. If $TEST='A', the checking is performed after the optimization is finished (at the final optimum point). Finally, if $TEST='O', only checking is performed and optimization is not started. The output of checking an optimization problem has the following form:

```
STANDARD TEST OF EXTERNAL SUBROUTINES
-------------------------------------
PROBLEM NO    1

PROBLEM
-------
NF = 2                    KDF =  2  KSF = 1  KCF = 2  NORMF = 0
NA = 0  NAL = 0  MAL = 0  KDA = -1  KSA = 0  KCA = 0  NORMA = 0
NC = 3  NCL = 0  MCL = 0  KDC =  1  KSC = 0  KCC = 2  NORMC = 0

PARAMETERS
----------
X    =   -.2000000000D+01    .1000000000D+01

DERIVATIVES
-----------
FF A =    .9090000000D+03
GF N =   -.2405999822D+04   -.6000004263D+03
GF A =   -.2406000000D+04   -.6000000000D+03
HF N =    .4402000148D+04    .8000000070D+03    .2000000002D+03
HF A =    .4402000000D+04    .8000000000D+03    .2000000000D+03

FC A =   -.1000000000D+01
GC N =    .1000000000D+01    .2000000032D+01
GC A =    .1000000000D+01    .2000000000D+01

FC A =    .5000000000D+01
GC N =   -.4000000070D+01    .9999999930D+00
GC A =   -.4000000000D+01    .1000000000D+01

FC A =    .5000000000D+01
GC N =   -.4000000070D+01    .2000000042D+01
GC A =   -.4000000000D+01    .2000000000D+01
```

Here the letter 'N' indicates a numerical differentiation and the letter 'A' indicates an analytical differentiation.

6.2. Testing optimization methods

The UFO system contains a great number of subroutines (collections of test problems) which serve for testing optimization methods. All of these subroutines begin with letter 'E' (external). The input subroutines have the second letter 'I' and the third letter 'U' or 'L' or 'N' for unconstrained or linearly constrained or nonlinearly constrained problems, respectively. The model specification subroutines have the second letter 'F' or 'A' or 'C' or 'E' or 'Y' for a model function or approximating functions or constraint functions or state functions or initial functions, respectively, and the third letter 'F' or 'G' or 'H' for values or gradients or Hessian matrices. The fourth letter is always 'U' or 'D' or 'S' or 'B' for universal or dense or sparse or partitioned problems, respectively. The last two digits specify individual test problems collections. When we want to carry out a test of the method selected, we use the specifications $COLLECTION='Y' and $NEXT=number_of_test_problems in the input batch file.

Tests corresponding to individual test problems collections are realized by using the following test input files:

`TEST01*.UFO` - tests for unconstrained optimization (25 dense problems from [22], [80]). External subroutines `EIUD01`, `EFFU01`, `EFGU01`, `EFHD01` are used.

`TEST02*.UFO` - tests for the sum of squares minimization (30 dense problems from [111]). External subroutines `EIUD02`, `EAFU02`, `EAGU02`, `EAHD02` are used.

`TEST03*.UFO` - tests for linearly constrained optimization (15 dense problems from [63]). External subroutines `EILD03`, `EFFU03`, `EFGU03` are used.

`TEST04*.UFO` - tests for medium-size linear programming (6 dense problems). External subroutine `EILD04` is used.

`TEST05*.UFO` - tests for medium-size quadratic programming (5 dense problems). External subroutine `EILD05` is used.

`TEST06*.UFO` - tests for minimax (25 dense problems from [101]). External subroutines `EIUD06`, `EAFU06`, `EAGU06`, `EAHD06` are used.

`TEST07*.UFO` - tests for inequality constrained nonlinear programming (34 dense problems from [63]). External subroutines `EIND07`, `EFFU07`, `EFGU07`, `ECFU07`, `ECGU07` are used.

`TEST08*.UFO` - tests for equality constrained nonlinear programming (31 dense problems from [63]). External subroutines `EIND08`, `EFFU08`, `EFGU08`, `ECFU08`, `ECGU08` are used.

`TEST09*.UFO` - tests for unconstrained global optimization (13 problems from [157]). External subroutines `EIUD09`, `EFFU09`, `EFGU09` are used.

`TEST10*.UFO` - tests for unconstrained optimization (15 sparse problems from [98]). External subroutines `EIUS10`, `EFFU10`, `EFGU10`, `EFHS10` are used.

`TEST11*.UFO` - tests for large-scale linear programming (18 sparse problems). External subroutine `EILS11` is used.

`TEST12*.UFO` - tests for large-scale quadratic programming (11 sparse problems). External subroutine `EILS12` is used.

`TEST13*.UFO` - tests for linearly constrained optimization (6 sparse problems). External subroutines `EILS13`, `EFFU13`, `EFGU13` are used.

`TEST14*.UFO` - tests for the sum of functions minimization (22 sparse problems from [98]). External subroutines `EIUB14`, `EAFU14`, `EAGU14` are used.

`TEST15*.UFO` - tests for the sum of squares minimization (24 sparse problems from [98]). External subroutines `EIUB15`, `EAFU15`, `EAGU15` are used.

`TEST16*.UFO` - tests for nonlinear equations solutions (32 dense problems). External subroutines `EIUD16`, `EAFU16`, `EAGU16` are used.

`TEST17*.UFO` - tests for nonlinear equations solutions (30 dense problems). External subroutines `EIUD17`, `EAFU17`, `EAGU17` are used.

TEST18*.UFO - tests for nonlinear equations (32 sparse problems from [98]). External subroutines EIUB18, EAFU18, EAGU18 are used.

TEST19*.UFO - tests for nonsmooth unconstrained optimization (25 dense problems from [101]). External subroutines EIUD19, EFFU19, EFGU19, EFHD19 are used.

TEST20*.UFO - tests for equality constrained sparse nonlinear programming (18 sparse problems from [101]). External subroutines EIUB20, EIUS20, EIND20, EINS20, EFFU20, EFGU20, EAFU20, EAGU20, ECFU20, ECGU20 are used.

TEST21*.UFO - tests for inequality constrained sparse nonlinear programming (1 dense problem). External subroutines EIUS21, EINS21, EFFU21, EFGU21, ECFU21, ECGU21 are used.

TEST22*.UFO - tests for linearly constrained minimax optimization (15 dense problems from [101]). External subroutines EIUD22, EAFU22, EAGU22, EAHD22 are used.

TEST23*.UFO - extended tests for unconstrained optimization (74 dense problems from [98]). External subroutines EIUD23, EFFU23, EFGU23 are used.

TEST24*.UFO - extended tests for the sum of squares minimization (115 dense problems from [22], [80], [98], [111]). External subroutines EIUD24, EAFU24, EAGU24 are used.

TEST25*.UFO - extended tests for the sum of functions minimization (74 sparse problems from [98]). External subroutines EIUB25, EAFU25, EAGU25 are used.

TEST26*.UFO - extended tests for the sum of squares minimization (52 sparse problems from [98]). External subroutines EIUB26, EAFU26, EAGU26 are used.

TEST27*.UFO - extended tests for the sum of squares minimization (82 dense problems from [98]). External subroutines EIUD27, EAFU27, EAGU27 are used.

TEST28*.UFO - extended tests for unconstrained optimization (90 dense problems from [22], [80], [98], [111]). External subroutines EIUD28, EFFU28, EFGU28 are used.

TEST29*.UFO - tests for nonsmooth unconstrained optimization (31 dense problems). External subroutines EIUD29, EFFU29, EFGU29, EFBU29 are used.

TEST30*.UFO - tests for optimization of dynamical systems (4 dense problems). External subroutines EIUD30, EEFU30, EEGU30, EYFU30, EYGU30 are used.

TEST31*.UFO - tests for differential equations (6 dense problems). External subroutines EIUD31, EEFU31 are used.

TEST32*.UFO - tests for the sum of squares minimization (6 dense problems from [82]). External subroutines EIUD32, EAFU32, EAGU32 are used.

TEST33*.UFO - tests for the sum of squares minimization (6 dense problems from [82]). External subroutines EIUD33, EAFU33, EAGU33 are used.

TEST34*.UFO - tests for large-scale linear programming (18 sparse problems). External subroutine EINS20 is used.

TEST35*.UFO - tests for large-scale quadratic programming (8 sparse problems). External subroutine EIQS35 is used.

TEST36*.UFO - extended tests for nonlinear equations solutions (94 dense problems). External subroutines EIUD36, EAFU36, EAGU36 are used.

TEST37*.UFO - extended tests for nonlinear equations solutions (64 dense problems). External subroutines EIUD37, EAFU37, EAGU37 are used.

In these input files, all necessary macrovariables are defined and the external subroutines are called. The external subroutines with the last two digits 01, ..., 37 are briefly described in the text files E01.TXT, ..., E37.TXT. The external subroutines with the last two digits 01, ..., 22 contain original test problems. The remaining external subroutines are their various combinations.

To demonstrate the use of the test input file we perform a test of the sum of squares minimization by using a hybrid method realized as a trust region method. The test input file TEST02.UFO has the following form:

```
$SET(INPUT)
  CALL EIUD02(NF,NA,NAL,X,FMIN,XMAX,NEXT,IEXT,IERR)
  IF(IERR.NE. 0) GO TO $$ENDTEST
$ENDSET
```

```
$SET(FMODELA)
  CALL EAFU02(NF,KA,X,FA,NEXT)
$ENDSET
$NF=12
$NA=400
$KOUT=0
$LOUT=1
$MOUT=-1
$MIT=500
$MFV=10000
$MODEL='AQ'
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=7
$UPDATE='F'
$TOLX='1.0$P-8'
$TOLF='1.0$P-16'
$TOLB='1.0$P-16'
$TOLG='1.0$P-6'
$COLLECTION='Y'
$NEXT=30
$BATCH
$STANDARD
```

The result (screen output) obtained has the following form (each row corresponds to one test problem and the last row is the summary):

```
 CLASS = GN - GM7   UPDATE = F   MODEL = AQ   HESF = D   NF =      12
   1 NIT=   12 NFV=   41 NFG=    0  FV BOUND  F=  .2465190329E-31 G= .222D-15
   2 NIT=   21 NFV=   72 NFG=    0  GRAD TOL  F=  24.49212684     G= .727D-07
   3 NIT=   33 NFV=  102 NFG=    0  FV BOUND  F=  .2035558620E-22 G= .581D-06
   4 NIT=   14 NFV=   47 NFG=    0  FV BOUND  F=  .0000000000     G= .000D+00
   5 NIT=    6 NFV=   21 NFG=    0  GRAD TOL  F=  .1422635916E-15 G= .807D-07
   6 NIT=   12 NFV=   63 NFG=    0  LARGE     F=  62.18109118     G= .111D-05
 US0G01: ( 6) MAXIMUM NUMBER OF REDUCTIONS
   7 NIT=    7 NFV=   32 NFG=    0  FV BOUND  F=  .3435033782E-26 G= .817D-12
   8 NIT=    5 NFV=   24 NFG=    0  GRAD TOL  F=  .4107438653E-02 G= .293D-08
   9 NIT=    1 NFV=    8 NFG=    0  GRAD TOL  F=  .5639663969E-08 G= .177D-07
  10 NIT=  124 NFV=  507 NFG=    0  STEP TOL  F=  43.97292759     G= .148D-02
  11 NIT=   69 NFV=  287 NFG=    0  FV BOUND  F=  .4579830844E-19 G= .660D-06
  12 NIT=   12 NFV=   53 NFG=    0  FV BOUND  F=  .1132907169E-20 G= .599D-10
  13 NIT=   10 NFV=   55 NFG=    0  GRAD TOL  F=  .1686647891E-09 G= .247D-06
  14 NIT=   41 NFV=  218 NFG=    0  FV BOUND  F=  .1029734226E-23 G= .235D-10
  15 NIT=   11 NFV=   62 NFG=    0  GRAD TOL  F=  .1537528024E-03 G= .511D-06
  16 NIT=   24 NFV=  165 NFG=    0  LARGE     F=  42911.10081     G= .248D-03
 US0G01: ( 6) MAXIMUM NUMBER OF REDUCTIONS
  17 NIT=   22 NFV=  139 NFG=    0  GRAD TOL  F=  .2732447349E-04 G= .278D-07
  18 NIT=   17 NFV=  128 NFG=    0  FV BOUND  F=  .2781024866E-18 G= .103D-08
  19 NIT=   13 NFV=  169 NFG=    0  GRAD TOL  F=  .2192388901E-01 G= .375D-07
  20 NIT=    9 NFV=  130 NFG=    0  GRAD TOL  F=  .2361269447E-09 G= .259D-06
  21 NIT=   12 NFV=  171 NFG=    0  FV BOUND  F=  .3845696913E-29 G= .220D-13
  22 NIT=   10 NFV=  143 NFG=    0  GRAD TOL  F=  .5059943798E-09 G= .247D-06
```

```
  23 NIT=   20 NFV=  277 NFG=    0  GRAD TOL  F=  .4392990273E-04 G= .196D-06
  24 NIT=   24 NFV=  334 NFG=    0  GRAD TOL  F=  .3081017822E-03 G= .354D-06
  25 NIT=   10 NFV=  143 NFG=    0  FV BOUND  F=  .1248755103E-25 G= .190D-11
  26 NIT=    9 NFV=  133 NFG=    0  GRAD TOL  F=  .1376154725E-06 G= .491D-07
  27 NIT=    6 NFV=   91 NFG=    0  FV BOUND  F=  .9462508091E-18 G= .139D-08
  28 NIT=    7 NFV=  104 NFG=    0  GRAD TOL  F=  .2107336641E-10 G= .379D-06
  29 NIT=    2 NFV=   39 NFG=    0  GRAD TOL  F=  .7987672380E-13 G= .203D-06
  30 NIT=    5 NFV=   78 NFG=    0  FV BOUND  F=  .2322757795E-26 G= .181D-12
 TOTAL   NIT=     568   NFV=    3836   NFG=       0   NDC=    1420 *    28
         NCG=       0   NRS=       3   NAD=       0   NRM=       0
```

6.3. Interface to the CUTE collection

The CUTE collection [11] is the most famous set of problems for testing optimization methods. These problems are written in the special so-called SIF format. Therefore a SIF decoder is necessary. Such a decoder is part of the CUTE collection, but this version can only be used for a relatively small set of optimization codes (e.g. for the LANCELOT [23] code). Since the UFO system has a special nature, the original SIF decoder had to be modified. This modification consists in replacing the subroutine SDLANC by the subroutine SDUFO and in preparing new interface subroutines `SIUBXX`, `SIUDXX`, `SIUSXX`, `SINBXX`, `SINDXX`, `SINSXX`, `SFFUXX`, `SFGUXX`, `SFFGUX`, `SCFUXX`, `SCGUXX`, `SCFGUX`, `SAFUXX`, `SAGUXX`, `SAFGUX` (instead of `USETUP`, `UFN`, `UGR`, `CSETUP`, `CFN`, `CGR` etc.).

The CUTE collection is not distributed with the UFO system. The SIF files together with SIF decoder subroutines have to be obtained from their authors (they are also available on the INTERNET address `http://www.dci.clrc.ac.uk/Activity/CUTE`). The special interface subroutines listed above are exceptions. They are contained in the library `CUTELIB.LIB`.

If we want to use a SIF file for testing the UFO system methods, it suffices to write the macroinstruction $SIF='SIF_file_name' in the input batch file. For example, if we want to use problem DTOC3 for testing recursive quadratic programming methods for sparse equality constrained nonlinear programming problems, then the problem specification (input batch file) has the following form.

```
$SIF='DTOC3'
$FORM='SE'
$MOUT=2
$BATCH
$STANDARD
```

Here DTOC3 is a name of the SIF file and SE is the form of recursive quadratic programming methods (section 3.20). The problem solution (basic screen output) has this form.

```
    NIC=  0 NIT=   0 NFV=    1 NFG=    8 F=  .000D+00 C=  .150D+02 G=  .000D+00
    NIC=  0 NIT=   1 NFV=    2 NFG=   16 F=  .235D+03 C=  .295D-14 G=  .739D-14
  0 NIC=  0 NIT=   2 NFV=    2 NFG=   16 F=  .235D+03 C=  .295D-14 G=  .739D-14
```

The form of additional output files `P.DIM` and `P.SIF` is shown in section 5.7.

If the sparsity pattern contains a relatively great number of nonzero elements, then default dimensions (e.g. $M, $MA, $MAH, $MHA, $MC, $MCH, $MHC) might be too small and therefore they must be specified in the input batch file.

7. Application of the UFO system (examples)

Before the solution of a given problem, the input file containing the problem description and other specifications for the macroprocessor must usually be prepared. This input file can contain only the macroinstruction $STANDARD (input file `STANDARD.UFO`). Then a full dialogue is processed. However, a more advantageous possibility is to prepare an input file containing the problem description while a method selection is left to the dialogue. Moreover, since a method selection can be made automatically by using knowledge bases coded in UFO templates, the batch mode is recommended.

When writing input file instructions, we have to observe some conventions. Since a control program contains a great number of common variables, we recommend using variables beginning with the letter 'W' for the problem description to avoid their double use. Real variables of this type should be declared at the beginning of the control program by the statement $FLOAT (for example $FLOAT W,W1,W2). Simple integers I,J,K,L need not be declared. We recommend using statement numbers smaller than 10000 for the problem description to avoid their double use.

The basic implementation of the UFO system is in a double precision arithmetic. Therefore usually $FLOAT='REAL*8' and $P='D'. We recommend writing real constants always in the form of $P or D specification (for example 1.0$P 2, 4.0$P-1 or 1.0D 2, 4.0D-1) since the conversions from a single precision, which depend on a compiler, can be incorrect. Instead of constants 0.0D0, 0.5D0, 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 1.0D1, we can use the common variables ZERO, HALF, ONE, TWO, THREE, FOUR, FIVE, TEN, which contain corresponding values.

In the following text, we demonstrate the application of the UFO system to 20 typical problems. Every example consists of the problem description, the problem specification (input file), comments to the problem specification and the problem solution (basic screen output) obtained on a PC computer. All input files contain the necessary data and can be used in the batch mode. These input files are included into the UFO system as the demo-files `PROB01.UFO`,...,`PROB20.UFO`.

7.1. Optimization with simple bounds

a) Problem description:

Suppose we have to find a local maximum of the objective function

$$F(x) = \frac{1}{n!}\left(\prod_{i=1}^{n} x_i\right) - 2$$

with simple bounds $0 \le x_i \le i$ for $1 \le i \le n$, where $n = 5$. The starting point is $x_i = 2$ for $1 \le i \le n$. The solution point is $x_i = i$ for $1 \le i \le n$ and the corresponding maximum value of the objective function is $F = -1.0$.

b) Problem specification (input file):

```
$FLOAT W
$SET(INPUT)
  DO 1 I=1,NF
  X(I)=2.D0 ; XL(I)=0.D0 ; XU(I)=DBLE(I) ; IX(I)=3
1 CONTINUE
$ENDSET
$SET(FGMODELF)
  W=1.D0
  DO 2 I=1,NF
  W=W*X(I)/DBLE(I)
2 CONTINUE
  FF=W-2.D0
  DO 3 I=1,NF
```

```
  GF(I)=W/X(I)
3 CONTINUE
$ENDSET
$IEXT=1
$NF=5
$KBF=2
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values and the simple bounds for variables. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the model function. Because we look for a maximum, we set $IEXT=1.

d) Problem solution (basic screen output):

```
 CLASS = VM - LG1   UPDATE = B   MODEL = FF   HESF = D   NF =       5
     NIT=    0 NFV=    1 NFG=    1  F=  1.866666667     G= .667D-01
     NIT=    1 NFV=    4 NFG=    4  F=  1.550000000     G= .150D+00
     NIT=    2 NFV=    7 NFG=    7  F=  1.200000000     G= .200D+00
     NIT=    3 NFV=    9 NFG=    9  F=  1.000000000     G= .000D+00
   0 NIT=    3 NFV=    9 NFG=    9 NDC=    0 NCG=   0 F= .100D+01 G= .000D+00
 FF =  -.1000000000D+01
 X  =   .1000000000D+01   .2000000000D+01   .3000000000D+01   .4000000000D+01
        .5000000000D+01
```

7.2. Minimization of the sum of squares

a) problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \frac{1}{2}\sum_{i=1}^{m}\left(x_4 e^{-x_1 t_i} + x_5 e^{-x_2 t_i} + x_6 e^{-x_3 t_i} - y_i\right)^2$$

where $m = 20$, $t_i = i/10$ and $y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}$ for $1 \le i \le m$. The starting point is $x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 1, x_5 = 1, x_6 = 1$. The solution point is $x_1 = 1, x_2 = 10, x_3 = 4, x_4 = 1, x_5 = 5, x_6 = 3$ and the corresponding minimum value of the objective function is $F = 0.0$

b) Problem specification (input file):

```
$FLOAT W,WA,WB,WC
$SET(INPUT)
  X(1)=1.D0 ; X(2)=2.D0 ; X(3)=1.D0
  X(4)=1.D0 ; X(5)=1.D0 ; X(6)=1.D0
  DO 1 KA=1,NA
    W=0.1D0*DBLE(KA)
    AM(KA)=EXP(-W)-5.D0*EXP(-1.0D1*W)+3.D0*EXP(-4.D0*W)
  1 CONTINUE
  XMAX=1.D1
```

```
  FMIN=0.D0
$ENDSET
$SET(FMODELA)
  W=0.1D0*FLOAT(KA)
  WA=EXP(-W*X(1))
  WB=EXP(-W*X(2))
  WC=EXP(-W*X(3))
  FA=X(4)*WA-X(5)*WB+X(6)*WC
$ENDSET
$NF=6
$NA=20
$NAL=0
$KBA=1
$MOS1=1
$MOUT=2
$NOUT=1
$MODEL='AQ'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of variables and the vector AM containing values $y_i, 1 \leq i \leq m$. Since the approximating functions contain exponentials, we define the maximum stepsize \$XMAX=10. By using the macrovariable \$FGMODELA we specify analytically the values of the approximating function. The gradients of the approximating functions are computed numerically. For the sum of squares minimization we set \$MODEL='AQ'. The specification \$KBA=1 indicates that the vector AM is used.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = N   MODEL = AQ   HESF = D   NF =       6
     NIT=    0 NFV=    7 NFG=    0  F=  .4652437783     G= .675D+00
     NIT=    1 NFV=   14 NFG=    0  F=  .1956020933     G= .239D+00
     NIT=    2 NFV=   21 NFG=    0  F=  .1720414281     G= .464D+00
     NIT=    3 NFV=   29 NFG=    0  F=  .7987956309E-01 G= .208D+00
     NIT=    4 NFV=   36 NFG=    0  F=  .4318860089E-01 G= .202D+00
     NIT=    5 NFV=   43 NFG=    0  F=  .1259503177E-01 G= .178D+00
     NIT=    6 NFV=   50 NFG=    0  F=  .9578438809E-02 G= .173D+00
     NIT=    7 NFV=   57 NFG=    0  F=  .1688533763E-02 G= .992D-01
     NIT=    8 NFV=   65 NFG=    0  F=  .7788403751E-04 G= .732D-02
     NIT=    9 NFV=   73 NFG=    0  F=  .5354727158E-04 G= .354D-03
     NIT=   10 NFV=   80 NFG=    0  F=  .2747378136E-04 G= .962D-03
     NIT=   11 NFV=   87 NFG=    0  F=  .1684586259E-04 G= .930D-02
     NIT=   12 NFV=   94 NFG=    0  F=  .1118448887E-04 G= .781D-02
     NIT=   13 NFV=  101 NFG=    0  F=  .2628745917E-05 G= .391D-02
     NIT=   14 NFV=  108 NFG=    0  F=  .5403804942E-11 G= .636D-05
     NIT=   15 NFV=  115 NFG=    0  F=  .4024797848E-23 G= .570D-11
   0 NIT=   15 NFV=  115 NFG=    0 NDC=   50 NCG=   0 F= .402D-23 G= .570D-11
 F  =   .4024797848D-23
 X  =   .4000000000D+01   .1000000000D+02   .1000000000D+01   .3000000000D+01
        .5000000000D+01   .1000000000D+01
```

7.3. Minimax approximation

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \max_{1 \le i \le m} |\frac{x_1 + t_i x_2}{1 + t_i x_3 + t_i^2 x_4 + t_i^3 x_5} - y_i|$$

where $m = 21$, $t_i = (i-1)/10 - 1$ and $y_i = e^{-t_i}$ for $1 \le i \le m$. The starting point is $x_1 = 0.5$, $x_2 = 0$, $x_3 = 0$, $x_4 = 0$, $x_5 = 0$. The solution point is $x_1 = 0.9998, x_2 = 0.2536$, $x_3 = -0.7466$, $x_4 = 0.2452$, $x_5 = -0.3749$ and the corresponding minimum value of the objective function is $F = 0.000122371$.

b) Problem specification (input file):

```
$FLOAT W
$SET(INPUT)
  X(1)=0.5D0 ; X(2)=0.0D0 ; X(3)=0.0D0
  X(4)=0.0D0 ; X(5)=0.0D0
$ENDSET
$SET(FMODELA)
  W=0.1D0*DBLE(KA-1)-1.0D0
  FA=(X(1)+W*X(2))/(1.0D0+W*(X(3)+W*(X(4)+W*X(5))))-EXP(W)
$ENDSET
$MODEL='AM'
$NF=5
$NA=21
$NAL=0
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FMODELA we specify analytically the values of the approximating functions. The gradients of the approximating functions are computed numerically. For minimax approximation we set $MODEL='AM'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = AM   HESF = D   NF =       5
     NIT=    0 NFV=    6 NFG=    0  F=  2.218281828     G= .100D+61
     NIT=    1 NFV=   13 NFG=    0  F=  .4253776766     G= .783D+00
     NIT=    2 NFV=   19 NFG=    0  F=  .8271275531E-01 G= .223D+00
     NIT=    3 NFV=   25 NFG=    0  F=  .1256835312E-01 G= .114D+00
     NIT=    4 NFV=   31 NFG=    0  F=  .6693249612E-02 G= .290D-01
     NIT=    5 NFV=   37 NFG=    0  F=  .6441286142E-02 G= .891D-02
     NIT=    6 NFV=   43 NFG=    0  F=  .1963725138E-02 G= .667D-01
     NIT=    7 NFV=   49 NFG=    0  F=  .2258132005E-03 G= .299D-01
     NIT=    8 NFV=   55 NFG=    0  F=  .1226123794E-03 G= .145D-03
     NIT=    9 NFV=   61 NFG=    0  F=  .1223712533E-03 G= .463D-07
   0 NIT=    9 NFV=   61 NFG=    0 NDC=    0 NCG=   0 F= .122D-03 G= .463D-07
```

```
 F  =    .1223712533D-03
 X  =    .9998776287D+00   .2535884404D+00  -.7466075717D+00   .2452015019D+00
        -.3749029100D-01
```

7.4. Nonsmooth optimization

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = -x_1 + 2 * (x_1^2 + x_2^2 - 1) + \frac{7}{4}|x_1^2 + x_2^2 - 1|$$

The starting point is $x_1 = -1,\ x_2 = -1.$. The solution point is $x_1 = 1,\ x_2 = 0$, and the corresponding minimum value of the objective function is $F = -1.0$.

b) Problem specification (input file):

```
$FLOAT W
$SET(INPUT)
  X(1)=-1.D0
  X(2)=-1.D0
$ENDSET
$SET(FGMODELF)
  W=X(1)**2+X(2)**2-1.D0
  FF=-X(1)+2.D0*W+1.75D0*ABS(W)
  W=SIGN(3.5$P 0,W)+4.D0
  GF(1)=W*X(1)-1.D0
  GF(2)=W*X(2)
$ENDSET
$NF=2
$KSF=3
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the objective function. For nonsmooth optimization we set $KSF=3.

d) Problem solution (basic screen output):

```
 CLASS = BM - L11   UPDATE = N   MODEL = FF   HESF = D   NF =       2
     NIT=    0 NFV=    1 NFG=    1  F=  4.750000000     G= .100D+61
     NIT=    1 NFV=    3 NFG=    3  F= -.3788888889     G= .850D+01
     NIT=    2 NFV=    4 NFG=    4  F= -.6061514369     G= .933D+00
     NIT=    3 NFV=    5 NFG=    5  F=  9.250000000     G= .802D+00
     NIT=    4 NFV=    6 NFG=    6  F= -.7284826639     G= .802D+00
     NIT=    5 NFV=    7 NFG=    7  F=  .8708184712     G= .348D+00
```

```
     NIT=    6 NFV=    8 NFG=    8  F= -.8275709577     G= .722D+00
     NIT=    7 NFV=    9 NFG=    9  F= -.8436035754     G= .162D+00
     NIT=    8 NFV=   10 NFG=   10  F= -.9986081259     G= .984D-01
     NIT=    9 NFV=   11 NFG=   11  F= -.9980886265     G= .114D+00
     NIT=   10 NFV=   12 NFG=   12  F= -.9992851884     G= .501D+00
     NIT=   11 NFV=   13 NFG=   13  F= -.9999999867     G= .530D-01
     NIT=   12 NFV=   14 NFG=   14  F= -.9999922205     G= .454D-05
     NIT=   13 NFV=   15 NFG=   15  F= -1.000000000     G= .986D-07
   0 NIT=   13 NFV=   15 NFG=   15 NDC=    0 NCG=   0 F=-.100D+01 G= .986D-07
 FF =  -.1000000000D+01
 X  =   .1000000000D+01   .0000000000D+00
```

7.5. Optimization with linear constraints

a) problem specification:

Suppose we have to find a local minimum of the objective function

$$F(x) = (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$$

over the set given by the linear constraints

$$x_1 + x_2 + x_3 + 4x_4 = 7$$

$$x_3 + 5x_5 = 6$$

The starting point is $x_1 = 10,\ x_2 = 7,\ x_3 = 2,\ x_4 = 3,\ x_5 = 0.8$. The solution point is $x_1 = 1,\ x_2 = 1,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1$ and the corresponding minimum value of the objective function is $F = 0.0$

b) Problem specification (input file):

```
$SET(INPUT)
  X(1)= 1.D1 ; X(2)= 7.D0 ; X(3)= 2.D0
  X(4)=-3.D0 ; X(5)=0.8D0
  IC(1)=5 ; CL(1)=7.D0
  CG(1)=1.D0 ; CG(2)=1.D0 ; CG(3)=1.D0
  CG(4)=4.D0 ; CG(5)=0.D0
  IC(2)=5 ; CL(2)=6.D0
  CG(6)=0.D0 ; CG(7)=0.D0 ; CG(8)=1.D0
  CG(9)=0.D0 ; CG(10)=5.D0
  FMIN =0.D0
$ENDSET
$SET(FMODELF)
  FF=(X(1)-X(2))**2+(X(3)-1.D0)**2+ &
     (X(4)-1.D0)**4+(X(5)-1.D0)**6
$ENDSET
$SET(GMODELF)
  GF(1)= 2.D0*(X(1)-X(2))
  GF(2)=-2.D0*(X(1)-X(2))
  GF(3)= 2.D0*(X(3)-1.D0)
  GF(4)= 4.D0*(X(4)-1.D0)**3
  GF(5)= 6.D0*(X(5)-1.D0)**5
$ENDSET
```

```
$NF=5
$NC=2
$NCL=2
$KBC=1
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables and the types and values of the general linear constraints. Since there are only equality constraints, we can specify only the left sides (CL(1) and CL(2)) and we can set $KBC=1. The specification $FMIN=0 is used, since the objective function value cannot be smaller then zero. By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function.

d) Problem solution (basic screen output):

```
 CLASS = VM - LG1   UPDATE = B   MODEL = FF   HESF = D   NF =       5
     NIT=    0 NFV=    1 NFG=    1  F=  266.0000640     G= .853D+02
     NIT=    1 NFV=    2 NFG=    2  F=  23.36590202     G= .911D+01
     NIT=    2 NFV=    3 NFG=    3  F=  9.082029409     G= .573D+01
     NIT=    3 NFV=    4 NFG=    4  F=  1.180947507     G= .213D+01
     NIT=    4 NFV=    5 NFG=    5  F=  .3278298455     G= .713D+00
     NIT=    5 NFV=    6 NFG=    6  F=  .1969059589     G= .356D+00
     NIT=    6 NFV=    7 NFG=    7  F=  .6972244149E-01 G= .176D+00
     NIT=    7 NFV=    8 NFG=    8  F=  .1791441524E-01 G= .115D+00
     NIT=    8 NFV=    9 NFG=    9  F=  .5643511948E-02 G= .887D-01
     NIT=    9 NFV=   10 NFG=   10  F=  .2190093826E-02 G= .438D-01
     NIT=   10 NFV=   11 NFG=   11  F=  .8383567168E-03 G= .639D-02
     NIT=   11 NFV=   12 NFG=   12  F=  .1933645225E-03 G= .101D-01
     NIT=   12 NFV=   13 NFG=   13  F=  .5495677142E-04 G= .137D-02
     NIT=   13 NFV=   14 NFG=   14  F=  .1463468059E-04 G= .117D-02
     NIT=   14 NFV=   15 NFG=   15  F=  .4872666765E-05 G= .220D-02
     NIT=   15 NFV=   16 NFG=   16  F=  .1203915690E-05 G= .118D-02
     NIT=   16 NFV=   17 NFG=   17  F=  .7914796148E-06 G= .108D-02
     NIT=   17 NFV=   18 NFG=   18  F=  .3667838398E-06 G= .403D-03
     NIT=   18 NFV=   19 NFG=   19  F=  .8991537030E-07 G= .133D-03
     NIT=   19 NFV=   20 NFG=   20  F=  .2725797538E-07 G= .111D-04
     NIT=   20 NFV=   21 NFG=   21  F=  .8421079674E-08 G= .900D-04
     NIT=   21 NFV=   22 NFG=   22  F=  .5133532765E-08 G= .150D-03
     NIT=   22 NFV=   23 NFG=   23  F=  .1721213276E-08 G= .246D-04
     NIT=   23 NFV=   24 NFG=   24  F=  .1038625095E-08 G= .772D-05
     NIT=   24 NFV=   25 NFG=   25  F=  .2779608772E-09 G= .207D-04
     NIT=   25 NFV=   26 NFG=   26  F=  .8280118186E-10 G= .122D-04
     NIT=   26 NFV=   27 NFG=   27  F=  .3182013288E-10 G= .291D-05
     NIT=   27 NFV=   28 NFG=   28  F=  .6044149143E-11 G= .801D-06
   0 NIT=   27 NFV=   28 NFG=   28 NDC=    0 NCG=   0 F= .604D-11 G= .801D-06
 FF =    .6044149143D-11
 X  =    .1003116995D+01   .1003116712D+01   .9999997424D+00   .9984416378D+00
```

```
.1000000052D+01
```

7.6. Minimax approximation with linear constraints

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \max(f_1(x),\ f_2(x),\ f_3(x))$$

with

$$f_1(x) = -\exp(x_1 - x_2)$$
$$f_2(x) = \sinh(x_1 - 1) - 1$$
$$f_3(x) = -\log(x_2) - 1$$

over the set given by the box constraint $x_2 \geq 1/100$ and the linear constraint

$$\frac{5}{100}\, x_1 - x_2 + \frac{1}{2} \geq 0.$$

The starting point is $x_1 = -1,\ x_2 = 1/100$. The solution point is $x_1 = 1.5264,\ x_2 = 0.5763$ and the corresponding minimum value of the objective function is $F = -0.448910$.

b) Problem specification (input file):

```
$SET(INPUT)
  X(1)=-1.D 0 ;                IX(1)=0
  X(2)= 1.D-2 ; XL(2)= 1.D-2 ; IX(2)=1
                CL(1)=-5.D-1 ; IC(1)=1
  CG(1)=5.D-2 ; CG(2)=-1.D 0
$ENDSET
$SET(FMODELA)
  IF (KA.EQ.1) FA=-EXP(X(1)-X(2))
  IF (KA.EQ.2) FA= SINH(X(1)-1.D0)-1.D0
  IF (KA.EQ.3) FA=-LOG(X(2))-1.D0
$ENDSET
$MODEL='AM'
$IEXT=-1
$NF=2
$NA=3
$NC=1
$NCL=1
$KBF=1
$KBC=1
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables and the types and values of both the box constraints and the general linear constraints. Since there are only one-sided constraints, we specify only the left sides (XL(2) and CL(1)) and we can set $KBF=1 and $KBC=1. By using the macrovariable $FMODELA we specify analytically the values of the approximating functions. The gradients of the approximating functions are computed numerically. For minimax approximation we set $MODEL='AM' and $IEXT=-1.

d) Problem solution (basic screen output):

```
CLASS = VM - LQ1   UPDATE = B   MODEL = AM   HESF = D   NF =       2
    NIT=    0 NFV=    3 NFG=    0  F=  3.605170186     G= .100D+61
    NIT=    1 NFV=    6 NFG=    0  F=  1.978554385     G= .363D+00
    NIT=    2 NFV=    9 NFG=    0  F=  .6817245960     G= .487D+00
    NIT=    3 NFV=   12 NFG=    0  F= -.2766372000     G= .595D+00
    NIT=    4 NFV=   15 NFG=    0  F= -.3855019085     G= .356D+00
    NIT=    5 NFV=   18 NFG=    0  F= -.4366539074     G= .936D-01
    NIT=    6 NFV=   21 NFG=    0  F= -.4488815345     G= .159D-02
    NIT=    7 NFV=   24 NFG=    0  F= -.4489107859     G= .358D-05
  0 NIT=    8 NFV=   24 NFG=    0 NDC=    0 NCG=   0 F=-.449D+00 G= .203D-10
F  =  -.4489107859D+00
X  =   .1526434615D+01   .5763217308D+00
```

7.7. Optimization with nonlinear constraints (nonlinear programming)

a) Problem description:

Suppose we have to find a local maximum of the objective function

$$F(x) = x_1 x_3$$

over the set given by the simple bounds $x_1 \geq 0,\ x_3 \geq 0,\ x_5 \geq 0,\ x_7 \geq 0$ and by the nonlinear constraints

$$(x_4 - x_6)^2 + (x_5 - x_7)^2 \geq 4$$

$$\frac{x_3 x_4 - x_2 x_5}{\sqrt{x_2^2 + x_3^2}} \geq 1$$

$$\frac{x_3 x_6 - x_2 x_7}{\sqrt{x_2^2 + x_3^2}} \geq 1$$

$$\frac{x_1 x_3 + (x_2 - x_1) x_5 - x_3 x_4}{\sqrt{(x_2 - x_1)^2 + x_3^2}} \geq 1$$

$$\frac{x_1 x_3 + (x_2 - x_1) x_7 - x_3 x_6}{\sqrt{(x_2 - x_1) + x_3^2}} \geq 1$$

The starting point is $x_1 = 3.0,\ x_2 = 0.0,\ x_3 = 2.0,\ x_4 = -1.5,\ x_5 = 1.5,\ x_6 = 5.0,\ x_7 = 0.0$. The solution point is $x_1 = 4.828,\ x_2 = 0.000, x_3 = 4.828,\ x_4 = 1.000,\ x_5 = 2.414,\ x_6 = 2.414,\ x_7 = 1.000$ and the corresponding minimum value of the objective function is $F = 23.3137$.

b) Problem specification (input file):

```
$FLOAT W
$SET(INPUT)
  X(1)= 3.0D0 ; XL(1)= 0.0D0 ; IX(1)= 1
  X(2)= 0.0D0
  X(3)= 2.0D0 ; XL(3)= 0.0D0 ; IX(3)= 1
  X(4)=-1.5D0
  X(5)= 1.5D0 ; XL(5)= 1.0D0 ; IX(5)= 1
  X(6)= 5.0D0
  X(7)= 0.0D0 ; XL(7)= 1.0D0 ; IX(7)= 1
  CL(1)=4.0D0 ; IC(1)= 1
  CL(2)=1.0D0 ; IC(2)= 1
  CL(3)=1.0D0 ; IC(3)= 1
  CL(4)=1.0D0 ; IC(4)= 1
  CL(5)=1.0D0 ; IC(5)= 1
$ENDSET
$SET(FMODELF)
  FF=X(1)*X(3)
$ENDSET
$SET(FMODELC)
  IF (KC.LE.0) THEN
  ELSE IF (KC.EQ.1) THEN
    FC=(X(4)-X(6))**2+(X(5)-X(7))**2
  ELSE IF (KC.EQ.2) THEN
    W=SQRT(X(2)**2+X(3)**2)
    FC=(X(3)*X(4)-X(2)*X(5))/W
  ELSE IF (KC.EQ.3) THEN
    W=SQRT(X(2)**2+X(3)**2)
    FC=(X(3)*X(6)-X(2)*X(7))/W
  ELSE IF (KC.EQ.4) THEN
    W=SQRT((X(2)-X(1))**2+X(3)**2)
    FC=(X(1)*X(3)+(X(2)-X(1))*X(5)-X(3)*X(4))/W
  ELSE IF (KC.EQ.5) THEN
    W=SQRT((X(2)-X(1))**2+X(3)**2)
    FC=(X(1)*X(3)+(X(2)-X(1))*X(7)-X(3)*X(6))/W
  ENDIF
$ENDSET
$NF=7
$NC=5
$NCL=0
$KBF=1
$KBC=1
$MOUT=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values and simple bounds for variables and the types and values of the general constraints. Since there are only one-sided simple bounds and one-sided general constraints, we set $KBF=1 and $KBC=1. By using the macrovariable $FMODELF we specify analytically the value of the model function. The gradient of the model function is computed

numerically.

d) Problem solution (basic screen output):

```
 CLASS = VM - LN1   UPDATE = N   MODEL = FF   HESF = D   NF =       7
     NIC=  0 NIT=   0 NFV=    8 NFG=    0 F= .600D+01 C= .294D+01 G= .000D+00
     NIC=  0 NIT=   1 NFV=   19 NFG=    0 F= .340D+02 C= .961D+00 G= .267D+01
     NIC=  0 NIT=   2 NFV=   31 NFG=    0 F= .291D+02 C= .807D-01 G= .107D+01
     NIC=  0 NIT=   3 NFV=   43 NFG=    0 F= .247D+02 C= .168D-01 G= .104D+01
     NIC=  0 NIT=   4 NFV=   55 NFG=    0 F= .237D+02 C= .356D-01 G= .704D+00
     NIC=  0 NIT=   5 NFV=   68 NFG=    0 F= .235D+02 C= .472D-01 G= .949D+00
     NIC=  0 NIT=   6 NFV=   81 NFG=    0 F= .233D+02 C= .104D-02 G= .240D+00
     NIC=  0 NIT=   7 NFV=   94 NFG=    0 F= .233D+02 C= .238D-04 G= .795D-01
     NIC=  0 NIT=   8 NFV=  107 NFG=    0 F= .233D+02 C= .394D-05 G= .288D-01
     NIC=  0 NIT=   9 NFV=  120 NFG=    0 F= .233D+02 C= .129D-04 G= .194D-01
     NIC=  0 NIT=  10 NFV=  133 NFG=    0 F= .233D+02 C= .672D-07 G= .885D-03
     NIC=  0 NIT=  11 NFV=  145 NFG=    0 F= .233D+02 C= .187D-08 G= .228D-03
     NIC=  0 NIT=  12 NFV=  158 NFG=    0 F= .233D+02 C= .402D-12 G= .215D-05
   0 NIC=  0 NIT=  13 NFV=  158 NFG=    0 F= .233D+02 C= .402D-12 G= .252D-06
 FF =    .2331370850D+02
 X  =    .4828427122D+01   .2703575049D-06   .4828427128D+01   .1000000135D+01
         .2414213603D+01   .2414213657D+01   .1000000000D+01
```

7.8. Global optimization

a) Problem description:

Suppose we have to find the global minimum of the objective function

$$F(x) = (x_1 - 3)^2(x_1 + 5)^2 + (x_2 - 2)^2(x_2 + 3)^2 - x_1^2 x_2^2$$

over the set given by the inequalities $-12 \leq x_1 \leq 10$ and $-12 \leq x_2 \leq 10$. The starting point is $x_1 = 0$, $x_2 = 0$. The solution point is $x_1 = -7.3300$, $x_2 = -6.4475$ and the global minimum value of the objective function is $F = -806.077$.

b) Problem specification (input file):

```
$SET(INPUT)
  XL(1)=-12.D0 ; XU(1)=10.D0
  XL(2)=-12.D0 ; XU(2)=10.D0
$ENDSET
$SET(FMODELF)
  FF=((X(1)-3.D0)*(X(1)+5.D0))**2+ &
     ((X(2)-2.D0)*(X(2)+3.D0))**2-(X(1)*X(2))**2
$ENDSET
$NF=2
$MOUT=1
$NOUT=1
$EXTREM='G'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds defining the investigated region. By using the macrovariable $FMODELF we specify analytically the value of the model function. The gradient of the model function is computed numerically. Since we require to find the global minimum we set $EXTREM='G'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LI1   UPDATE = B   MODEL = FF   HESF = D   NF =       2
   0  NIT=    55  NFV=   657  NEX=   4  F= -.806D+03

 EXTREM   1 :
 F =  -.8060772623D+03
 X =  -.7329989942D+01  -.6447506480D+01

 EXTREM   2 :
 F =  -.3072281498D+03
 X =  -.6228926443D+01   .4363683088D+01

 EXTREM   3 :
 F =  -.1504539067D+03
 X =   .3836710587D+01  -.4317610609D+01

 EXTREM   4 :
 F =  -.5795091449D+02
 X =   .3368245253D+01   .2827173277D+01
```

7.9. Large scale optimization (sparse Hessian matrix)

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \sum_{i=1}^{n} \left((3 - 2x_i)x_i - x_{i-1} - x_{i+1} + 1\right)^2, \quad x_{n+1} = x_0 = 0$$

where $n = 100$. The starting point is $x_i = -1$ for $1 \leq i \leq n$. The minimum value of the objective function is $F = 0.0$

b) Problem specification (input file):

```
$FLOAT A
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
    J=2*(I-1)+1
    IH(I)=J
    JH(J)=I
    JH(J+1)=I+1
1 CONTINUE
  IH(NF+1)=2*NF
$ENDSET
```

```
$SET(FMODELF)
  FF=0.0D0
  DO 2 J=1,NF
    A=(3.0D0-2.0D0*X(J))*X(J)+1.0D0
    IF (J.GT.1) A=A-X(J-1)
    IF (J.LT.NF) A=A-X(J+1)
    FF=FF+A*A
2 CONTINUE
$ENDSET
$SET(GMODELF)
  GF(1)=0.0D0
  DO 3 J=1,NF
    A=(3.0D0-2.0D0*X(J))*X(J)+1.0D0
    IF (J.GT.1) A=A-X(J-1)
    IF (J.LT.NF) A=A-X(J+1)
    A=A+A
    GF(J)=GF(J)+A*(3.0D0-4.0D0*X(J))
    IF (J.GT.1) GF(J-1)=GF(J-1)-A
    IF (J.LT.NF) GF(J+1)=-A
3 CONTINUE
$ENDSET
$NF=100
$M=1000
$MOUT=2
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of variables and the sparsity pattern of the Hessian matrix. The sparse Hessian matrix, indicated by the statement \$HESF='S', is tridiagonal so that the number of its upper half nonzero elements is 2*NF−1=199. We set \$M=500, since a greater space is needed for sparse matrix processing. By using the macrovariable \$FMODELF we specify analytically the value of the model function. By using the macrovariable \$GMODELF we specify analytically the gradient of the model function.

d) problem solution (basic screen output):

```
 CLASS = MN - GM3   UPDATE = N    MODEL = FF   HESF = S    NF =     100
     NIT=    0 NFV=    1 NFG=    4  F=  410.0000000     G= .380D+02
     NIT=    1 NFV=    2 NFG=    8  F=  51.26265737     G= .123D+02
     NIT=    2 NFV=    3 NFG=   12  F=  6.162928937     G= .694D+01
     NIT=    3 NFV=    4 NFG=   16  F=  .2405291932     G= .141D+01
     NIT=    4 NFV=    5 NFG=   20  F=  .7598786985E-03 G= .122D+00
     NIT=    5 NFV=    6 NFG=   24  F=  .2442230672E-05 G= .636D-02
     NIT=    6 NFV=    7 NFG=   28  F=  .3898601533E-07 G= .822D-03
     NIT=    7 NFV=    8 NFG=   32  F=  .1062176733E-09 G= .481D-04
     NIT=    8 NFV=    9 NFG=   36  F=  .5393507093E-12 G= .401D-05
     NIT=    9 NFV=   10 NFG=   40  F=  .3871095331E-14 G= .271D-06
   0 NIT=    9 NFV=   10 NFG=   40 NDC=    0 NCG=   2 F= .387D-14 G= .271D-06
```

7.10. Large-scale optimization (sparse Jacobian matrix)

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \sum_{i=1}^{n} f_i^A(x)$$

where n=100 and

$$\begin{aligned} f_i^A(x) &= \left((3-2x_i)x_i - x_{i+1} + 1\right)^2 && , \ i = 1 \\ f_i^A(x) &= \left((3-2x_i)x_i - x_{i-1} - x_{i+1} + 1\right)^2 && , 2 \le i \le n-1 \\ f_i^A(x) &= \left((3-2x_i)x_i - x_{i-1} + 1\right)^2 && , \ i = n \end{aligned}$$

The starting point is $x_i = -1$ for $1 \le i \le n$. The minimum value of the objective function is $F = 0.0$ (This problem is equivalent to the previous problem).

b) Problem specification (input file):

```
$FLOAT WA
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
1 CONTINUE
  L=1
  DO 2 I=1,NF
    IAG(I)=L
    IF (I.GT.1) THEN
    JAG(L)=I-1
    L=L+1
    ENDIF
    JAG(L)=I
    L=L+1
    IF (I.LT.NF) THEN
    JAG(L)=I+1
    L=L+1
    ENDIF
2 CONTINUE
  IAG(NF+1)=L
$ENDSET
$SET(FMODELA)
  WA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0
  IF (KA.GT. 1) WA=WA-X(KA-1)
  IF (KA.LT.NF) WA=WA-X(KA+1)
  FA=WA*WA
$ENDSET
$SET(GMODELA)
  WA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0
  IF (KA.GT. 1) WA=WA-X(KA-1)
  IF (KA.LT.NF) WA=WA-X(KA+1)
```

```
  WA=WA+WA
  GA(KA)=WA*(3.0D0-4.0D0*X(KA))
  IF (KA.GT. 1) GA(KA-1)=-WA
  IF (KA.LT.NF) GA(KA+1)=-WA
$ENDSET
$NF=100
$NA=100
$MA=300
$M=600
$MOUT=2
$MODEL='AF'
$JACA='S'
$HESF='B'
$FMIN='0.0$P 0'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of variables and the sparsity pattern of the Jacobian matrix. The sparse Jacobian matrix, indicated by the statement \$JACA='S', is tridiagonal and the number of its nonzero elements is 3*NF−2=298. Therefore we set \$MA=300. Since we use the partitioned Hessian matrix indicated by the statement \$HESF='B', we must specify the number of its nonzero elements (it is 6*NF−2). Therefore we set \$M=600. By using the macrovariable \$FMODELA we specify analytically the values of the approximating functions. By using the macrovariable \$GMODELA we specify analytically the gradients of the approximating functions. For the sum of values minimization we set \$MODEL='AF'.

d) problem solution (basic screen output):

```
 CLASS = VM - LM3   UPDATE = B   MODEL = AF   HESF = B   NF =     100
     NIT=    0 NFV=    1 NFG=    1  F=  410.0000000     G= .380D+02
     NIT=    1 NFV=    2 NFG=    2  F=  56.19847311     G= .887D+01
     NIT=    2 NFV=    3 NFG=    3  F=  33.51055573     G= .597D+01
     NIT=    3 NFV=    5 NFG=    5  F=  8.212655678     G= .709D+01
     NIT=    4 NFV=    7 NFG=    7  F=  1.405304652     G= .448D+01
     NIT=    5 NFV=    8 NFG=    8  F=  .1546538446     G= .147D+01
     NIT=    6 NFV=    9 NFG=    9  F=  .8654952764E-02 G= .276D+00
     NIT=    7 NFV=   10 NFG=   10  F=  .5809356546E-03 G= .115D+00
     NIT=    8 NFV=   11 NFG=   11  F=  .1714867549E-04 G= .187D-01
     NIT=    9 NFV=   12 NFG=   12  F=  .1059237179E-05 G= .423D-02
     NIT=   10 NFV=   13 NFG=   13  F=  .4823394334E-07 G= .105D-02
     NIT=   11 NFV=   14 NFG=   14  F=  .1475859572E-08 G= .217D-03
     NIT=   12 NFV=   15 NFG=   15  F=  .1093278164E-10 G= .129D-04
     NIT=   13 NFV=   16 NFG=   16  F=  .1983979791E-12 G= .131D-05
     NIT=   14 NFV=   17 NFG=   17  F=  .1844060681E-13 G= .667D-06
   0 NIT=   14 NFV=   17 NFG=   17 NDC=    0 NCG=   4 F= .184D-13 G= .667D-06
```

7.11. Large-scale sum of squares optimization (sparse Jacobian matrix)

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \sum_{i=1}^{n} \left(f_i^A(x)\right)^2$$

where $n = 100$ and

$$\begin{aligned}
f_i^A(x) &= (3-2x_i)x_i - x_{i+1} + 1 && , \ i = 1 \\
f_i^A(x) &= (3-2x_i)x_i - x_{i-1} - x_{i+1} + 1 && , 2 \leq i \leq n-1 \\
f_i^A(x) &= (3-2x_i)x_i - x_{i-1} + 1 && , \ i = n
\end{aligned}$$

The starting point is $x_i = -1$ for $1 \leq i \leq n$. The minimum value of the objective function is $F = 0.0$ (This problem is equivalent to the previous problem).

b) Problem specification (input file):

```
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
1 CONTINUE
  L=1
  DO 2 I=1,NA
    IAG(I)=L
    IF (I.GT.1) THEN
    JAG(L)=I-1
    L=L+1
    ENDIF
    JAG(L)=I
    L=L+1
    IF (I.LT.NA) THEN
    JAG(L)=I+1
    L=L+1
    ENDIF
2 CONTINUE
  IAG(NA+1)=L
$ENDSET
$SET(FMODELA)
  I=KA
  FA=(3.0D0-2.0D0*X(I))*X(I)+1.0D0
  IF (I.GT.1) FA=FA-X(I-1)
  IF (I.LT.NA) FA=FA-X(I+1)
$ENDSET
$SET(GMODELA)
  I=KA
  GA(I)=3.0D0-4.0D0*X(I)
  IF (I.GT.1) GA(I-1)=-1.0D0
  IF (I.LT.NA) GA(I+1)=-1.0D0
$ENDSET
$NF=100
$NA=100
$MA=300
```

```
$M=600
$MOUT=2
$MODEL='AQ'
$JACA='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables and the sparsity pattern of the Jacobian matrix. The sparse Jacobian matrix, indicated by the statement $JACA='S', is tridiagonal and the number of its nonzero elements is 3*NF−2=298. Therefore we set $MA=300. Since we do not use the sparse Hessian matrix, we do not specify the number of its nonzero elements. By using the macrovariable $FMODELA we specify analytically the values of the approximating functions. By using the macrovariable $GMODELA we specify analytically the gradients of the approximating functions. For the sum-of-squares minimization we set $MODEL='AQ'.

d) problem solution (basic screen output):

```
 CLASS = GN - GE3   UPDATE = N   MODEL = AQ   HESF = N   NF =     100
     NIT=    0 NFV=    1 NFG=    1  F=  205.0000000     G= .190D+02
     NIT=    1 NFV=    2 NFG=    2  F=  6.361704571     G= .230D+01
     NIT=    2 NFV=    3 NFG=    3  F=  .2275266679E-01 G= .118D+00
     NIT=    3 NFV=    4 NFG=    4  F=  .4866007329E-06 G= .532D-03
     NIT=    4 NFV=    5 NFG=    5  F=  .2346349433E-15 G= .128D-07
   0 NIT=    4 NFV=    5 NFG=    5 NDC=    4 NCG=   0 F= .235D-15 G= .128D-07
```

7.12. Large-scale nonlinear equations

a) Problem description:

Suppose we have to solve the system of the nonlinear equations

$$\begin{aligned} f_i^A(x) &= (3-2x_i)x_i - x_{i+1} + 1 = 0 && , i = 1 \\ f_i^A(x) &= (3-2x_i)x_i - x_{i-1} - x_{i+1} + 1 = 0 && , 2 \le i \le n-1 \\ f_i^A(x) &= (3-2x_i)x_i - x_{i-1} + 1 = 0 && , i = n \end{aligned}$$

where n=100. The starting point is $x_i = -1$ for $1 \le i \le n$. The minimum value of the objective function is $F = 0.0$ (This problem is equivalent to the previous problem).

b) Problem specification (input file):

```
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
1 CONTINUE
$ENDSET
$SET(FMODELA)
  I=KA
  FA=(3.0D0-2.0D0*X(I))*X(I)+1.0D0
```

```
  IF (I.GT.1) FA=FA-X(I-1)
  IF (I.LT.NA) FA=FA-X(I+1)
$ENDSET
$NF=100
$NA=100
$MOUT=2
$MODEL='NE'
$JACA='N'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FMODELA we specify analytically the values of functions in the nonlinear equations. For solving nonlinear equations we set $MODEL='AQ'.

d) problem solution (basic screen output):

```
 CLASS = TN - GE3   UPDATE = N   MODEL = AQ   HESF = N   NF =     100
     NIT=    0 NFV=    1 F=  205.0000000
     NIT=    1 NFV=    7 F=  5.259558960
     NIT=    2 NFV=   14 F=  .1625254125E-01
     NIT=    3 NFV=   21 F=  .2498485030E-06
     NIT=    4 NFV=   28 F=  .6201104372E-16
   0 NIT=    4 NFV=   28 NDC=    4 NCG=   1 F= .620D-16
```

7.13. Large-scale linear programming

a) Problem description:

Suppose we have to find the global maximum of the linear function

$$F(x) = \sum_{i=1}^{n} (-1)^i x_i$$

with simple bounds $-20 \leq x_i \leq 20,\ 1 \leq x_i \leq n$, and linear constraints

$$-x_i + x_{i+1} - x_{i+2} = i,\ \ 1 \leq i \leq n_C$$

where $n = 20$ and $n_C = 18$. The starting point is not given. The maximum value of the linear objective function is $F = 7.0$

b) Problem specification (input file):

```
$SET(INPUT)
  DO 1 I=1,NF
    IX(I)=3
    XL(I)=-2.0D1
    XU(I)=2.0D1
    GF(I)=FLOAT((-1)**I)
1 CONTINUE
  DO 2 KC=1,NC
```

```
    IC(KC)=5
    CL(KC)=FLOAT(KC)
    $SETCG(KC,KC,-1.0D0)
    $SETCG(KC,KC+1, 1.0D0)
    $SETCG(KC,KC+2,-1.0D0)
2 CONTINUE
$ENDSET
$IEXT=1
$NF=20
$NC=18
$NCL=18
$MC=54
$KBF=2
$KBC=1
$MOUT=2
$NOUT=1
$MODEL='FL'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the bounds for variables and the sparsity pattern with numerical values of the constraint Jacobian matrix. We use the procedure UKMCI1. The sparse Jacobian matrix, indicated by the statement \$JACC='S', is tridiagonal and the number of its nonzero elements is 3*(NF−2)=54. Therefore, we set \$MC=54. The option \$MODEL='FL' indicates the linear programming problem.

d) Problem solution (basic screen output):

```
 CLASS = LP - LN1   UPDATE = N   MODEL = FL   HESF = N   NF =      20
 NUMITR=    1 INEW=   20 IOLD=   15 KINP=   0 IU=   48  F= .980D+04
 NUMITR=    2 INEW=   19 IOLD=   20 KINP=   0 IU=   49  F= .208D+04
 NUMITR=    3 INEW=    0 IOLD=   20 KINP=   0 IU=   49  F= .000D+00
 NUMITR=    3 NEL=     3 NREF=    1 KINP=   0 IU=   49  F= .000D+00  ITERL= 1
 NUMITR=    1 INEW=   15 IOLD=   19 KINP=   0 IU=   49  F= .900D+01
 NUMITR=    2 INEW=   20 IOLD=   18 KINP=   0 IU=   48  F= .700D+01
 NUMITR=    3 INEW=    0 IOLD=   18 KINP=   0 IU=   48  F= .700D+01
 NUMITR=    3 NEL=     3 NREF=    1 KINP=   0 IU=   48  F= .700D+01  ITERL= 2
   0 NIC=  0 NIT=   6 NFV=    2 NFG=    0 F=-.700D+01 C= .000D+00 G= .000D+00
 FF =  -.7000000000D+01
 X  =  -.2000000000D+01   .0000000000D+00   .1000000000D+01  -.1000000000D+01
       -.5000000000D+01  -.8000000000D+01  -.8000000000D+01  -.6000000000D+01
       -.5000000000D+01  -.7000000000D+01  -.1100000000D+02  -.1400000000D+02
       -.1400000000D+02  -.1200000000D+02  -.1100000000D+02  -.1300000000D+02
       -.1700000000D+02  -.2000000000D+02  -.2000000000D+02  -.1800000000D+02
```

7.14. Large-scale quadratic programming

a) Problem description:

Suppose we have to find the global minimum of the quadratic function

$$F(x) = \frac{1}{2}(2x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 2x_1x_3 + 2x_3x_4) - x_1 - 3x_2 + x_3 - x_4$$

with simple bounds $x_i \geq 0$ for $1 \leq i \leq n$ and linear constraints

$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 &\leq 5, \\ 3x_1 + x_2 + 2x_3 + x_4 &\leq 4, \\ 2x_2 + 8x_3 &\geq 3. \end{aligned}$$

The starting point is $x_i = 1/2$ for $1 \leq i \leq n$.. The minimum value of the quadratic objective function is $F = -4.681818$.

b) Problem specification (input file):

```
$SET(INPUT)
  DO 1 I=1,NF
  X(I)= 0.5D0
  XL(I)=0.0D0
  IX(I)=1
1 CONTINUE
  GF(1)=-1.0D0; GF(2)=-3.0D0; GF(3)= 1.0D0; GF(4)=-1.0D0
  IH(1)= 1; IH(2)= 3; IH(3)= 4; IH(4)= 6; IH( 5)= 7
  JH(1)= 1; JH(2)= 3; JH(3)= 2; JH(4)= 3; JH( 5)= 4; JH(6)= 4
  HF(1)= 2.0D0; HF(2)=-1.0D0; HF(3)= 1.0D0
  HF(4)= 2.0D0; HF(5)= 1.0D0; HF(6)= 1.0D0
  IC(1)=2; IC(2)=2; IC(3)=1
  CL(1)= 5.0D0; CL(2)= 4.0D0; CL(3)= 3.0D0
  ICG(1)=1; ICG(2)=5; ICG(3)=9; ICG(4)=11
  JCG(1)=1; JCG(2)=2; JCG(3)=3; JCG(4)=4; JCG( 5)=1
  JCG(6)=2; JCG(7)=3; JCG(8)=4; JCG(9)=2; JCG(10)=3
  CG(1)= 1.0D0; CG(2)= 2.0D0; CG(3)= 1.0D0; CG(4)= 1.0D0; CG( 5)= 3.0D0
  CG(6)= 1.0D0; CG(7)= 2.0D0; CG(8)=-1.0D0; CG(9)= 2.0D0; CG(10)= 8.0D0
$ENDSET
$NF=4
$NC=3
$NCL=3
$MC=10
$M=100
$KBF=1
$KBC=1
$MOUT=2
$NOUT=1
$MODEL='FQ'
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds for variables, the sparsity pattern with numerical values of the model Hessian matrix, and the sparsity pattern with numerical values of the

constraint Jacobian matrix. The sparse Hessian matrix is indicated by the statement $HESF='S'. We set $M=100 as a sufficiently large dimension for the working fields. The sparse Jacobian matrix is indicated by the statement $JACC='S' and the number of its nonzero elements is $MC=10. The option $MODEL='FQ' indicates the quadratic programming problem.

d) Problem solution (basic screen output):

```
 CLASS = QP - LN2   UPDATE = N   MODEL = FQ   HESF = S   NF =       4
 MODE =   1   NRED =   0   N =   4   IOLD =   0   INEW =   0
 MODE =   1   NRED =   1   N =   3   IOLD =   0   INEW =   3   ADDITION
 MODE =   1   NRED =   2   N =   2   IOLD =   0   INEW =  -3   ADDITION
 MODE =   1   NRED =   3   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   4   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   4   N =   3   IOLD =   1   INEW =   0   DELETION
 MODE =   1   NRED =   5   N =   2   IOLD =   0   INEW =   1   ADDITION
 MODE =   1   NRED =   6   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   7   N =   2   IOLD =   0   INEW =   0
   0 NIC=  0 NIT=   7 NFV=    2 NFG=    0 F=-.468D+01 C= .000D+00 G= .000D+00
 FF =  -.4681818182D+01
 X  =   .2727272727D+00   .2090909091D+01   .0000000000D+00   .5454545455D+00
```

7.15. Large-scale optimization with linear constraints

a) Problem description:

The problem we have solved is in fact the Hock and Schittkowski problem number 119 (see [44]) which has 16 variables and 8 linear constraints. The minimum value of the objective function is $F = 244.899$.

b) Problem specification (input field):

```
$FLOAT WI,WJ
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=10.0D0; XL(I)=0.0D0; XU(I)=5.0D0; IX(I)=3
1 CONTINUE
  IH( 1)= 1; IH( 2)= 6; IH( 3)=10; IH( 4)=15; IH( 5)=19
  IH( 6)=24; IH( 7)=27; IH( 8)=30; IH( 9)=33; IH(10)=36
  IH(11)=38; IH(12)=40; IH(13)=42; IH(14)=44; IH(15)=45
  IH(16)=46; IH(17)=47;
  JH( 1)= 1; JH( 2)= 4; JH( 3)= 7; JH( 4)= 8; JH( 5)=16
  JH( 6)= 2; JH( 7)= 3; JH( 8)= 7; JH( 9)=10;
  JH(10)= 3; JH(11)= 7; JH(12)= 9; JH(13)=10; JH(14)=14
  JH(15)= 4; JH(16)= 7; JH(17)=11; JH(18)=15;
  JH(19)= 5; JH(20)= 6; JH(21)=10; JH(22)=12; JH(23)=16
  JH(24)= 6; JH(25)= 8; JH(26)=15;
  JH(27)= 7; JH(28)=11; JH(29)=13;
  JH(30)= 8; JH(31)=10; JH(32)=15;
  JH(33)= 9; JH(34)=12; JH(35)=16;
  JH(36)=10; JH(37)=14;
  JH(38)=11; JH(39)=13;
  JH(40)=12; JH(41)=14;
  JH(42)=13; JH(43)=14;
```

```
  JH(44)=14;
  JH(45)=15;
  JH(46)=16;
  DO 2 I=1,NC
    IC(I)=5
2 CONTINUE
  CL(1)= 2.5D0
  CL(2)= 1.1D0
  CL(3)=-3.1D0
  CL(4)=-3.5D0
  CL(5)= 1.3D0
  CL(6)= 2.1D0
  CL(7)= 2.3D0
  CL(8)=-1.5D0
  $SETCG(1, 1, 0.22D0)
  $SETCG(1, 2, 0.20D0)
  $SETCG(1, 3, 0.19D0)
  $SETCG(1, 4, 0.25D0)
  $SETCG(1, 5, 0.15D0)
  $SETCG(1, 6, 0.11D0)
  $SETCG(1, 7, 0.12D0)
  $SETCG(1, 8, 0.13D0)
  $SETCG(1, 9, 1.00D0)
  $SETCG(2, 1,-1.46D0)
  $SETCG(2, 3,-1.30D0)
  $SETCG(2, 4, 1.82D0)
  $SETCG(2, 5,-1.15D0)
  $SETCG(2, 7, 0.80D0)
  $SETCG(2,10, 1.00D0)
  $SETCG(3, 1, 1.29D0)
  $SETCG(3, 2,-0.89D0)
  $SETCG(3, 5,-1.16D0)
  $SETCG(3, 6,-0.96D0)
  $SETCG(3, 8,-0.49D0)
  $SETCG(3,11, 1.00D0)
  $SETCG(4, 1,-1.10D0)
  $SETCG(4, 2,-1.06D0)
  $SETCG(4, 3, 0.95D0)
  $SETCG(4, 4,-0.54D0)
  $SETCG(4, 6,-1.78D0)
  $SETCG(4, 7,-0.41D0)
  $SETCG(4,12, 1.00D0)
  $SETCG(5, 4,-1.43D0)
  $SETCG(5, 5, 1.51D0)
  $SETCG(5, 6, 0.59D0)
  $SETCG(5, 7,-0.33D0)
  $SETCG(5, 8,-0.43D0)
  $SETCG(5,13, 1.00D0)
  $SETCG(6, 2,-1.72D0)
  $SETCG(6, 3,-0.33D0)
  $SETCG(6, 5, 1.62D0)
  $SETCG(6, 6, 1.24D0)
```

```
  $SETCG(6, 7, 0.21D0)
  $SETCG(6, 8,-0.26D0)
  $SETCG(6,14, 1.00D0)
  $SETCG(7, 1, 1.12D0)
  $SETCG(7, 4, 0.31D0)
  $SETCG(7, 7, 1.12D0)
  $SETCG(7, 9,-0.36D0)
  $SETCG(7,15, 1.00D0)
  $SETCG(8, 2, 0.45D0)
  $SETCG(8, 3, 0.26D0)
  $SETCG(8, 4,-1.10D0)
  $SETCG(8, 5, 0.58D0)
  $SETCG(8, 7,-1.03D0)
  $SETCG(8, 8, 0.10D0)
  $SETCG(8,16, 1.00D0)
$ENDSET
$SET(FGMODELF)
  FF=0.0D0
  DO 3 I=1,NF
    GF(I)=0.0D0
3 CONTINUE
  DO 5 I=1,NF
    WI=X(I)*(X(I)+1.0D0)+1.0D0
    K1=IH(I)
    K2=IH(I+1)-1
    DO 4 K=K1,K2
      J=JH(K)
      WJ=X(J)*(X(J)+1.0D0)+1.0D0
      FF=FF+WI*WJ
      GF(I)=GF(I)+(2.0D0*X(I)+1.0D0)*WJ
      GF(J)=GF(J)+WI*(2.0D0*X(J)+1.0D0)
  4 CONTINUE
5 CONTINUE
$ENDSET
$NF=16
$M=200
$NC=8
$NCL=8
$MC=54
$KBF=2
$KBC=1
$MOUT=2
$NOUT=1
$JACC='S'
$HESF='S'
$ADD(INTEGER,'\IH($NF+1)\JH($M)')
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds for variables, the sparsity pattern with

numerical values of the model Hessian matrix, and the sparsity pattern with numerical values of the constraint Jacobian matrix. We use the procedure UKMCI1. The sparse Hessian matrix is indicated by the statement $HESF='S'. The sparse Jacobian matrix is indicated by the statement $JACC='S'. The option $MODEL='FF' indicates a general objective function. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the model function. Since the Hessian structure appears in the gradient definition and the conjugate gradient method is used as default, we have to declare files IH and JH.

d) Problem solution (basic screen output):

```
 CLASS = CD - LC1   UPDATE = N   MODEL = FF   HESF = N   NF =      16
     NIT=    0 NFV=    1 NFG=    1  F=  6318.492582     G= .000D+00
     NIT=    1 NFV=    3 NFG=    3  F=  4814.710061     G= .000D+00
     NIT=    2 NFV=    4 NFG=    4  F=  3584.460640     G= .293D+01
     NIT=    3 NFV=    5 NFG=    5  F=  3506.282573     G= .544D+03
     NIT=    4 NFV=    8 NFG=    8  F=  1887.324185     G= .171D+03
     NIT=    5 NFV=    9 NFG=    9  F=  654.7437615     G= .445D+02
     NIT=    6 NFV=   10 NFG=   10  F=  341.8186624     G= .484D+01
     NIT=    7 NFV=   11 NFG=   11  F=  313.7362910     G= .634D+01
     NIT=    8 NFV=   12 NFG=   12  F=  288.1967005     G= .281D+01
     NIT=    9 NFV=   13 NFG=   13  F=  250.0377363     G= .808D+01
     NIT=   10 NFV=   15 NFG=   15  F=  245.0427300     G= .237D+01
     NIT=   11 NFV=   17 NFG=   17  F=  244.8997708     G= .347D-01
     NIT=   12 NFV=   19 NFG=   19  F=  244.8996976     G= .174D-02
     NIT=   13 NFV=   21 NFG=   21  F=  244.8996975     G= .272D-03
     NIT=   14 NFV=   23 NFG=   23  F=  244.8996975     G= .860D-06
   0 NIT=   14 NFV=   23 NFG=   23 NDC=    0 NCG=   0 F= .245D+03 G= .860D-06
 FF =    .2448996975D+03
 X  =    .3984734987D-01   .7919831513D+00   .2028703518D+00   .8443579338D+00
         .1269906444D+01   .9347387223D+00   .1681961949D+01   .1553008745D+00
         .1567870329D+01  -.4549423462D-14  -.1919595591D-14   .4278076897D-15
         .6602040882D+00   .2269087313D-15   .6742559445D+00  -.4145543553D-14
```

7.16. Large-scale optimization with nonlinear equality constraints

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \sum_{i=1}^{n} \left(f_i^A(x)\right)^2$$

where $n = 100$ and

$$\begin{aligned} f_i^A(x) &= (3-2x_i)x_i - x_{i+1} + 1 && , i = 1 \\ f_i^A(x) &= (3-2x_i)x_i - x_{i-1} - x_{i+1} + 1 && , 2 \le i \le n-1 \\ f_i^A(x) &= (3-2x_i)x_i - x_{i-1} + 1 && , i = n \end{aligned}$$

over the set given by the nonlinear equality constraints

$$8x_i(x_i^2 - x_{i-1}) - 2(1 - x_i) + 4(x_i - x_{i+1}^2) + x_{i-1}^2 - x_{i-2} + x_{i+1} - x_{i+2}^2 = 0,\ 3 \le i \le n-2$$

The starting point is $x_i = -1,\ 1 \le i \le n$. The minimum value of the objective function is $F = 5.29056$.

b) Problem specification (input file):

```
$FLOAT WA,WB
$SET(INPUT)
  DO 1 I=1,NF
  X(I)=-1.D0
1 CONTINUE
  M=0
  IH(1)=1
  DO 2 I=1,NF
  M=M+1
  JH(M)=I
  IF (I.LE.NF-1) THEN
  M=M+1
  JH(M)=I+1
  ENDIF
  IF (I.LE.NF-2) THEN
  M=M+1
  JH(M)=I+2
  ENDIF
  IH(I+1)=M+1
2 CONTINUE
  MC=0
  ICG(1)=1
  DO 3 I=3,NF-2
  MC=MC+1
  JCG(MC)=I-2
  MC=MC+1
  JCG(MC)=I-1
  MC=MC+1
  JCG(MC)=I
  MC=MC+1
  JCG(MC)=I+1
  MC=MC+1
  JCG(MC)=I+2
  ICG(I-1)=MC+1
3 CONTINUE
  DO 4 KC=1,NC
  IC(KC)=5
  CL(KC)=0.D0
4 CONTINUE
$ENDSET
$SET(FMODELF)
  FF=0.D0
  DO 5 J=1,NF
  WA=(3.D0-2.D0*X(J))*X(J)+1.D0
```

```
  IF (J.GT. 1) WA=WA-X(J-1)
  IF (J.LT.NF) WA=WA-X(J+1)
  FF=FF+WA**2
5 CONTINUE
$ENDSET
$SET(GMODELF)
  DO 6 J=1,NF
  GF(J)=0.D0
6 CONTINUE
  DO 7 J=1,NF
  WA=(3.D0-2.D0*X(J))*X(J)+1.D0
  IF (J.GT. 1) WA=WA-X(J-1)
  IF (J.LT.NF) WA=WA-X(J+1)
  WB=2.D0*WA
  GF(J)=GF(J)+WB*(3.D0-4.D0*X(J))
  IF (J.GT. 1) GF(J-1)=GF(J-1)-WB
  IF (J.LT.NF) GF(J+1)=GF(J+1)-WB
7 CONTINUE
$ENDSET
$SET(FMODELC)
  K=KC+2
  FC=8.D0*X(K)*(X(K)**2-X(K-1))-2.D0*(1.D0-X(K))+
 & 4.D0*(X(K)-X(K+1)**2)+X(K-1)**2-X(K-2)+X(K+1)-
 & X(K+2)**2
$ENDSET
$SET(GMODELC)
  K=KC+2
  GC(K-2)=-1.D0
  GC(K-1)=-8.D0*X(K)+2.D0*X(K-1)
  GC(K)=24.D0*X(K)**2-8.D0*X(K-1)+6.D0
  GC(K+1)=-8.D0*X(K+1)+1.D0
  GC(K+2)=-2.D0*X(K+2)
$ENDSET
$NF=100
$M=1500
$NC=96
$NCL=0
$NCE=NC
$MC=500
$KBC=1
$MOUT=2
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Hessian matrix, the sparsity pattern of the constraint Jacobian matrix, and the constraint specifications. The sparse Hessian matrix, indicated by the statement $HESF='S', is tridiagonal so that the number of its upper half nonzero elements is 2*NF−1=199. We set $M=1500, since a greater

space is needed for sparse matrix processing. The sparse Jacobian matrix is indicated by the statement $JACC='S'. Since there are only the equality constraints, we can specify the right hand sides CL(KC), 1 $\leq$ KC $\leq$ NC, and set $KBC=1. The statement $NCE=NC expresses that all constraints are equalities. By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. By using the macrovariable $FMODELC we specify analytically the values of the constraint functions. By using the macrovariable $GMODELC we specify analytically the gradients of the constraint functions.

d) problem solution (basic screen output):

```
 CLASS = MN - LK3   UPDATE = N   MODEL = FF   HESF = S   NF =      100
     NIC=  0 NIT=   0 NFV=    1 NFG=   10 F= .410D+03 C= .280D+02 G= .380D+02
     NIC=  0 NIT=   1 NFV=    2 NFG=   20 F= .374D+04 C= .880D+01 G= .126D+02
     NIC=  0 NIT=   2 NFV=    3 NFG=   30 F= .587D+03 C= .285D+01 G= .489D+01
     NIC=  0 NIT=   3 NFV=    4 NFG=   40 F= .269D+03 C= .127D+01 G= .640D+01
     NIC=  0 NIT=   4 NFV=    5 NFG=   50 F= .966D+02 C= .851D+00 G= .890D+01
     NIC=  0 NIT=   5 NFV=    9 NFG=   60 F=-.557D+02 C= .731D+00 G= .776D+01
     NIC=  0 NIT=   6 NFV=   11 NFG=   70 F= .295D+01 C= .311D+00 G= .385D+01
     NIC=  0 NIT=   7 NFV=   12 NFG=   80 F= .533D+01 C= .490D-01 G= .850D+00
     NIC=  0 NIT=   8 NFV=   13 NFG=   90 F= .530D+01 C= .252D-02 G= .189D+00
     NIC=  0 NIT=   9 NFV=   14 NFG=  100 F= .529D+01 C= .927D-03 G= .514D-01
     NIC=  0 NIT=  10 NFV=   15 NFG=  110 F= .529D+01 C= .190D-03 G= .111D-01
     NIC=  0 NIT=  11 NFV=   16 NFG=  120 F= .529D+01 C= .238D-04 G= .117D-02
     NIC=  0 NIT=  12 NFV=   17 NFG=  130 F= .529D+01 C= .447D-06 G= .220D-04
     NIC=  0 NIT=  13 NFV=   18 NFG=  140 F= .529D+01 C= .168D-09 G= .824D-08
   0 NIC=  0 NIT=  14 NFV=   18 NFG=  140 F= .529D+01 C= .168D-09 G= .824D-08
```

7.17. Large-scale optimization with nonlinear equality and inequality constraints

a) Problem description:

Suppose we have to find a local minimum of the objective function

$$F(x) = \sum_{j=2}^{n-2} \left(x_{i-1}^2 + x_i^2 + 2x_{i+1}^2 + xi + 2^2 - 5x_{i-1} - 5x_i - 21x_{i+1} + 7x_{i+2} \right)$$

where $n = 100$ over the set given by the nonlinear constraints

$$\begin{aligned} x_{j-1}^2 + x_j^2 + x_{j+1}^2 + x_{j+2}^2 + x_{j-1} - x_j + x_{j+1} - x_{j+2} \leq 8, \qquad & \mathrm{mod}(k,3) = 0 \\ x_{j-1}^2 + 2x_j^2 + x_{j+1}^2 + 2x_{i+2}^2 - x_{j-1} - x_{j+2} \leq 10, \qquad & \mathrm{mod}(k,3) = 1 \\ 2x_{j-1}^2 + x_j^2 + x_{j+1}^2 + 2x_{j-1} - x_j - x_{j+2} \leq 5, \qquad & \mathrm{mod}(k,3) = 2 \end{aligned}$$

where $j = 2(\mathrm{div}(k-1,3)+1)$, $1 \leq k \leq 3(n-2)/2$ (div(k,l) is the integer division and mod(k,l) is the remainder after integer division). The starting point is $x_i = 0$, $1 \leq i \leq n$. The minimum value of the objective function is $F = -1303.79$.

b) Problem specification (input file):

```
$SET(INPUT)
   DO 1 I=1,NF
   X(I)=0.0D0
 1 CONTINUE
```

```
   DO 2 I=1,NF
   IH(I)=I
   JH(I)=I
 2 CONTINUE
   IH(NF+1)=NF+1
   MC=0
   KC=0
   DO 5 J=1,NC/3
   CL(KC+1)=8.0D0
   CL(KC+2)=1.0D1
   CL(KC+3)=5.0D0
   I=2*(J-1)
   DO 4 K=1,3
   IC(KC+K)=2
   ICG(KC+K)=MC+1
   DO 3 L=1,4
   JCG(MC+L)=I+L
 3 CONTINUE
   MC=MC+4
 4 CONTINUE
   KC=KC+3
 5 CONTINUE
   ICG(NC+1)=MC+1
$ENDSET
$SET(FMODELF)
   FF=0.0D0
   DO 11 J=2,NF-2,2
   FF=FF+X(J-1)**2+X(J)**2+2.0D0*X(J+1)**2+X(J+2)**2&
-5.0D0*X(J-1)-5.0D0*X(J)-2.1D1*X(J+1)+7.0D0*X(J+2)
11 CONTINUE
$ENDSET
$SET(GMODELF)
   DO 12 I=1,NF
   GF(I)=0.0D0
12 CONTINUE
   DO 13 J=2,NF-2,2
   GF(J-1)=GF(J-1)+2.0D0*X(J-1)-5.0D0
   GF(J)=GF(J)+2.0D0*X(J)-5.0D0
   GF(J+1)=GF(J+1)+4.0D0*X(J+1)-2.1D1
   GF(J+2)=GF(J+2)+2.0D0*X(J+2)+7.0D0
13 CONTINUE
$ENDSET
$SET(FMODELC)
   J=2*((KC-1)/3+1)
   L=MOD(KC,3)
   GO TO (21,22,23), L+1
21 FC=X(J-1)**2+X(J)**2+X(J+1)**2+X(J+2)**2&
+X(J-1)-X(J)+X(J+1)-X(J+2)
   GO TO 24
22 FC=X(J-1)**2+2.0D0*X(J)**2+X(J+1)**2+2.0D0*X(J+2)**2&
-X(J-1)-X(J+2)
   GO TO 24
```

```
23 FC=2.0D0*X(J-1)**2+X(J)**2+X(J+1)**2&
+2.0D0*X(J-1)-X(J)-X(J+2)
24 CONTINUE
$ENDSET
$SET(GMODELC)
   J=2*((KC-1)/3+1)
   L=MOD(KC,3)
   GO TO (25,26,27), L+1
25 GC(J-1)=2.0D0*X(J-1)+1.0D0
   GC(J)=2.0D0*X(J)-1.0D0
   GC(J+1)=2.0D0*X(J+1)+1.0D0
   GC(J+2)=2.0D0*X(J+2)-1.0D0
   GO TO 28
26 GC(J-1)=2.0D0*X(J-1)-1.0D0
   GC(J)=4.0D0*X(J)
   GC(J+1)=2.0D0*X(J+1)
   GC(J+2)=4.0D0*X(J+2)-1.0D0
   GO TO 28
27 GC(J-1)=4.0D0*X(J-1)+2.0D0
   GC(J)=2.0D0*X(J)-1.0D0
   GC(J+1)=2.0D0*X(J+1)
   GC(J+2)=-1.0D0
28 CONTINUE
$ENDSET
$NF=100
$M=1000
$NC=147
$NCL=0
$MC=1000
$KBC=1
$MOUT=2
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Hessian matrix, the sparsity pattern of the constraint Jacobian matrix, and the constraint specifications. The sparse Hessian matrix, indicated by the statement $HESF='S', is diagonal so that the number of its upper half nonzero elements is NF=100. We set $M=1000, since a greater space is needed for sparse matrix processing. The sparse Jacobian matrix is indicated by the statement $JACC='S'. Since there are only one sided inequality constraints, we can specify the right hand sides CL(KC), $1 \leq \mathrm{KC} \leq \mathrm{NC}$, and set $KBC=1 (we can also specify CU(KC), $1 \leq \mathrm{KC} \leq \mathrm{NC}$, and set $KBC=2). By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. By using the macrovariable $FMODELC we specify analytically the values of the constraint functions. By using the macrovariable $GMODELC we specify analytically the gradients of the constraint functions.

d) problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = FF   HESF = S   NF =     100
     NIC=  0 NIT=   0 NFV=    1 NFG=    7 F= .000D+00 C= .000D+00 G= .257D+02
     NIC=  0 NIT=   1 NFV=    2 NFG=   14 F=-.247D+04 C= .112D+02 G= .112D+02
     NIC=  0 NIT=   2 NFV=    3 NFG=   21 F=-.219D+04 C= .107D+02 G= .126D+02
     NIC=  0 NIT=   3 NFV=    4 NFG=   28 F=-.197D+04 C= .696D+01 G= .779D+01
     NIC=  0 NIT=   4 NFV=    5 NFG=   35 F=-.191D+04 C= .594D+01 G= .728D+01
     NIC=  0 NIT=   5 NFV=    6 NFG=   42 F=-.176D+04 C= .402D+01 G= .127D+02
     NIC=  0 NIT=   6 NFV=    7 NFG=   49 F=-.167D+04 C= .306D+01 G= .976D+01
     NIC=  0 NIT=   7 NFV=    8 NFG=   56 F=-.141D+04 C= .797D+00 G= .106D+02
     NIC=  0 NIT=   8 NFV=    9 NFG=   63 F=-.136D+04 C= .311D+00 G= .548D+01
     NIC=  0 NIT=   9 NFV=   10 NFG=   70 F=-.130D+04 C= .000D+00 G= .844D+00
     NIC=  0 NIT=  10 NFV=   11 NFG=   77 F=-.130D+04 C= .902D-02 G= .733D+00
     NIC=  0 NIT=  11 NFV=   12 NFG=   84 F=-.130D+04 C= .704D-02 G= .631D+00
     NIC=  0 NIT=  12 NFV=   13 NFG=   91 F=-.130D+04 C= .289D-02 G= .914D-01
     NIC=  0 NIT=  13 NFV=   14 NFG=   98 F=-.130D+04 C= .294D-03 G= .209D-02
     NIC=  0 NIT=  14 NFV=   15 NFG=  105 F=-.130D+04 C= .368D-04 G= .214D-03
     NIC=  0 NIT=  15 NFV=   16 NFG=  112 F=-.130D+04 C= .839D-06 G= .446D-05
     NIC=  0 NIT=  16 NFV=   17 NFG=  119 F=-.130D+04 C= .830D-08 G= .446D-07
     NIC=  0 NIT=  17 NFV=   18 NFG=  126 F=-.130D+04 C= .825D-10 G= .445D-09
     NIC=  0 NIT=  18 NFV=   19 NFG=  133 F=-.130D+04 C= .822D-12 G= .444D-11
   0 NIC=  0 NIT=  19 NFV=   19 NFG=  133 F=-.130D+04 C= .822D-12 G= .444D-11
```

7.18. Optimization of dynamical systems - general integral criterion

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \frac{1}{2}\int_0^T (y_1^2(t) + y_2^2(t))dt + \frac{1}{2}(y_1^2(T) + y_2^2(T))$$

where $T = 1.5$ and where

$$\frac{dy_1(t)}{dt} = y_2(t), \qquad y_1(0) = x_1$$

$$\frac{dy_2(t)}{dt} = (1 - y_1^2(t))y_2(t) - y_1(t), \quad y_2(0) = 1$$

b) Problem specification (input field):

```
$SET(INPUT)
  X(1)=ZERO
  TA=ZERO
  TAMAX=1.5D 0
$ENDSET
$SET(FMODELF)
  FF=HALF*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODELF)
  DF(1)=YA(1)
  DF(2)=YA(2)
$ENDSET
$SET(FMODELA)
```

```
  FA=HALF*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODELA)
  DA(1)=YA(1)
  DA(2)=YA(2)
$ENDSET
$SET(FMODELE)
  GO TO (1,2) KE
1 FE=YA(2)
  GO TO 3
2 FE=-YA(1)+(ONE-YA(1)**2)*YA(2)
3 CONTINUE
$ENDSET
$SET(DMODELE)
  GO TO (4,5) KE
4 DE(1)=ZERO
  DE(2)=ONE
  GO TO 6
5 DE(1)=-ONE-TWO*YA(1)*YA(2)
  DE(2)=ONE-YA(1)**2
6 CONTINUE
$ENDSET
$SET(FMODELY)
  GO TO (7,8) KE
7 FE=X(1)
  GO TO 9
8 FE=ONE
9 CONTINUE
$ENDSET
$SET(GMODELY)
   GO TO (10,11) KE
10 GE(1)=ONE
   GO TO 12
11 GE(1)=ZERO
12 CONTINUE
$ENDSET
$NF=1
$NE=2
$MODEL='DF'
$MOUT=2
$NOUT=1
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial value of the variable x_1 as well as the initial and terminal times 0 and T, respectively. By using the macrovariables $FMODELA and $DMODELA we specify the subintegral function and by using the macrovariables $FMODELF and $DMODELF we specify the terminal function. The right hand sides of the differential equations are specified by using the

macrovariables $FMODELE and $DMODELE, the while initial values and their derivatives are given by using the macrovariables $FMODELY and $GMODELY. The option $MODEL='DF' indicates a general integral criterion.

d) Problem solution (basic screen output):

```
 CLASS = VM - LI1   UPDATE = B   MODEL = DF   HESF = D   NF =       1
     NIT=    0 NFV=    1 NFG=    0  F=  2.763393900     G= .242D+01
     NIT=    1 NFV=    3 NFG=    0  F=  1.974913643     G= .513D+00
     NIT=    2 NFV=    4 NFG=    0  F=  1.944235577     G= .468D-02
     NIT=    3 NFV=    5 NFG=    0  F=  1.944233169     G= .122D-03
     NIT=    4 NFV=    6 NFG=    0  F=  1.944233168     G= .205D-07
   0 NIT=    4 NFV=    6 NFG=    0 NDC=    0 NCG=   0 F= .194D+01 G= .205D-07
 FF =   .7671653645D+00
 X  =   .6169838477D+00
```

7.19. Optimization of dynamical systems - special integral criterion

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \frac{1}{2}\int_0^T (y_1(t) - 1/(1+t))^2 dt$$

where $T = 1$ and where

$$\frac{dy_1(t)}{dt} = -x_1 y_1(t),\ y_1(0) = x_2$$

b) Problem specification (input field):

```
$SET(INPUT)
  X(1)=2.0D 0
  X(2)=0.0D 0
  TA=ZERO
  TAMAX=ONE
$ENDSET
$SET(FMODELE)
  FE=-X(1)*YA(1)**2
  YE=ONE/(ONE+TA)
  WE=ONE
$ENDSET
$SET(GDMODELE)
  GE(1)=-YA(1)**2
  GE(2)=ZERO
  DE(1)=-TWO*X(1)*YA(1)
$ENDSET
$SET(FMODELY)
  FE=X(2)
$ENDSET
$SET(GMODELY)
  GE(1)=ZERO
```

```
  GE(2)=ONE
$ENDSET
$MODELA='Y'
$NF=2
$NE=1
$MODEL='DQ'
$CLASS='GN'
$UPDATE='F'
$MOUT=2
$NOUT=1
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of the variables x_1 and x_2 as well as the initial and terminal times 0 and T, respectively. The right hand side of the differential equation is specified by using the macrovariables \$FMODELE and \$GDMODELE, while the initial values and their derivatives are given by using the macrovariables \$FMODELY and \$GMODELY. The option \$MODEL='DQ' together with \$MODELA='Y' indicate a special integral criterion.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = F   MODEL = DQ   HESF = D   NF =       2
     NIT=    0 NFV=    1 NFG=    1  F=  .2500000000     G= .693D+00
     NIT=    1 NFV=    3 NFG=    2  F=  .3379696559E-01 G= .114D+00
     NIT=    2 NFV=    5 NFG=    3  F=  .1598937577E-02 G= .613D-02
     NIT=    3 NFV=    7 NFG=    4  F=  .1195750953E-04 G= .225D-02
     NIT=    4 NFV=    9 NFG=    5  F=  .1909017677E-08 G= .300D-04
     NIT=    5 NFV=   11 NFG=    6  F=  .2793082966E-15 G= .200D-08
  0 NIT=    5 NFV=   11 NFG=    6 NDC=    7 NCG=   0 F= .279D-15 G= .200D-08
 F  =   .2793082966D-15
 X  =   .9999999725D+00   .9999999990D+00
```

7.20. Initial value problem for ordinary differential equations

a) Problem description:

Suppose we have to find a solution of the Van der Pol equation

$$\frac{dy_1(t)}{dt} = y_2(t), \qquad y_1(0) = 2$$

$$\frac{dy_1(t)}{dt} = (1 - y_1^2(t))y_2(t) - y_1(t), \quad y_2(0) = 0$$

in the interval $0 \leq t \leq T$ where $T = 20$.

b) Problem specification (input field):

```
$SET(INPUT)
  YA(1)=2.0D0
```

```
  YA(2)=0.0D0
  TA=0.0D0
  TAMAX=1.0D1
$ENDSET
$SET(FMODELE)
  IF (KE.EQ.1) THEN
    FE=YA(2)
  ELSE
    FE=(1.0D0-YA(1)**2)*YA(2)-YA(1)
  ENDIF
$ENDSET
$NA=21
$NE=2
$MODEL='DE'
$MED=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of the variables y_1 and y_2 as well as the initial and terminal times 0 and T, respectively. The right hand sides of the differential equations are specified by using the macrovariable $FMODELE. The option $MODEL='N' indicates integration of a system of ordinary differential equations.

d) Problem solution (basic screen output):

```
 CLASS = NO - NN0   UPDATE = N   MODEL = NO   HESF = N   NF =       0
   0  NSTP=  47  NACC=  35  NREJ=  12  NEV =1228  NEG =   0
    1  AT=   .0000000000D+00
       AY=   .2000000000D+01   .0000000000D+00
    2  AT=   .5000000000D+00
       AY=   .1837719210D+01  -.5345234547D+00
    3  AT=   .1000000000D+01
       AY=   .1508144241D+01  -.7802180795D+00
    4  AT=   .1500000000D+01
       AY=   .1040932817D+01  -.1124320556D+01
    5  AT=   .2000000000D+01
       AY=   .3233165976D+00  -.1832974600D+01
    6  AT=   .2500000000D+01
       AY=  -.8409663144D+00  -.2677481056D+01
    7  AT=   .3000000000D+01
       AY=  -.1866073894D+01  -.1021060289D+01
    8  AT=   .3500000000D+01
       AY=  -.1981111880D+01   .2780991398D+00
    9  AT=   .4000000000D+01
       AY=  -.1741768298D+01   .6246661652D+00
   10  AT=   .4500000000D+01
       AY=  -.1369680461D+01   .8749049693D+00
   11  AT=   .5000000000D+01
       AY=  -.8370774151D+00   .1307088938D+01
```

```
12  AT=   .5500000000D+01
    AY=   .1492104708D-01   .2187559874D+01
13  AT=   .6000000000D+01
    AY=   .1279043316D+01   .2437814560D+01
14  AT=   .6500000000D+01
    AY=   .1981657547D+01   .3834607828D+00
15  AT=   .7000000000D+01
    AY=   .1920152630D+01  -.4358385236D+00
16  AT=   .7500000000D+01
    AY=   .1630052536D+01  -.7027644971D+00
17  AT=   .8000000000D+01
    AY=   .1213232754D+01  -.9878137411D+00
18  AT=   .8500000000D+01
    AY=   .5979891894D+00  -.1543326967D+01
19  AT=   .9000000000D+01
    AY=  -.4129152480D+00  -.2526902993D+01
20  AT=   .9500000000D+01
    AY=  -.1638546749D+01  -.1781241644D+01
21  AT=   .1000000000D+02
    AY=  -.2008340784D+01   .3290648928D-01
```

8. Model examples for demonstration of graphic output

Here we introduce several problem specifications (input files) which demonstrate the application of the graphic screen output. The graphic screen output can be used only on PC computers under the MS DOS system. This possibility is not allowed on the UNIX workstations.

The input files are included into the UFO system as demo-files `PROC01.UFO`,...,`PROC08.UFO`. Corresponding graphic pictures are included in the appendix. The data recommended for graphic pictures are introduced in lines which begin by the directive $REM.

8.1. Nonlinear regression

```
$SET(INPUT)
  LDIM=5
  X(1)=7.0D20
  X(2)=1.0D4
  X(3)=2.2D0
  X(4)=1.01D0
  X(5)=7.0D17
  X(6)=7.0D3
  X(7)=1.6D0
  X(8)=1.01D0
  X(9)=1.0D16
  X(10)=4.0D3
  X(11)=1.5D0
  X(12)=1.01D0
  X(13)=2.0D15
  X(14)=4.0D3
  X(15)=1.3D0
  X(16)=1.01D0
  X(17)=1.0D16
  X(18)=5.0D2
  X(19)=1.2D0
  X(20)=1.01D0
  BETA=5.95D0
  CALL BIUD01(NF,LDIM,NA,X,XL,XU,IX,AT,AM)
$ENDSET
$SET(FMODELA)
  CALL BAFU01(NF,LDIM,KA,NA,X,AT,FA,BETA)
$ENDSET
$SET(GMODELA)
  CALL BAGU01(NF,LDIM,KA,NA,X,AT,GA,BETA)
$ENDSET
$NF=30
$NA=500
$KOUT=0
$KOUT1=0
$KOUT2=100
$KOUT3=1
$LOUT=0
$MOUT=2
$MIT=100
$MODEL='AQ'
```

```
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=7
$UPDATE='F'
$TOLX='1.0$P-16'
$TOLF='1.0$P-16'
$TOLB='1.0$P-16'
$TOLG='1.0$P-6'
$KBA=1
$KBF=2
$GRAPH='Y'
$SCAN='Y'
$BATCH
$ADD(REAL,'\BETA\AT($NA)')
$ADD(SUBROUTINES)
      SUBROUTINE BIUD01(N,L,NA,X,XL,XU,IX,AT,AM)
      INTEGER N,L,NA,IX(N),I,K
      REAL*8 X(N),XL(N),XU(N),AT(NA),AM(NA)
      N=4*L
      K=0
      DO 1 I=1,L
      X(K+1)=LOG(X(K+1))
      XL(K+1)=LOG(1.0D+0)
      XU(K+1)=LOG(1.0D+40)
      IX(K+1)=3
      X(K+2)=LOG(X(K+2))
      XL(K+2)=LOG(1.0D+0)
      XU(K+2)=LOG(1.0D+10)
      IX(K+2)=3
      XL(K+3)=1.0D-2
      XU(K+3)=1.0D+2
      IX(K+3)=3
      XL(K+4)=1.00001D0
      XU(K+4)=1.00000D1
      IX(K+4)=3
      K=K+4
    1 CONTINUE
      OPEN (11,FILE='PROC01.DAT',STATUS='OLD')
      NA=0
    2 NA=NA+1
      READ (11,'(2D14.6)',ERR=3) AT(NA),AM(NA)
      GO TO 2
    3 NA=NA-1
      RETURN
      END
      SUBROUTINE BAFU01(N,L,KA,NA,X,AT,FA,BETA)
      INTEGER N,L,KA,NA
      REAL*8 X(N),AT(NA),FA,Q(8),QD(8)
      REAL*8 ARG,POM,BK,B6INT,BETA
      INTEGER J,K
      COMMON /BCOM/ Q,QD
```

```
      DATA BK /8.617385D-5/
      FA=0.0D 0
      K=0
      DO 1 J=1,L
      ARG=X(K+3)/(BK*AT(KA))
      IF (KA.EQ.1) THEN
      Q(J)=B6INT(AT(KA),ARG)
      FA=FA+EXP(X(K+1)+X(K+2)-ARG)
      ELSE
      POM=X(K+4)-1.0D0
      FA=FA+EXP(X(K+1)+X(K+2)-ARG)*
     & (1.0D0+(POM/BETA)*EXP(X(K+1))*(B6INT(AT(KA),ARG)-
     & Q(J)))**(-X(K+4)/POM)
      ENDIF
      K=K+4
    1 CONTINUE
      RETURN
      END
      SUBROUTINE BAGU01(N,L,KA,NA,X,AT,GA,BETA)
      INTEGER N,L,KA,NA
      REAL*8 X(N),AT(NA),GA(N)
      REAL*8 FAC,ARG,POM,POW,BK,B6INT,B6INTD,A,B,C,D,E,F,G
      REAL*8 Q(8),QD(8),QQ,QQD,BETA
      INTEGER J,K
      COMMON /BCOM/ Q,QD
      DATA BK /8.617385D-5/
      K=0
      DO 1 J=1,L
      FAC=1.0D0/(BK*AT(KA))
      ARG=FAC*X(K+3)
      IF (KA.EQ.1) THEN
      Q(J)=B6INT(AT(KA),ARG)
      QD(J)=FAC*B6INTD(AT(KA),ARG)
      QQ=0.0D0
      QQD=0.0D0
      ELSE
      QQ=B6INT(AT(KA),ARG)-Q(J)
      QQD=FAC*B6INTD(AT(KA),ARG)-QD(J)
      ENDIF
      POM=X(K+4)-1.0D0
      POW=-X(K+4)/POM
      A=EXP(X(K+1)+X(K+2)-ARG)
      B=EXP(X(K+1))
      G=B*QQ
      C=(1.0D0+(POM/BETA)*G)
      D=C**POW
      E=POW*D/C
      F=POM*POM
      GA(K+1)=A*(D+E*(POM/BETA)*G)
      GA(K+2)=A*D
      GA(K+3)=A*(-FAC*D+E*(POM/BETA)*B*QQD)
      GA(K+4)=A*D*(LOG(C)/F+POW*G/(C*BETA))
```

```
      K=K+4
    1 CONTINUE
      RETURN
      END
      FUNCTION B6INT(T,X)
      REAL*8 T,X,B6INT
      REAL*8 A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6
      DATA A1,A2,A3,A4,A5,A6 /41.0D+0, 590.0D+0, 3648.0D+0,
     &                      9432.0D+0, 8028.0D+0, 720.0D+0/
      DATA B1,B2,B3,B4,B5,B6 /42.0D+0, 630.0D+0, 4200.0D+0,
     &                   12600.0D+0, 15120.0D+0, 5040.0D+0/
      B6INT=(1.0D0-(A6+X*(A5+X*(A4+X*(A3+X*(A2+X*(A1+X))))))/
     &     (B6+X*(B5+X*(B4+X*(B3+X*(B2+X*(B1+X))))))*EXP(-X)*T
      RETURN
      END
      FUNCTION B6INTD(T,X)
      REAL*8 T,X,B6INTD
      REAL*8 A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6
      REAL*8 C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,DIS,DEN,DISD,DEND
      DATA A1,A2,A3,A4,A5,A6 /41.0D+0, 590.0D+0, 3648.0D+0,
     &                      9432.0D+0, 8028.0D+0, 720.0D+0/
      DATA B1,B2,B3,B4,B5,B6 /42.0D+0, 630.0D+0, 4200.0D+0,
     &                   12600.0D+0, 15120.0D+0, 5040.0D+0/
      DATA C1,C2,C3,C4,C5 /205.0D+0, 2360.0D+0, 10944.0D+0,
     &                                18863.0D+0, 8028.0D+0/
      DATA D1,D2,D3,D4,D5 /210.0D+0, 2520.0D+0, 12600.0D+0,
     &                              25200.0D+0, 15120.0D+0/
      DIS=A6+X*(A5+X*(A4+X*(A3+X*(A2+X*(A1+X)))))
      DEN=B6+X*(B5+X*(B4+X*(B3+X*(B2+X*(B1+X)))))
      DISD=C5+X*(C4+X*(C3+X*(C2+X*(C1+6.0D0*X))))
      DEND=D5+X*(D4+X*(D3+X*(D2+X*(D1+6.0D0*X))))
      B6INTD=((DIS-DISD+DEND*DIS/DEN)/DEN-1.0D0)*EXP(-X)*T
      RETURN
      END
$ENDADD
$STANDARD
```

8.2. Nonlinear minimax optimization

```
$FLOAT W
$SET(INPUT)
  X(1)=0.5D0 ; X(2)=0.0D0 ; X(3)=0.0D0
  X(4)=0.0D0 ; X(5)=0.0D0
$ENDSET
$SET(FMODELA)
  W=0.1D0*DBLE(KA-1)-1.0D0
  FA=(X(1)+W*X(2))/(1.0D0+W*(X(3)+W*(X(4)+W*X(5))))-EXP(W)
$ENDSET
$MODEL='AM'
$NF=5
$NA=21
```

```
$NAL=0
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$PATH='E'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=3, XL=-5, XU=5
```

8.3. Transformer network design

```
$SET(INPUT)
  NEXT=4
  CALL EIUD06(NF,NA,NAL,X,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU06(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU06(NF,KA,X,GA,NEXT)
$ENDSET
$NF=6
$NA=11
$NAL=0
$MOUT=1
$MODEL='AM'
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$PATH='E'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=3, XL=-5, XU=5
```

8.4. Global optimization

```
$SET(INPUT)
  NEXT=4
  CALL EIUD09(NF,XL,XU,NEXT,IERR)
$ENDSET
$SET(FMODELF)
  CALL EFFU09(NF,X,FF,NEXT)
$ENDSET
$NF=4
```

```
$MOUT=1
$GCLASS=1
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$EXTREM='G'
$BATCH
$STANDARD

$REM VAR=1, XL=-3.8, XU=3.8
$REM VAR=2, XL=-3.8, XU=3.8
```

8.5. Nonsmooth optimization

```
$SET(INPUT)
  NEXT=17
  CALL EIUD19(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
  MA=NF+3
$ENDSET
$SET(FMODELF)
  CALL EFFU19(NF,X,FF,NEXT)
$ENDSET
$SET(GMODELF)
  CALL EFGU19(NF,X,GF,NEXT)
$ENDSET
$KSF=3
$NF=30
$MOUT=-1
$MODEL='FF'
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$PATH='Y'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=4, XL=-5, XU=5
```

8.6. The Rosenbrock function

```
$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
$GRAPH='Y'
$MAP='Y'
$ISO='Y'
$PATH='Y'
$BATCH
$STANDARD
```

8.7. Ordinary differential equations

```
$FLOAT W1,W2,W3,W4
$SET(INPUT)
  TA=0.0D0
  YA(1)=0.994D0
  YA(2)=0.0D0
  YA(3)=0.0D0
  YA(4)=-2.00158510637908252240537862224D0
  TAMAX=17.06521656015796255889172062 49D0
$ENDSET
$SET(FMODELE)
  W1=0.012277471D0
  W2=1.D0-W1
  W3=(YA(1)+W1)**2+YA(2)**2
  W3=W3*SQRT(W3)
  W4=(YA(1)-W2)**2+YA(2)**2
  W4=W4*SQRT(W4)
  GO TO (1,2,3,4) KE
1 FE=YA(3)
  GO TO 5
2 FE=YA(4)
  GO TO 5
3 FE=YA(1)+2*YA(4)-W2*(YA(1)+W1)/W3-W1*(YA(1)-W2)/W4
  GO TO 5
4 FE=YA(2)-2*YA(3)-W2*YA(2)/W3-W1*YA(2)/W4
5 CONTINUE
$ENDSET
$NE=4
$NA=2000
$MODEL='DE'
$SOLVER='DP5'
$MOUT=-1
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$MED=1
$GRAPH='Y'
$BATCH
$STANDARD
```

8.8. The Lorenz attractor

```
$FLOAT W1,W2,W3
$SET(INPUT)
  W1=10.0D0
  W2=28.0D0
  W3=8.0D0/3.0D0
  TA=0.0D0
  YA(1)=-8.0D0
  YA(2)= 8.0D0
  YA(3)=W2-1.0D0
  TAMAX=50.0D0
$ENDSET
$SET(FMODELE)
  GO TO (1,2,3) KE
1 FE=-W1*YA(1)+W1*YA(2)
  GO TO 4
2 FE=-YA(1)*YA(3)+W2*YA(1)-YA(2)
  GO TO 4
3 FE=YA(1)*YA(2)-W3*YA(3)
4 CONTINUE
$ENDSET
$NE=3
$NA=2000
$MODEL='D'
$SOLVER='DP8'
$MOUT=-1
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$MED=1
$GRAPH='Y'
$BATCH
$STANDARD
```

References

[1] M.Altman: Generalized gradient methods of minimizing a functional. Bull. Acad. Polon. Sci., Ser. Sci. Math. Astronom. Phys. 14 (1966) 313-318.

[2] L.Armijo: Minimization of functions having continuous partial derivatives. Pacific J. Math. 16 (1966) 1-3.

[3] M.Al-Baali, R.Fletcher: Variational methods for nonlinear least squares. JOTA 36 (1985) 405-421.

[4] M.C.Biggs: Minimization algorithms making use of nonquadratic properties of the objective function. J. Inst. math. Appl. 8 (1971) 315-327.

[5] M.C.Biggs: A note on minimization algorithms which make use of non-quadratic properties of the objective function. Journal of the Institute of Mathematics and its Applications 12 (1973) 337-338.

[6] P.Bjorstadt, J.Nocedal: Analysis of a new algorithm for one-dimensional minimization. Computing 22 (1979) 93-100.

[7] C.G.E.Boender, A.H.G.Rinnoy Kan: Bayessian stopping rules for multistart global optimization methods. Math. Programming 37 (1987) 59-80.

[8] C.G.E.Boender, A.H.G.Rinnoy Kan, G.T.Timmer, L.Stougie: A stochastic method for global optimization. Mathematical programming 22 (1982) 125-140.

[9] P.T.Boggs, J.W.Tolle: A strategy for global convergence in a sequential quadratic programming algorithm. SIAM Journal on Numerical Analysis 26 (1989) 600-623.

[10] I.D.L.Bogle, J.D.Perkins: A New Sparsity Preserving Quasi-Newton Update for Solving Nonlinear Equations. SIAM Journal on Scientific and Statistical Computations 11 (1990) 621-630.

[11] I.Bongartz, A.R.Conn, N. Gould, P.L.Toint: CUTE: constrained and unconstrained testing environment. Report.

[12] C.G.Broyden: The convergence of a class of double rank minimization algorithms. Part 1 - general considerations. Part 2 - the new algorithm. J. Inst. Math. Appl. 6 (1970) 76-90, 222-231.

[13] C.G.Broyden: A class of methods for solving nonlinear simultaneous equations. Math. of Comput. 19 (1965) 577-593.

[14] K.M.Brown, J.E.Dennis: A new algorithm for nonlinear least squares curve fitting. In: "Mathematical Software" (J.Rice ed.) Academic Press, London 1971.

[15] J.R. Bunch, B.N. Parlett: Direct methods for solving symmetric indefinite systems of linear equations. SIAM J. Numer. Anal. 8 (1971) 639-655.

[16] R.H.Byrd, R.B.Schnabel, G.A.Shultz: Approximate solution of the trust region problem by minimization over two-dimensional subspaces. Math. Programming 40 (1988) 247-263.

[17] R.H.Byrd, J.Nocedal, R.B.Schnabel: Representation of quasi-Newton matrices and their use in limited memory methods. Math Programming 63 (1994) 129-156.

[18] T.F.Chan: Rank revealing QR factorizations. Linear Algebra Appl. 88/89 (1987) 67-82.

[19] T.F.Coleman, B.S.Garbow J.S.Moré: Software for estimation sparse Hessian matrices. ACM Trans. of Math. Software 11 (1985) 363-367.

[20] T.F.Coleman: Large sparse numerical optimization. Springer-Verlag, Berlin, 1984.

[21] T.F.Coleman, B.S.Garbow, J.S.Moré: Software for estimating sparse Jacobian matrices. ACM Trans. of Math. Software 10 (1984) 329-345.

[22] A.R. Conn, N.I.M. Gould, P.L. Toint: Testing a class of methods for solving minimization problems with simple bounds on the variables. Mat. Comput. 50 (1988) 399-430.

[23] A.R.Conn, N. Gould, P.L.Toint: LANCELOT. A Fortran Package for Large-Scale Nonlinear Optimization. Springer Verlag, Berlin 1992.

[24] H.Curry: The method of steepest descent for nonlinear minimization problems. Quart. Appl. Math. 2 (1944) 258-261.

[25] W.C.Davidon: Variable metric method for minimisation. A.E.C. Research and Development Report ANL-5990, 1959.

[26] W.C.Davidon: Optimally conditioned optimization algorithms without line searches. Math. Programming 9 (1975) 1-30.

[27] T.A.Davis, I.S.Duff: An unsymmetric pattern multifrontal methodfor sparse LU factorization. Report No. TR-93-018, CIS Department, University of Florida, Gainesville 1993.

[28] N.Y.Deng, Y.Xiao, F.J.Zhou: Nonmonotonic trust region algorithm. JOTA 76 (1993) 259-285.

[29] R.S.Dembo, T.Steihaug: Truncated-Newton algorithms for large-scale unconstrained minimization. Math. Programming 26 (1983) 190-212.

[30] J.E.Dennis: Some computational techniques for the nonlinear least squares problem. In: "Numerical solution of nonlinear algebraic equations" (G.D.Byrne, C.A.Hall, eds.) Academic Press, London 1974.

[31] J.E.Dennis, H.H.W.Mei: An unconstrained optimization algorithm which uses function and gradient values. Report No. TR-75-246. Dept. of Computer Sci., Cornell University 1975.

[32] J.E.Dennis, R.B.Schnabel: Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall, Englewood Cliffs, New Jersey 1983.

[33] J.E.Dennis, R.E.Welsch: Techniques for Nonlinear Least Squares and Robust Regression. Communications in Statistics B 7 (1978) 345-359.

[34] J.E.Dennis, N.Vicente: On the convergence theory of trust-region-based algorithms for equality-constrained optimization. SIAM J. on Optimization 7 (1997) 927-950.

[35] I.S.Duff, J.K.Reid: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. of Math. Software 9 (1983) 302-325.

[36] R.Fletcher: A new approach to variable metric algorithms. Computer J. 13 (1970) 317-322.

[37] R.Fletcher: A modified Marquardt subroutine for nonlinear least squares. Report No. R-6799, Theoretical Physics Division, A.E.R.E. Harwell, 1971.

[38] R.Fletcher: A general quadratic programming algorithm. J. Inst. Math. Appl. 7 (1971) 76-91.

[39] R.Fletcher: Practical methods of optimization (Second edition). Wiley, New York, 1987.

[40] R.Fletcher, M.J.D.Powell: A rapidly convergent descent method for minimization. Computer J. 6 (1963) 163-168.

[41] R.Fletcher, C.M.Reeves: Function minimization by conjugate gradients. Computer J. 7 (1964) 149-154.

[42] R.Fletcher, C.Xu: Hybrid methods for nonlinear least squares. IMA J. Numer. Anal. 7 (1987) 371-389.

[43] R.Fletcher: Second order corrections for nondifferentiable optimization. In: "Numerical analysis, Dundee 1981" (G.A.Watson ed.), Lecture Notes in Mathematics912, Springer-Verlag, Berlin 1982.

[44] R.Fletcher: Nonlinear programmming without a penalty function. Numerical analysis report NA/171, University of Dundee, 1997.

[45] R.W.Freund, N.M.Nachtigal: A new Krylov-subspace method for symmetric indefinite linear systems. Report No. ORNL/TM-12754, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1994.

[46] R.P.Ge: A filled function method for finding a global minimizer of a function of several variables. Math. Programming 46 (1990) 191-204.

[47] R.P.Ge, Y.F.Qin: A Class of filled functions for finding global minimizers of a function of several variables, JOTA 54 (1987) 241-252.

[48] J.C.Gilbert, C.Lemarechal: Some numerical experiments with variable-storage quasi-Newton algorithms. Math. Programming, 45 (1989) 407-435.

[49] P.E.Gill, W.Murray: A numerically stable form of the simplex algorithm. Linear Algebra Appl. 7 (1973) 99-138.

[50] P.E.Gill, W.Murray: Newton type methods for unconstrained and linearly constrained optimization. Math. Programming 7 (1974) 311-350.

[51] P.E.Gill, W.Murray: Numerically stable methods for quadratic programming. Math. Programming 14 (1978) 349-372.

[52] P.E.Gill, W.Murray, M.H.Wright: Practical optimization. Academic Press, London 1981.

[53] P.E.Gill, M.W.Leonard: Limited-memory reduced-Hessian methods for large-scale unconstrained optimization. Technical Report NA 97-1, Department of Mathematics, University of California, San Diego, 1997.

[54] D.Goldfarb: A family of variable metric algorithms derived by variational means. Math Comput. 24 (1970) 23-26.

[55] D.Goldfarb, A.U.Idnani: A numerically stable dual method for solving strictly convex quadratic programms. Report No. 81-102, Dept.of Computer Sci., The City College of New York, 1981.

[56] A.A.Goldstein: On steepest descent. SIAM J. Control 3 (1965) 147-151.

[57] G.H.Golub, C.F.Van Loan: Matrix computations (second edition). Johns Hopkins University Press, Baltimore 1989.

[58] A.Griewank, P.L.Toint: Partitioned variable metric updates for large scale structured optimization problems. Numer. Math. 39 (1982) 119-137.

[59] L.Grippo, F.Lampariello, S.Lucidi: A nonmonotone line search technique for Newton's method. SIAM J. Numer. Anal. 23 (1986) 707-716.

[60] E.Hairer, S.P.Norsett, G.Wanner: Solving ordinary differential equations I. Springer Series in Computational Mathematics 8, Springer Verlag, Berlin 1987.

[61] S.P.Han: Variable metric methods for minimizing a class of nondifferentiable functions. Math. Programming 20 (1981) 1-13.

[62] M.R.Hestenes, C.M.Stiefel: Methods of conjugate gradient for solving linear systems. J. Res. NBS 49 (1964) 409-436.

[63] W.Hock, K.Schittkowski: Test examples for nonlinear programming codes. Lecture notes in economics and mathematical systems 187. Springer Verlag, Berlin 1981.

[64] R.Hooke, T.A.Jeeves: Direct search solution of numerical and statistical problems. J. Assoc. Comp. Mach. 8 (1961) 212-221.

[65] S.Hoshino: A formulation of variable metric methods. J. Inst. Math. Appl. 10 (1972) 394-403.

[66] M.E.Hribar, J.Nocedal: Improvement to the Horizontal Subproblem. Preprint 1996.

[67] Y.F.Hu, Y.Liu, C.Storey: Efficient generalized conjugate gradient algorithms, Part 1 - theory, Part 2 - implementation. JOTA 69 (1991) 129-137, 139-152.

[68] Y.F.Hu, C.Storey: Motivating quasi-Newton updates by preconditioned conjugate gradient methods. Report No. A150, Dept. of Math. Sci., Loughborough Univ. of Technology, Loughborough 1991.

[69] C.M.Ip, M.J.Todd: Optimal conditioning and convergence in rank one quasi-Newton updates. SIAM J. Numer. Anal. 25 (1988) 206-221.

[70] K.C.Kiwiel: An ellipsoid trust region bundle method for nonsmooth convex minimization. SIAM J. on Control and Optimization 27 (1989) 737-757.

[71] M.Lalee, J.Nocedal, T.Plantenga: On the implementation of an algorithm for large-scale equality constrained optimization. Preprint 1994.

[72] C.L.Lawson, R.J.Hanson: Solving least squares problems. Prentice-Hall, Englewood Cliffs, New Jersey 1974.

[73] A.V.Levy, A.Montalvo: The tunneling algorithm for the global minimization of functions. SIAM Journal Sci. Stat. Comp. 6 (1985) 15-19.

[74] G.Li: Successive column correction algorithms for solving sparse nonlinear systems of equations. Mathematical Programming 43 (1989) 187-207.

[75] P.Lindstrom, P.A.Wedin: A new linesearch algorithm for nonlinear least squares problems. Math. Programming 29 (1984) 268-296.

[76] D.C.Liu, J.Nocedal: On the limited memory BFGS method for large-scale optimization. Math. Programming 45 (1989) 503-528.

[77] L.Lukšan: Dual method for solving a special problem of quadratic programming as a subproblem at linearly constrained nonlinear minimax approximation. Kybernetika 20 (1984) 445-457.

[78] L.Lukšan: An implementation of recursive quadratic programming variable metric methods for linearly constrained nonlinear minimax approximation. Kybernetika 21 (1985) 22-40.

[79] L.Lukšan: Variable metric methods. Unconstrained minimization. Academia, Prague 1990 (in Czech).

[80] L.Lukšan: Computational experience with improved variable metric methods for unconstrained minimization. Kybernetika 26 (1990) 415-431.

[81] L.Lukšan: Computational experience with improved conjugate gradient methods for unconstrained minimization. Kybernetika 28 (1992) 249-262.

[82] L.Lukšan: A note on comparison of statistical software for nonlinear regression. Computational Statistics Quaterly 6 (1991) 321-324.

[83] L.Lukšan: Variationally derived scalling and variable metric updates from the preconvex part of the Broyden family. JOTA 73 (1992) 299-307.

[84] L.Lukšan: Inexact trust region method for large sparse nonlinear least squares. Kybernetika 29 (1993) 305-324.

[85] L.Lukšan: Efficient trust region method for nonlinear least squares. Kybernetika 32 (1996) 105-120.

[86] L.Lukšan: Computational experience with known variable metric updates. JOTA 83 (1994) 27-47.

[87] L.Lukšan: Inexact trust region method for large sparse systems of nonlinear equations. JOTA 81 (1994) 569-590.

[88] L.Lukšan: Combined trust region methods for nonlinear least squares. Kybernetika 32 (1996) 121-138.

[89] L.Lukšan: Hybrid methods for large sparse nonlinear least squares. JOTA 89 (1996) 575-595.

[90] L.Lukšan, J.Vlček: Optimization of dynamical systems. Kybernetika 32 (1996) 465-482.

[91] L.Lukšan, J.Vlček: Simple scaling for variable metric updates. Report No. 611, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1995.

[92] L.Lukšan, J.Vlček: Efficient algorithm for large sparse equality constrained nonlinear programming problems. Technical Report V-652. Prague, ICS AS CR 1996, 17 p.

[93] L.Lukšan, J.Vlček: Truncated trust region methods based on preconditioned iterative subalgorithms for large sparse systems of nonlinear equations. Journal of Optimization Theory and Applications, Vol. 95, 1997, No. 3, pp. 637-658.

[94] L.Lukšan, J.Vlček: A bundle-Newton method for nonsmooth unconstrained minimization. Mathematical Programming, Vol. 83, 1998, pp 373-391.

[95] L.Lukšan, J.Vlček: Computational experience with globally convergent descent methods for large sparse systems of nonlinear equations. Optimization Methods and Software, Vol. 8, 1998, pp. 201-223.

[96] L.Lukšan, J.Vlček: Indefinitely preconditioned inexact Newton method for large sparse equality constrained nonlinear programming problems. Numerical Linear Algebra with Applications, Vol. 5, 1998, pp. 219-247.

[97] L.Lukšan, J.Vlček: Globally convergent variable metric method for convex nonsmooth unconstrained minimization. To appear in Journal of Optimization Theory and Applications.

[98] L.Lukšan, J.Vlček: Subroutines for testing large sparse and partially separable unconstrained and equality constrained optimization problems. Technical Report V-767. Prague, ICS AS CR 1998.

[99] L.Lukšan, J.Vlček: Numerical experience with iterative methods for equality constrained nonlinear programming problems. Technical Report V-779. Prague, ICS AS CR, 2000.

[100] L.Lukšan, J.Vlček: NDA: Algorithms for nondifferentiable optimization. Technical Report V-797. Prague, ICS AS CR, 2000.

[101] L.Lukšan, J.Vlček: Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report V-798. Prague, ICS AS CR 2000.

[102] L.Lukšan, J.Vlček: Preconditioning of saddle-point systems. In: Proceedings of the conference SIMONA 2000, Liberec 2000.

[103] M.M.Mäkelä, J.Neittaanmäki: Nonsmooth Optimization. World Scientific Publishing Co. Ltd. London 1992.

[104] E.S.Marwill: Exploiting sparsity in Newton-like methods. Ph.D. Thesis, Cornell University, Ithaca 1978.

[105] J.M.Martinez: A quasi-Newton method with modification of one column per iteration. Computing 33 (1984) 353-362.

[106] J.M.Martinez, M.C.Zambaldi: An inverse column-updating method for solving large-scale nonlinear systems of equations. Optimization Methods and Software 1 (1992) 129-140.

[107] J.Miao: Two infeasible interior-point predictor-corrector algorithms for linear programming. SIAM J. Optimization 6 (1996) 587-599.

[108] R.B.Mifflin, J.L.Nazareth: The least-prior deviation quasi-Newton update. Technical Report, Dept. of Pure and Applied Math., Washington State University, Pullman 1991.

[109] S.Mizuno: Polynomiality of infeasible-interior-point algorithms for linear programming. Math Programming 67 (1994) 109-119.

[110] J.J.Moré: The Levenberg-Maquardt algorithm. Implementation and theory. In: "Numerical Analysis" (G.A.Watson ed.) Springer Verlag, Berlin 1978.

[111] J.J.Moré, B.S.Garbow, K.E.Hillström: Testing unconstrained optimization software. ACM Trans. Math. Software 7 (1981) 17-41.

[112] J.J.Moré, D.C.Sorensen: Computing a trust region step. Report No. ANL-81-83, Argonne National Laboratory. 1981.

[113] S.G.Nash, A.Sofer: Preconditioning reduced matrices. SIAM J. on Matrix Analysis and Application 17 (1996) 47-68.

[114] J.A.Nelder, R.Mead: A simplex method for function minimization. Computer J. 7 (1965) 308-313.

[115] J.Nocedal: Updating quasi-Newton Matrices with limited storage. Math. Comput. 35 (1980) 773-782.

[116] J.Nocedal, Y.Yuan: Combining trust region and line search techniques. To appear.

[117] S.S.Oren, D.G.Luenberger: Self scaling variable metric (SSVM) algorithms. Part 1 - criteria and sufficient condition for scaling a class of algorithms. Part 2 - implementation and experiments. Management Sci. 20 (1974) 845-862, 863-874.

[118] S.S.Oren, E. Spedicato: Optimal conditioning of self scaling variable metric algorithms. Math Programming 10 (1976) 70-90.

[119] C.C.Paige and M.A.Saunders: LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software 8 (1982) 43-71.

[120] E.Polak, G.Ribiére: Note sur la convergence des methodes de directions conjugées. Revue Francaise Inform. Mech. Oper. 16-R1(1969) 35-43.

[121] M.J.D.Powell: A new algorithm for unconstrained optimization. In: "Nonlinear Programming" (J.B.Rosen O.L.Mangasarian, K.Ritter eds.) Academic Press, London 1970.

[122] M.J.D.Powell: Convergence properties of a class of minimization algoritms. In "Nonlinear Programming 2" (O.L.Mangasarian, R.R.Meyer, S.M.Robinson eds.). Academic Press, London 1975.

[123] M.J.D.Powell: Restart procedures of the conjugate gradient method. Math. Programming 12 (1977) 241-254.

[124] M.J.D.Powell: A fast algorithm for nonlinearly constrained optimization calculations. In:"Numerical analysis" (G.A.Watson ed.). Springer Verlag, Berlin 1977.

[125] M.J.D.Powell: Convergence properties of algorithms for nonlinear optimization. Report No. DAMPT 1985/NA1, University of Cambridge, 1985.

[126] H.Ramsin, P.A.Wedin: A Comparison of Some Algorithms for the Nonlinear Least Squares Problem. BIT 17 (1977) 72-90.

[127] A.H.G.Rinnoy Kan, C.G.E.Boender, G.T.Timmer: A stochastic approach to global optimization. Computational Mathematical Programming, NATO ASI Series Vol. F15.

[128] A.H.G.Rinnoy Kan, G.T.Timmer: Stochastic global optimization methods, Part I: Clustering methods, Part II: Multi-level methods. Math. Programming 39 (1987), North-Holland 26-56, 57-78.

[129] Y.Saad, M.Schultz: GMRES a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM Journal on Scientific and Statistical Computations 7 (1986) 856-869.

[130] R.B.Schnabel, E.Eskow: A new Choleski factorization. SIAM J. Sci. Stat. Comput. 11 (19900, 1136-1158.

[131] L.K.Schubert: Modification of a quasi-Newton method for nonlinear equations with a sparse Jacobian. Math. of Comput. 24 (1970) 27-30. (1991) 75-100.

[132] D.F.Shanno: Conditioning of quasi-Newton methods for function minimization. Math. Comput. 24 (1970) 647-656.

[133] D.F.Shanno, K.J.Phua: Matrix conditioning and nonlinear optimization. Math. Programming 14 (1978) 144-160.

[134] E.Spedicato: A class of rank-one positive definite quasi-Newton updates for unconstrained minimization. Math. Operationsforsch. Statist. Ser. Optimization 14 (1963) 61-70.

[135] E.Spedicato, M.T.Vespucci: Numerical experiments with variations of the Gauss-Newton algorithm for nonlinear least squares. JOTA 57 (1988) 323-339.

[136] E.Spedicato, J.Greenstadt: On some classes of variationally derived quasi-Newton methods for systems of nonlinear algebraic equations. Numer. Math. 29 (1978) 363-380.

[137] T.Steihaug: Local and superlinear convergence for truncated iterated projections methods. Math. Programming 27 (1983) 176-190.

[138] T.Steihaug: The conjugate gradient method and trust regions in large-scale optimization. SIAM J. Numer. Anal. 20 (1983) 626-637.

[139] N.M.Steen, G.D.Byrne: The problem of minimizing nonlinear functionals. I. Least squares. In: "Numerical solution of nonlinear algebraic equations" (G.D.Byrne, C.A.Hall, eds.) Academic Press, London 1974.

[140] G.W.Stewart: A modification of Davidon's minimization method to accept difference approximations of derivatives. J. ACM 14 (1967) 72-83.

[141] M.Šiška: Macroprocessor BEL for the UFO system (version 1989). Report No. 448 (in Czech), Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1989.

[142] M.Šiška: Macroprocessor UFO (version 1990). Report No. 484 (in Czech), Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.

[143] P.L.Toint: On sparse and symmetric matrix updating subject to a linear equation. Math of Comp. 31 (1977) 954-961.

[144] P.L.Toint: On large scale nonlinear least squares calculations. SIAM J. Sci. Stat. Comput. 8 (1987) 416-435.

[145] C.H.Tong: A comparative study of preconditioned Lanczos methods for nonsymmetric linear systems. Report No. SAND91-8240B, Sandia National Laboratories, Livermore 1992.

[146] D.Touati-Ahmed, C.Storey: Efficient hybrid conjugate gradient techniques. JOTA 64 (1990), pp. 379-397.

[147] M.Tůma: A quadratic programming algorithm for large and sparse problems. Kybernetika 27 (1991) 155-167.

[148] M.Tůma: Sparse fractioned variable metric updates. Report No. 497, Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.

[149] M.Tůma: Intermediate fill-in in sparse QR decomposition. In: "Linear Algebra for Large Scale and Real-Time Applications", (B.de Moor, G.H.Golub, M.Moonen, eds.), Kluwer Academic Publishers, London 1993, pp. 475-476.

[150] P.S.Vassilevski, D.Lazarov: Preconditioning mixed finite element saddle-point elliptic problems. Numerical Linear Algebra with Applications 3 (1996) 1-20.

[151] H.A.Van der Vorst: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 15 (1992) 631-644.

[152] J.Vlček: Bundle algorithms for nonsmooth unconstrained optimization. Report No. 608, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1994.

[153] J.Vlček, L.Lukšan: Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. Technical Report B 8/1999. Department of Mathematical Information Technology. University of Jyväskylä, 1999.

[154] H.Yabe, T.Takahashi: Factorized quasi-Newton methods for nonlinear least squares problems. Math. Programming 51 (1991) 75-100.

[155] Y.Zhang, R.P.Tewarson: Least-change updates to Choleski factors subject to nonlinear quasi-Newton condition. IMA J. Numer. Anal. 7 (1987) 509-521.

[156] Y.Zhang, R.P.Tewarson: Quasi-Newton algorithms with updates from the preconvex part of Broyden's family. IMA J. Numer. Anal. 8 (1988) 487-509.

[157] A.Žilinskas, A.A.Thorn: Global optimization. Springer Verlag, Berlin 1990.

Index of macrovariables

Appendix A. Demonstration of the text dialogue mode

Suppose that the model function has the form

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

(the Rosenbrock function) and the starting point is $x_1 = -1.2$ and $x_2 = 1.0$. If we type the statement `UFOGO` (without batch input file specification), then the following questions (which we supplement together with answers) appear on the screen.

```
UFO PREPROCESSOR    V.3.1.

                    ? INPUT ( ) ?
   USER SUPPLIED INPUT:

        HERE THE STARTING POINT, BOUNDS FOR VARIABLES,
        TYPES OF CONSTRAINTS, THE STRUCTURE OF SPARSE PROBLEM,
        AND OTHER INPUT DATA HAVE TO BE SPECIFIED.

X(1) = -1.2D0; X(2) = 1.0D0
                    ? GRAPH (N) ?
   SPECIFICATION OF GRAPHICAL OUTPUT
      N - GRAPHICAL OUTPUT SUPPRESSED
      Y - GRAPHICAL OUTPUT REQUIRED
                    ? DISPLAY (N) ?
   SPECIFICATION OF EXTENDED SCREEN OUTPUT
      N - EXTENDED SCREEN OUTPUT SUPPRESSED
      Y - EXTENDED SCREEN OUTPUT REQUIRED
                    ? MODEL (FF) ?
   TYPE OF OBJECTIVE FUNCTION
      FF - GENERAL FUNCTION
      FL - LINEAR FUNCTION
      FQ - QUADRATIC FUNCTION
      AF - SUM OF FUNCTIONS
      AQ - SUM OF SQUARES
      AP - SUM OF POWERS
      AM - MINIMAX
      DF - DIFFERENTIAL SYSTEM WITH GENERAL INTEGRAL CRITERION
      DQ - DIFFERENTIAL SYSTEM WITH INTEGRAL OF SQUARES
      DE - DIFFERENTIAL EQUATIONS
      NE - NONLINEAR EQUATIONS
      NO - MODEL IS NOT SPECIFIED
```

? NF (0) ?

NUMBER OF VARIABLES

2

? IEXT (0) ?

TYPE OF EXTREMUM
0 - MINIMUM
1 - MAXIMUM

? FMODELF (*) ?

MODEL OF OBJECTIVE FUNCTION

FF = <FORTRAN_EXPRESSION>

FF = 1.0D2*(X(1)**2 - X(2))**2 + (X(1) - 1.0D0)**2

? GMODELF (*) ?

MODEL OF GRADIENT OF OBJECTIVE FUNCTION

GF(1) = <FORTRAN_EXPRESSION>
GF(2) = <FORTRAN_EXPRESSION>
.
.
GF(NF) = <FORTRAN_EXPRESSION>

? HMODELF (*) ?

MODEL OF HESSIAN MATRIX

HF(1) = <FORTRAN_EXPRESSION>
HF(2) = <FORTRAN_EXPRESSION>
.
.
HF(M) = <FORTRAN_EXPRESSION>

? KCF (2) ?

COMPLEXITY OF THE OBJECTIVE FUNCTION
1 - EASY COMPUTED FUNCTION
2 - REASONABLE BUT NOT EASY COMPUTED FUNCTION
3 - EXTREMELY COMPLICATED FUNCTION

? KSF (1) ?

SMOOTHNESS OF THE OBJECTIVE FUNCTION:
1 - SMOOTH AND WELL-CONDITIONED FUNCTION
2 - SMOOTH BUT ILL-CONDITIONED FUNCTION
3 - NONSMOOTH FUNCTION

? HESF (D) ?

TYPE OF HESSIAN MATRIX:
- D - DENSE
- S - SPARSE WITH KNOWN (GENERAL) STRUCTURE
- N - HESSIAN MATRIX IS NOT USED

? KBF (0) ?

TYPE OF SIMPLE BOUNDS:
- 0 - NO SIMPLE BOUNDS
- 1 - ONE SIDED SIMPLE BOUNDS
- 2 - TWO SIDED SIMPLE BOUNDS

? KBC (0) ?

TYPE OF GENERAL CONSTRAINTS:
- 0 - NO GENERAL CONSTRAINTS
- 1 - ONE SIDED GENERAL CONSTRAINTS
- 2 - TWO SIDED GENERAL CONSTRAINTS

? EXTREM (L) ?

TYPE OF OPTIMIZATION
- L - LOCAL OPTIMIZATION
- G - GLOBAL OPTIMIZATION

? NORMF (0) ?

SCALING SPECIFICATION FOR VARIABLES:
- 0 - NO SCALING IS PERFORMED
- 1 - SCALING FACTORS ARE DETERMINED AUTOMATICALLY
- 2 - SCALING FACTORS ARE SUPPLIED BY USER

? INPUTDATA (N) ?

READ INPUT VALUES OF X (Y OR N)

? TEST (N) ?

STANDARD TEST OF EXTERNAL SUBROUTINES:
- N - NO TEST
- Y - PERFORM TEST BEFORE SOLUTION
- A - PERFORM TEST AFTER SOLUTION
- O - PERFORM TEST WITHOUT SOLUTION

? KOUT (0) ?

LEVEL OF TEXT FILE OUTPUT:
ABS(KOUT)=0 - NO PRINT OR PAPER SAVING PRINT
ABS(KOUT)=1 - STANDARD PRINT OF ITERATIONS
ABS(KOUT)=2 - ADDITIONAL PRINT OF STEPSIZE SELECTION
ABS(KOUT)=3 - ADDITIONAL PRINT OF DIRECTION DETERMINATION
AND VARIABLE METRIC UPDATE
ABS(KOUT)=4 - ADDITINAL PRINT OF CONSTRAINT HANDLING
ABS(KOUT)=5 - ADDITIONAL PRINT OF NUMERICAL DIFFERENTIATION
KOUT<0 - ADDITIONAL PRINT OF DATA AND OPTIONS IN THE HEADING

? LOUT (1) ?

LEVEL OF TEXT FILE OUTPUT:
0 - NO PRINT
1 - COPY OF THE BASIC SCREEN OUTPUT
-1 - PAPER SAVING PRINT

? MOUT (-2) ?

LEVEL OF BASIC SCREEN OUTPUT:
ABS(MOUT)=0 - NO OUTPUT
ABS(MOUT)=1 - FINAL OUTPUT
ABS(MOUT)=2 - ADDITIONAL OUTPUT IN EACH ITERATION
ABS(MOUT)=3 - ADDITIONAL FINAL OUTPUT OF LINEAR OR
QUADRATIC PROGRAMMING
ABS(MOUT)=4 - ADDITIONAL OUTPUT IN EACH ITERATION
OF LINEAR OR QUADRATIC PROGRAMMING
MOUT<0 - FINAL OUTPUT WITH TERMINATION CRITERION

1

? NOUT (0) ?

LEVEL OF BASIC SCREEN OUTPUT:
0 - BASIC FINAL OUTPUT
1 - EXTENDED FINAL OUTPUT

1

? MSELECT (1) ?

SELECTION OF OPTIMIZATION METHOD
1 - AUTOMATICAL SELECTION OF METHOD
2 - MANUAL SELECTION OF METHOD
3 - MANUAL SELECTION OF METHOD AND IMPORTANT PARAMETERS
4 - MANUAL SELECTION OF METHOD AND ALL PARAMETERS

? LAPACK (N) ?

USE LAPACK SUBROUTINES
N - ONLY UFO SUBROUTINES
Y - CONNECTION TO LAPACK POSSIBLE

? OUTPUT ( ) ?

USER SUPPLIED OUTPUT:

HERE THE RESULTS OBTAINED IN THE OPTIMIZATION PROCESS
CAN BE USED FOR ADDITIONAL COMPUTATIONS AND FOR A
SPECIFIC OUTPUT.

? OUTPUTDATA (N) ?

WRITE OUTPUT VALUES OF X (Y OR N)

UFO PREPROCESSOR STOP

Each question is represented by one frame which contains the contents of the question (name of the macrovariable that has to be defined), the default value (in brackets) and an explanation of the requirement. If no default value is wanted, the corresponding value or text has to be typed. The dialogue can be ended by pressing the key <! > .

The result of the UFO preprocessor action is the following control program (reported in a slightly shortened form) consisting of global declarations, input specifications, problem definition, method realization and control variables adjustement:

```
*
*     -------------------
*     GLOBAL DECLARATIONS
*     -------------------
*
      INTEGER ITIME
      INTEGER IMD
      INTEGER IX(1)
      REAL*8 UXVDOT
      REAL*8 GF(2)
      REAL*8 X(2)
      REAL*8 HD(2)
      REAL*8 HF(2*(2+1)/2)
      REAL*8 S(2)
      REAL*8 ALF
      REAL*8 BET
      REAL*8 XO(2)
      REAL*8 GO(2)
      INTEGER IMB
*
*     commons placed here were omitted
*     since they require a large space
*
*     -------------------
*     END OF DECLARATIONS
*     -------------------
*
      OPEN (2,FILE='P.OUT',STATUS='UNKNOWN')
      OPEN (3,FILE='P.DIM',STATUS='UNKNOWN')
      CALL UYCLEA
      CALL UYINTP
*
```

```
*     ----------
*     METHOD (1)
*     ----------
*
      CALL UYINT1
      CALL UOTES1('VM','L','I','1','B','FF
     &    ','D',NF)
      X(1)=-1.2D0
      X(2)=1.0D0
      CALL UYCLST
      WRITE(3,'(/''PROBLEM: NEXT ='',I8)') NEXT
      IF (NF.GT.2) THEN
      CALL UOERR2('UZLMIN',80,NF,2)
      CALL UOERR4
      ITERM=-80
      TXFU='LACK SPC'
      ENDIF
      WRITE(3,'(''NUMBER OF VARIABLES:        NF  ='',I8)') NF
      M=NF*(NF+1)/2
      IF (ITERM.LT.0) STOP
      CALL UYTIM1(ITIME)
      NDECF=0
      IF (ITERM.NE.0) GO TO 11200
      CALL UO0FU1(NF,NA,NAL,MAL,NC,NCL,MCL,EPS0,EPS1,EPS2,EPS3,EPS4,EPS5
     &    ,EPS6,EPS7,EPS8,EPS9,ETA0,ETA1,ETA2,ETA3,ETA4,ETA5,ETA6,ETA7,E
     &    TA8,ETA9,ALF1,ALF2,ALF3,BET1,BET2,BET3,GAM1,GAM2,GAM3,DEL1,DEL
     &    2,DEL3,RPF1,RPF2,RPF3,RGF1,RGF2,RGF3,FMIN,XMAX,XDEL,REXP,MET,M
     &    ET1,MET2,MET3,MES,MES1,MES2,MES3,MOT,MOT1,MOT2,MOT3,MOS,MOS1,M
     &    OS2,MOS3,MEP,MEP1,MEP2,MEP3,MEG,MEG1,MEG2,MEG3,MEX,MEX1,MEX2,M
     &    EX3,MED,MED1,MED2,MED3,MCG,MCG1,MFP,MFP1,MPF,MPF1,MGF,MGF1,MLP
     &    ,MLP1,MQP,MQP1,MEQ,MEQ1,MSG,MSG1,KSF,KCF,KSA,KCA,KSC,KCC,KTERS
     &    ,INITD,INITS,INITH,IREM,IADD,IRES1,IRES2,MRED,IRAN1,IRAN2,ISAM
     &    1,ISAM2,KINP,IPRN)
*
*     ----------------------
*     VARIABLE METRIC METHOD
*     TEMPLATE : U1FDU1
*     ----------------------
*
      ASSIGN 11130 TO IMD
      CALL UYPRO1('UXFU',1)
      CALL UYPRO2(FMIN,FO)
11110 CONTINUE
*
*     -----------------
*     MODEL DESCRIPTION
*     -----------------
*
11600 CALL UF1F01(NF,GF,GF,FF,F)
      GOTO (11640,11610,11620) ISB+1
11610 CONTINUE
      ASSIGN 11710 TO IMB
```

```
11700 CONTINUE
      NFV=NFV+1
      FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
      GOTO IMB
11710 CONTINUE
      GOTO 11600
11620 CONTINUE
      CALL UF0GS2(NF,X,IX,X,GF,FF,HD,R,SNORM,1.0D-15,1.0D-15,2,1)
      GOTO (11600,11630) ISB+1
11630 CONTINUE
      ASSIGN 11910 TO IMB
      GOTO 11700
11910 CONTINUE
      GO TO 11620
11640 CONTINUE
*
*     ------------------------
*     END OF MODEL DESCRIPTION
*     ------------------------
*
      GO TO IMD
11130 CONTINUE
      CALL UYTRUG(NF,N,X,GF,GF,UMAX,GMAX)
      CALL UO2FU3(NF,M,NA,NC,X,GF,HF,X,X,F,DMAX,GMAX)
      CALL UYFUT1(N,F,FO,UMAX,GMAX,DMAX,ITES,IRES1,IRES2,INEW)
      IF(ITERM.NE.0) GOTO 11190
11140 CONTINUE
      ASSIGN 11140 TO IMD
      CALL UUDSD1(N,HF,1)
      GOTO (11150,11110) ISB+1
11150 CONTINUE
      IF(ITERM.NE.0) GOTO 11190
      CALL UYCPSD(NF,IX,HF,HD,MCG1)
      CALL UYTRUH(NF,N,X,HF)
*
*     ------------------------
*     DIRECTION DETERMINATION
*     TEMPLATE : UDGLG1
*     ------------------------
*
      CALL UOD1D1
      IF (IDECF.LT.0) THEN
      IDECF=9
      INF=0
      ENDIF
      TDXX(1:4)='INV '
      IF (IDECF.EQ.0) THEN
*
*     INVERSION
*
      ALF=ETA2
      CALL UXDPGF(N,HF,INF,ALF,BET)
```

```
      CALL UXDPGI(N,HF)
      NDECF=NDECF+1
      IDECF=9
      ELSE IF (IDECF.EQ.9) THEN
      ELSE
      ITERD=-1
      TDXX='BAD DEC9'
      CALL UOERR1('UDDLI1',1)
      GO TO 12630
      ENDIF
      GNORM=SQRT(UXVDOT(N,GF,GF))
*
*     NEWTON LIKE STEP
*
      CALL UXDSMM(N,HF,GF,S)
      CALL UXVNEG(N,S,S)
      INITD=MAX(ABS(INITD),1)
      ITERD=1
      IF(INF.EQ.0) THEN
      TDXX(5:8)=' POS'
      ELSEIF(INF.LT.0) THEN
      TDXX(5:8)=' ZER'
      ELSE
      TDXX(5:8)=' NEG'
      ENDIF
      SNORM=SQRT(UXVDOT(N,S,S))
      NRED=INF
      CALL UOD1D5(ALF,BET,INF)
12630 CALL UOD1D2(N,GF,S)
*
*     ----------------------------
*     END OF DIRECTION DETERMINATION
*     ----------------------------
*
      IF (KD.GT.0) P=UXVDOT(N,GF,S)
      CALL UD1TL1(NF,N,GF,S,EPS0,ALF1,ALF2,R,P,GNORM,SNORM,RMIN,RMAX,XMA
     &    X,XDEL,MES,INITD,INITH)
      IF(ITERM.NE.0) GOTO 11190
      IF(IREST.NE.0) GOTO 11140
      CALL UYTRUS(NF,X,X,XO,GF,GO,S,S,RO,FP,FO,F,PO,P,CMAX,CMAXO)
11170 CONTINUE
      ASSIGN 11170 TO IMD
      CALL US0L01(EPS1,RO,RP,R,FO,FP,F,PO,PP,FMIN,FMAX,PAR1,PAR2,RMAX,RM
     &    IN,SNORM,MODE,KTERS,MES,MES1,MES2,INITS,MRED)
      GOTO (11174,11172) ISB+1
11172 CONTINUE
      CALL UXVDIR(NF,R,S,XO,X)
      GOTO 11110
11174 CONTINUE
      IF (ITERS.LE.0) THEN
      CALL UYZER0(NF,X,XO,R,F,FO,FF,P,PO,MOT3)
      IF(IDIR.EQ.0) THEN
```

```
      CALL UYRES1(TSXX)
      CALL UYSET1
      GO TO 11140
      ELSE IF (MOT3.EQ.0) THEN
      CALL UYSET1
      GO TO 11140
      ELSE
      ITERD=0
      ENDIF
      ENDIF
      IF(KD.GT.LD) THEN
      ASSIGN 11180 TO IMD
      GO TO 11110
      ENDIF
11180 CONTINUE
      TXFU=TUXX
      CALL UYUPSD(NF,X,IX,XO,GF,GO,HD,P,MCG1)
      CALL UYTRUD(NF,X,X,XO,GF,GO,R,F,FO,P,PO,DMAX)
      CALL UUDBI1(N,HF,S,XO,GO,R,PO,F,FO,P,1.0D 60,8)
      IF(IDIR.EQ.0) THEN
      IF(ITERH.NE.0) CALL UYRES1('UPDATE  ')
      GOTO 11130
      ELSE
      GOTO 11140
      ENDIF
11190 CONTINUE
      IF(ITERM.LT.0) TXFU=TDXX
      CALL UYEPI1(1)
11200 CONTINUE
      CALL UOERR3(KOUT,LOUT,MOUT,ITERM,IER)
      CALL UO1FU2(NF,NA,NC,X,X,X,X,FF,F,FO,DMAX,GMAX,XMAX,EPS0,EPS1,EPS2
     &    ,EPS3,EPS4,EPS5,BET1,BET2,GAM1,GAM2,ETA1,ETA2,MET,MET1,MET2,ME
     &    T3,MOT,MOT1,MOT2,MOT3,MES,MES1,MES2,MES3,MOS,MOS1,MOS2,MOS3,IN
     &    ITD,INITS,INITH,IRES1,KTERS,IPRN)
13599 CONTINUE
*
*     -----------------
*     END OF METHOD (1)
*     -----------------
*
      CALL UYTIM2(ITIME)
      CLOSE (2)
      CLOSE (3)
      END
*
*     ------------------------
*     INITIATION OF METHOD (1)
*     ------------------------
*
      SUBROUTINE UYINT1
*
*     commons placed here were omitted
```

```
*     since they require a large space
*
      REAL*8 XDELS,RPF1S,RPF2S,RPF3S,RGF1S,RGF2S,RGF3S
      COMMON/UMCLST/ XDELS,RPF1S,RPF2S,RPF3S,RGF1S,RGF2S,RGF3S
      ETA0=1.0D-15
      ETA9=1.0D 60
      ITR=6
      IRD=5
      IWR=2
*
*     many other assignments follow which were
*     omitted since they require a large space
*
      END
*
*     ---------------------
*     INITIATION OF PROBLEM
*     ---------------------
*
      SUBROUTINE UYINTP
*
*     commons placed here were omitted
*     since they require a large space
*
      NF=2
      IEXT=0
      KCF=2
      KSF=1
      KBF=0
      KBC=0
      NORMF=0
      KDF=0
      KDA=-1
      KDC=-1
      KDE=-1
      KDY=-1
      END
*     ----------------------------------------
*     BROYDEN CLASS OF VARIABLE METRIC UPDATES
*     TEMPLATE : UUDBI1
*     ----------------------------------------
      SUBROUTINE UUDBI1(N,H,S,XO,GO,R,PO,F,FO,P,ETA9,MET)
*
*     commons placed here were omitted
*     since they require a large space
*
      REAL*8 H(N*(N+1)/2),S(N),XO(N),GO(N),R,PO,ETA9
      REAL*8 F,FO,P
      REAL*8 AA,CC
      COMMON /UMFUN1/ AA,CC
      REAL*8 UXVDOT,UNFUN1
      REAL*8 DIS,POM,POM3,POM4,A,B,C,GAM,RHO,PAR
```

```
      REAL*8 DEN
      INTEGER IUPDT
      LOGICAL L1,L3
      EXTERNAL UNFUN1
      IF (MET.LE.0) GO TO 22
      CALL UOU1D1(N,XO,GO)
      IF (IDECF.NE.9) THEN
      ITERH=-1
      TUXX='BAD DEC9'
      CALL UOERR1('UUDBI2',1)
      GO TO 22
      ENDIF
      L1=ABS(4).GE.3.OR.ABS(4).EQ.2.AND.NIT.EQ.KIT
      L3=.NOT.L1
*
*     DETERMINATION OF THE PARAMETERS A, B, C
*
      B=UXVDOT(N,XO,GO)
      IF (B.LE.ZERO) THEN
      ITERH=2
      TUXX='B - NEG.'
      GO TO 22
      ENDIF
      CALL UXDSMM(N,H,GO,S)
      A=UXVDOT(N,GO,S)
      IF (A.LE.ZERO) THEN
      ITERH=1
      TUXX='A - NEG.'
      GO TO 22
      ENDIF
      IF(MET.GE.4.OR.L1) THEN
      IF (ITERD.NE.1) THEN
      MET=1
      C=ZERO
      ELSE
      C=-R*PO
      IF (C.LE.ZERO) THEN
      ITERH=3
      TUXX='C - NEG.'
      GO TO 22
      ENDIF
      ENDIF
      ELSE
      C=ZERO
      ENDIF
*
*     DETERMINATION OF THE PARAMETER RHO (NONQUADRATIC PROPERTIES)
*
      IF (FO-F+P.EQ.0) THEN
      RHO=ONE
      ELSE
      RHO=HALF*B/(FO-F+P)
```

```
      ENDIF
      IF(RHO.LE.1.0D-2) RHO=ONE
      IF(RHO*1.0D-2.GE.ONE) RHO=ONE
      AA=A/B
      CC=C/B
      IUPDT=0
      IF (L1) THEN
*
*     DETERMINATION OF THE PARAMETER GAM (SELF SCALING)
*
      IF (C.LE.ZERO) THEN
      PAR=A/B
      POM3=0.8D 0
      POM4=8.0D 0
      ELSE
      PAR=SQRT(A/C)
      POM3=0.7D 0
      POM4=6.0D 0
      ENDIF
      GAM=RHO/PAR
      IF (NIT.NE.KIT) THEN
      L3=GAM.LT.POM3.OR.GAM.GT.POM4
      ENDIF
      ENDIF
      IF (L3) THEN
      GAM=ONE
      PAR=RHO/GAM
      ENDIF
*
*     NEW UPDATE
*
      POM=ONE/(AA*CC)
      IF (POM.LT.ONE) THEN
      DEN=MAX(POM+1.0D-15,SQRT(C/A))
      POM=(DEN-POM)/(ONE-POM)
      TUXX='NEW     '
      GO TO 20
      ENDIF
   17 CONTINUE
*
*     BFGS UPDATE
*
      POM=ONE
      DIS=PAR+AA
      CALL UXVDIR(N,-DIS,XO,S,XO)
      DIS=ONE/(B*DIS)
      CALL UXDSMU(N,H,DIS,XO)
      CALL UXDSMU(N,H,-DIS,S)
      TUXX='BFGS    '
      GO TO 21
   20 CONTINUE
*
```

```
*     GENERAL UPDATE
*
      DEN=PAR+POM*AA
      DIS=POM/DEN
      CALL UXDSMU(N,H,(PAR*DIS-ONE)/A,S)
      CALL UXVDIR(N,-DIS,S,XO,S)
      CALL UXDSMU(N,H,DEN/B,S)
   21 CONTINUE
      ITERH=0
      IF (GAM.EQ.ONE) GO TO 22
*
*     SCALING
*
      CALL UXDSMS(N,H,GAM)
   22 CONTINUE
      CALL UOU1D2(N,H,S,RHO,GAM,PAR,A,B,C,POM,ETA9)
      RETURN
      END
```

The results (screen output) obtained by using this control program have the following form:

```
   0 NIT=   40 NFV=  138 NFG=    0  GRAD TOL  F=  .5038712822E-13 G= .828D-05
 FF =   .5038712822D-13
 X  =   .1000000098D+01   .1000000177D+01
```

Appendix B. The BEL interpreter

The BEL (Batch Editor Language) interpreter, developed as a part of the UFO project, is especially determined for the generation of computer programs, batch editing of texts, preparation of print files, filtering of text files etc. The BEL interpreter allows us to generate a prescribed output file from the input file (template) which is a mixture of text lines and special instructions.

The UFO system is organized in such a way that a control program does not have to be written in the FORTRAN language immediately. Instead, the procedure written in the UFO control language is supplied. By using the installation template, the compiler of the UFO control language (UFOCLP - UFO Control Language Preprocessor) generates a table of symbols which, together with the user supplied procedure, is offered to the BEL interpreter. The BEL interpreter then generates the resulting control program which is written in the FORTRAN language.

B.1. General description

Although the BEL interpreter can be used in various general applications, it was developed especially for the generation of FORTRAN programs. It is:

1. Interpreter, since instructions contained in the input text are interpreted and immediately realized.
2. Batch editor, since it serves for editing batch files.
3. Macroprocessor, since it makes it possible to define or modify special macrovariables which can be substituted into the processed text.

The macrovariable can be an integer constant, a logical constant, a string of characters, a set of text lines, a set of BEL instructions, even a text file.

The BEL interpreter requires an input text file and a table of symbols. The input text file (template) consists of standard text lines together with the BEL instructions. The table of symbols contains names and values of the macrovariables used.

The BEL instructions, contained in the input text file, can be of two types:

1. Directives, i.e. control instructions and instructions for manipulation with the table of symbols. These instructions begin with the special character CHDIR. In the subsequent text, we will suppose that CHDIR='$' ('$' is the default value).
2. Substitutions, i.e.instructions for substituting macrovariables into the text. These instructions begin with the special character CHSUB. In the subsequent text, we will suppose that CHSUB='$' ('$' is the default value).

The BEL interpreter works in the following way:

1. The line of the input file is read.
2. The line is recognized and if the character CHSUB is found, a pertinent substitution is realized.
3. If the first character (different from blank) is CHDIR, the line is a directive line. The recognized directive is realized.

This process is repeated until the directive $END or the end of the file is found. Note that we suppose that CHSUB and CHDIR have the same values. This is allowed, since the correct meaning is recognized from the context.

At the end of this subsection, we stress some specific features and advantages of the BEL interpreter.

1. The substitution is recursive. The depth of recursion only depends on the declared work space size.

2. Substitution is allowed in both the text lines and the directives.

3. The names and values of macrovariables can have an arbitrary length which again only depends on the declared work space size.

4. The set of directives is relatively small with a consistent syntax. It contains all important instructions ($IF-$ELSEIF-$ELSE-$ENDIF, $DO-$ENDDO, $REPEAT-$UNTIL etc.)

5. The control parameters (CHDIR, CHSUB etc.) can be changed during the work of the BEL interpreter. This makes it possible to generate a program written in the BEL language which can be immediately processed.

6. The BEL interpreter is a fully portable device. It can be implemented in an arbitrary system containing FORTRAN 77 compiler.

B.2. List of instructions

Substitutions:

$INTEGER	- Substitute by the absolute label computed from the relative label.
$NAME, $(NAME)	- Substitute by the value of the macrovariable NAME.
$DATA(NAME)	- Substitute by a new item from the list of items which is a value of the macrovariable NAME.
$DEF(NAME)	- Substitute by '.TRUE.' if the macrovariable NAME is defined in the table of symbols. Otherwise substitute by '.FALSE.'
$INT(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is an integer constant. Otherwise substitute by '.FALSE.'
$LOG(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is a logical constant. Otherwise substitute by '.FALSE.'
$REAL(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is a real constant. Otherwise substitute by '.FALSE.'
$$	- Substitute '$' (replace '$$' by '$'). This makes possible to insert the character CHSUB into the text.

Directives:

$ADD	- Add a value to a macrovariable.
$ADD, $ENDADD	- Add text lines to a macrovariable.
$CLEAR	- Clear value of a macrovariable which is a list of items type.
$DO, $ENDDO	- Cycle.
$ERASE	- Erase a macrovariable from the table of symbols.
$EXIT	- Termination of the BEL interpreter work.
$HELP, $CHECK	- Set a default value to a macrovariable which has not been previously defined.
$IF, $ELSEIF, $ELSE, $ENDIF	- Conditioned instruction.
$INCLUDE	- Insert a macrovariable or a text file into the output file.
$OPTION	- Change some optional parameter of the BEL interpreter.
$REM	- Remark.
$REPEAT, $UNTIL	- Cycle.
$RESTORE	- Adjust the list of items pointer to the first item.
$REWIND	- Rewind the file on a given unit.
$SET	- Set a value to a macrovariable.
$SET, $ENDSET	- Set text lines to a macrovariable.
$STOP	- Termination of the BEL interpreter work.
$SUBST	- Substitute a text file into the input file.

B.3. Special characters

The following special characters are important for the BEL interpreter work:

$ - CHSUB (Substitution Character) - this is the first character in every substitution. If '$' should be inserted into the text, we have to use '$$'.

$ - CHDIR (Directive Character) - if the first character on the line is CHDIR, then the line is a directive line (CHSUB and CHDIR are distinguished by the context).

& - CHCON (Continuation Character) - if the last character on the line is CHCON, then it is assumed that the logical line continues on the next physical line.

; - CHEOL (End Of Line Character) - this character specifies the end of the logical line if it does not coincide with the end of the physical line. This makes it possible to write several logical lines by using the same physical line.

\ - CHDS (Data Separator Character) - this character separates individual items in the list of items type macrovariable.

The use of special characters can be demonstrated by the following simple example. Assume that the input text has the form

```
$A='Paul\Peter\Jane\Mary'
This is a list of my brothers and sis&
ters:
$DO(I=1,4); $DATA(A); $ENDDO
```

Then the output from the BEL interpreter has the form

```
This is a list of my brothers and sisters:
Paul
Peter
Jane
Mary
```

The special characters can be changed by the directive $OPTION. But no special character has to be the alphabet or the digit. Moreover, different special characters have to differ (with the exception of CHSUB and CHDIR).

B.4. Description of instructions

This subsection contains a detailed description of the syntax and action of individual BEL instructions. The following definitions will be used:

<digit> ::= 0 | 1 | 2 | 3 | ..... | 9

<alphabet> ::= A | B | C | D | ..... | Z

<character> ::= an arbitrary character with the exception of apostrophe

<integer constant> ::= (+ | -) <digit> {<digit>}

<logical constant> ::= .TRUE. | .FALSE.

<macroname> ::= <alphabet> {<alphabet> | <digit>}

<string of characters> ::= '{<character> | ''}

<text> ::= <string of characters> '{; <string of characters>}

<list of items> ::= <string of characters> '{\ <string of characters>}

Substitutions:

$INTEGER

Syntax:
The type of INTEGER is an integer constant. Although it can have an arbitrary value, an application to the control program generation requires it to be positive and lower than LABEL2 (see the directive $OPTION).

Action:
The integer constant INTEGER is a relative label in a given template. The absolute label, substituted into the control program, is computed by the formula LABEL=LABEL1+K*LABEL2, where LABEL1 and LABEL2 are options of the BEL interpreter (see the directive $OPTION) and K is a serial number of the application of the directive $SUBST.

Example:

$10

generates

10010

if the main template is used or

10110

after the first application of the directive $SUBST.

$NAME, $(NAME)

Syntax:
The type of NAME is a macroname. This substitution has two forms, either $NAME or $(NAME). The latter form is required if the substitution appears inside a continuous string of characters to separate the NAME from the adjacent text.

Action:
The string '$NAME' is replaced by the value of the macrovariable NAME.

Example:

$A='UFO'
$A SYSTEM

generates

UFO SYSTEM

$DATA(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The string '$DATA(NAME)' is replaced by the next item of the list of items which is a value of the macrovariable NAME. If the next item does not exist, the list of items pointer is returned to the first item. Additional information is contained in the description of the directive $RESTORE.

Example:

$LIST='ITEM1\ITEM2\ITEM3'
$DATA(LIST)
$DATA(LIST)

$DATA(LIST)
$DATA(LIST)

generates

ITEM1
ITEM2
ITEM3
ITEM1

$DEF(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the macrovariable NAME is defined in the table of symbols, the string '$DEF(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=10
$DEF(A)

generates

.TRUE.

$INT(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is an integer constant, the string '$INT(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=-25
$INT(A)

generates

.TRUE.

$LOG(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is a logical constant, the string '$LOG(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=.FALSE.
$LOG(A)

generates

.TRUE.

$REAL(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is a real constant (i.e. a string of characters which satisfies the syntactic rules for FORTRAN real constants), the string '$REAL(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

```
$A='-0.09D-12'
$REAL(A)
```

generates

.TRUE.

$$

Action:
The string '$$' is replaced by the character '$'. This substitution allows us to insert the character '$' into the generated text or into the macrovariable.

Example:

```
$I='NAME'
$$DEF($I)
```

generates

$DEF(NAME)

Directives:

$ADD(NAME1,NAME2 or VALUE)

Syntax:
The type of NAME1 and NAME2 is a macroname.
The type of VALUE is an integer constant or a logical constant or a string of characters.

Action:
The value of the macrovariable NAME2 or the VALUE is added to the value of the macrovariable NAME1 (the resulting value of the macrovariable NAME1 is $NAME1$NAME2 in the first case).

Example:

```
$NAME='TOM'
$ADD(NAME,' JONES')
Name: $NAME
```

generates

Name: TOM JONES

$ADD(NAME)
TEXT
$ENDADD

Syntax:
The type of NAME is a macroname.
The type of TEXT is a text.

Action:
The TEXT is added to the value of the macrovariable NAME.

Example:

```
$SET(A)
   Day: 31
$ENDSET
$ADD(A)
   Month: December
   Year: 1998
$ENDADD
```

generates

```
   Day: 31
   Month: December
   Year: 1998
```

Remark: Only substitutions are realized in the text TEXT (not directives).

$CLEAR(NAME)

Syntax:
The type of NAME is a macroname.

Action:
This directive clears a list-of-items-type value of the macrovariable NAME, i.e. it deletes all duplications of items. Small and capital letters of items are not distinguished.

Example:

```
$DECL='N\IX(N)\N\M\ I\J\N\M'
$CLEAR(DECL)
$END='$DATA(DECL)'
$REPEAT
   $I='$DATA(DECL)'
   INTEGER $I
$UNTIL(I=END)
```

generates

```
   INTEGER IX(N)
   INTEGER M
   INTEGER I
   INTEGER J
   INTEGER N
```

$DO(NAME=INDEX1,INDEX2,INDEX3)
TEXT
$ENDDO

Syntax:
The type of NAME is a macroname.
The type of INDEX1, INDEX2, INDEX3 is a macroname or an integer constant.
The type of TEXT is a text.

Action:
This directive has a similar meaning as the statement DO in the FORTRAN language:
NAME is the cycle counter.

INDEX1 is the initial value of the cycle counter.
INDEX2 is the final value of the cycle counter.
INDEX3 is the change of the cycle counter after a cycle step.
If INDEX3 is not present, the default value INDEX1=1 is assumed.
The cycle counter NAME does not have to be changed in the cycle step.
The value INDEX3 does not have to be equal to 0.
The body of the cycle is terminated by $ENDDO.
If INDEX1>INDEX2 and INDEX3>0 or INDEX1<INDEX2 and INDEX3<0, then the cycle is not realized.
Cycles can be nested. The maximum depth of nested cycles is 20.

Example:

```
$A='X\Y\Z'
$DO(I=1,5,2)
   A($I,1)=C($I)+$DATA(A)
$ENDDO
```

generates

```
   A(1,1)=C(1)+X
   A(3,1)=C(3)+Y
   A(5,1)=C(5)+Z
```

$ERASE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The macrovariable NAME is erased from the table of symbols.

Example:

```
$A=1
$DEF(A)
$ERASE(A)
$DEF(A)
```

generates

```
   .TRUE.
   .FALSE.
```

$EXIT

Action:
The directive $EXIT has the same meaning as the end of the file achievement. If the nested files are processed (see the description of the directive $SUBST), the directive $EXIT realizes return to the higher level file (if the higher level file does not exist, then $EXIT has the same meaning as $STOP).

$HELP
TEXT
$CHECK(NAME,DEFAULT,TYPE,LEVEL,TRANSFER)

Syntax:
The type of TEXT is a text.
The type of NAME is a macroname.
The type of DEFAULT is either a macroname or an integer constant or a logical constant or a string of characters.

The type of TYPE is either a list of items or one of the strings INT (integer), LOG (logical), REAL (real).
The type of LEVEL is an integer constant.
The type of TRANSFER is a logical constant.

Action:
The text TEXT appears on the screen if the dialogue mode is used. The value of the macrovariable $NAME is checked to have the type TYPE. If the macrovariable $NAME is not defined or if it has a wrong value, the value DEFAULT is used. The value of LEVEL gives the lowest level of the dialogue (1,2,3 or 4) from which the text TEXT appears on the screen. The value of TRANSFER specifies transfer of the variable $NAME into the control program (YES if transfer is accepted or NO if transfer is suppressed).

Example:

```
$HELP
   TYPE OF THE HESSIAN MATRIX:
        D - DENSE
        B - SPARSE WITH KNOWN (PARTITIONED) STRUCTURE
        S - SPARSE WITH KNOWN (GENERAL) STRUCTURE
        N - HESSIAN MATRIX IN NOT USED
$CHECK(HESF,'N','D\B\S\N',1,NO)
```

$IF(CONDITION) LINE

Syntax:
The CONDITION can be of the following types:
The type of CONDITION is a macroname and a value of CONDITION is a logical constant.
The type of CONDITION is a logical constant (.TRUE. or .FALSE.).
The type of CONDITION is a string of the form PART1<operator>PART2.
The type of PART1 and PART2 can be a macroname or an integer constant or a logical constant or a string (values of PART1 and PART2 have to be of the same type) and <operator> can have the following forms:

= equal to
<> not equal to
< less than (for integer values only)
<= less than or equal to (for integer values only)
> greater than (for integer values only)
>= greater than or equal to (for integer values only)

LINE is either a text line or a directive.

Action:
If the condition CONDITION is satisfied, LINE is inserted into the output file (if it is a text line) or carried out (if it is a directive). If the values of PART1 and PART2 are strings, then small and capital letters are not distinguished and blanks are ignored.

Example:

```
$A='J O H N'
$IF(A='John') Yes
$IF(A<>'Mary') No
```

generates

```
   Yes
   No
```

$IF(CONDITION1)
  TEXT1
$ELSEIF(CONDITION2)
  TEXT3
  .
  .
  .
$ELSE
  TEXT
$ENDIF

Syntax:
CONDITION1 and CONDITION2 have the same syntax and meaning as CONDITION in the previous case. The number of repeated $ELSEIF is not limited, $ELSEIF or $ELSE can be omitted.

Action:
This directive has a similar meaning as the conditioned statement IF-ELSEIF-ELSE-ENDIF in the FORTRAN language. The conditioned statements can be nested. The maximum depth of nested conditioned statements is 20.

Example:

```
$A=10
$L=.FALSE.
$IF(A=10)
   A = A + 1
   B = B + 1
   $IF(L)
   C = C + 1
   $ENDIF
$ELSE
   WRITE(*,*) I
$ENDIF
```

generates

```
   A = A + 1
   B = B + 1
```

$INCLUDE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The directive $INCLUDE(NAME) is a special case of substitution. This directive makes it possible to insert (into the generated text) one or more lines, which were previously assigned to the macrovariable NAME. In contrast to the standard substitution $NAME, the inserted lines are not processed by the BEL interpreter, so the directives are not carried out.

Example:

```
$SET(LINES)
   $ADD(A)
   X = Y + Z
   CALL SUB(X)
   $ENDADD
$ENDSET
```

$INCLUDE(LINES)

generates

```
$ADD(A)
X = Y + Z
CALL SUB(X)
$ENDADD
```

$INCLUDE('FILE')

Syntax:
The type of FILE is a string.

Action:
The directive $INCLUDE('FILE') is a special case of substitution. This directive makes it possible to insert (into the generated text) the text which is stored in the file with the name FILE. The inserted text is not processed by the BEL interpreter, so the directives are not carried out.

Example:

$INCLUDE('C:\UFO\UMCOMN.I')

includes FORTRAN common blocks into the generated text (these common blocks are stored in the file `C:\UFO\UMCOMN.I`.

$OPTION(OPTIONNAME=NAME or VALUE)

Syntax:
OPTIONNAME is a selected name from the table of optional parameters (see below).
The type of NAME is a macroname. The value of NAME has to be an integer constant or a logical constant or a string of character and has to correspond to the type of OPTIONNAME.
The type of VALUE has to be an integer constant or a logical constant or a string of character and has to correspond to the type of OPTIONNAME.

Action:
This directive makes us possible to change selected optional parameter of the BEL interpreter. Optional parameters are contained in the following table.

Name	Type	Default	Description
CHDIR	char.	'$'	see B.3
CHEOL	char.	';'	see B.3
CHCON	char.	'&'	see B.3
CHDS	char.	'\'	see B.3
FN1	char.	' '	first part of the file name
FN2	char.	'.I '	last part of the file name
ILNLEN	int.	80	physical length of the input line
OLNLEN	int.	80	physical length of the output line
IUNIT	int.	-	No. of the input file unit
OUNIT	int.	-	No. of the output file unit
INUNIT	int.	-	No. of the $INCLUDE files unit
IIUNIT	int.	-	No. of the interactive mode input unit
OIUNIT	int.	-	No. of the interactive mode output unit
DIALOG	int.	1	level of dialogue (0 or 1 or 2)
MODERW	int.	1	READ/WRITE mode (1 or 2 or 3)
LABEL1	int.	10000	initial label
LABEL2	int.	100	difference between two consecutive labels
LSUBS	log.	.TRUE.	substitutions carried out
LOUT	log.	.TRUE.	output file created
LSMLET	log.	.TRUE.	small letters used in instructions
LFORTO	log.	.TRUE.	output in standard FORTRAN format
LFRFMT	log.	.TRUE.	input in free FORTRAN format (used only if LFORTO=.TRUE.)
SIFDEC	log.	.FALSE.	using the SIF decoder
DIALGR	log.	.FALSE.	using the graphic dialogue

$REM

Action:
The rest of the line (following after $REM) is ignored by the BEL interpreter. The directive $REM is used for remarks.

$REPEAT
TEXT
$UNTIL(CONDITION)

Syntax:
The type of TEXT is text.
CONDITION has the same syntax and meaning as that in the directive $IF(...).

Action:
This directive has a similar meaning as the statement REPEAT-UNTIL in the PASCAL language:
The cycle is terminated whenever the condition CONDITION is satisfied (at least one realization is carried out).
Cycles can be nested. The maximum depth of nested cycles is 20.

Example:

```
$N=20
$REAL='X($N)\G($N)\H($N,$N)\.END.'
$REPEAT
   $I=$DATA(REAL)
   REAL $I
$UNTIL(I='.END.')
```

generates

```
REAL X(20)
REAL G(20)
REAL H(20,20)
```

$RESTORE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The directive $RESTORE(NAME) can only be used if the value of the macrovariable NAME is a list of items. Such a macrovariable uses a pointer which points out the next called item. The directive $RESTORE adjust this pointer to point out the first item of the list (if the end of this list is found, the pointer is adjusted to point out the first item without applying the directive $RESTORE).

Example:

```
$A='X\Y\Z'
   $DATA(A)
   $DATA(A)
$RESTORE(A)
   $DATA(A)
```

generates

```
X
Y
X
```

$REWIND(UNIT)

Syntax:
The type of UNIT is an integer constant.

Action:
The file opened on the unit with the number UNIT is rewound, so it can again be read from the first record (numbering of I/O units is used in the FORTRAN language).

$NAME1 = NAME2 or VALUE

$SET(NAME1 = NAME2 or VALUE)

Syntax:
The type of NAME1 and NAME2 is a macroname.
The type of VALUE is an integer constant or a logical constant or a string of characters.
This directive has two forms. The latter form is used if the macroname is identical with a directive (e.g. $SET(REM='REMARK')).

Action:
The new macrovariable with the name NAME1 and the value equal to the value of the macrovariable NAME2 or constant VALUE is inserted into the table of symbols. If the macrovariable NAME1 has already been defined in the table of symbols, then it is changed.

$SET(NAME)
TEXT
$ENDSET

Syntax:
The type of NAME is a macroname.

The type of TEXT is text.

Action:
The macrovariable NAME is inserted into the table of symbols with the value TEXT. If the macrovariable NAME has already been defined in the table of symbols, then it is changed.

Example:

```
$SET(INIT)
   CALL EIUD01(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
   IF (IERR.NE.0) GO TO $$ENDTEST
$ENDSET
$INIT
```

generates

```
   CALL EIUD01(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
   IF (IERR.NE.0) GO TO $ENDTEST
```

Remark: Only substitutions are realized in the text TEXT (not directives).

$STOP

Action:
The directive $STOP terminates the BEL interpreter work.

$SUBST('FILE')

Syntax:
The type of FILE is a string.

Action:
This directive performs the following actions:
The new reference label is computed (using the parameters LABEL1 and LABEL2 of the BEL interpreter).
The file with the name FILE is opened.
This file is processed by the BEL interpreter.
The file with the name FILE is closed.
The old reference label is restored.

This directive is similar to the directive $INCLUDE('FILE'). But the inserted text is now processed by the BEL interpreter. All substitutions and directives are carried out. The directive $SUBST('FILE') serves for dividing large texts into segments and makes it possible to generate texts by using conditioned branching. This is advantageously used for generation of the control program in the UFO system where templates corresponding to individual subroutines are such segments.

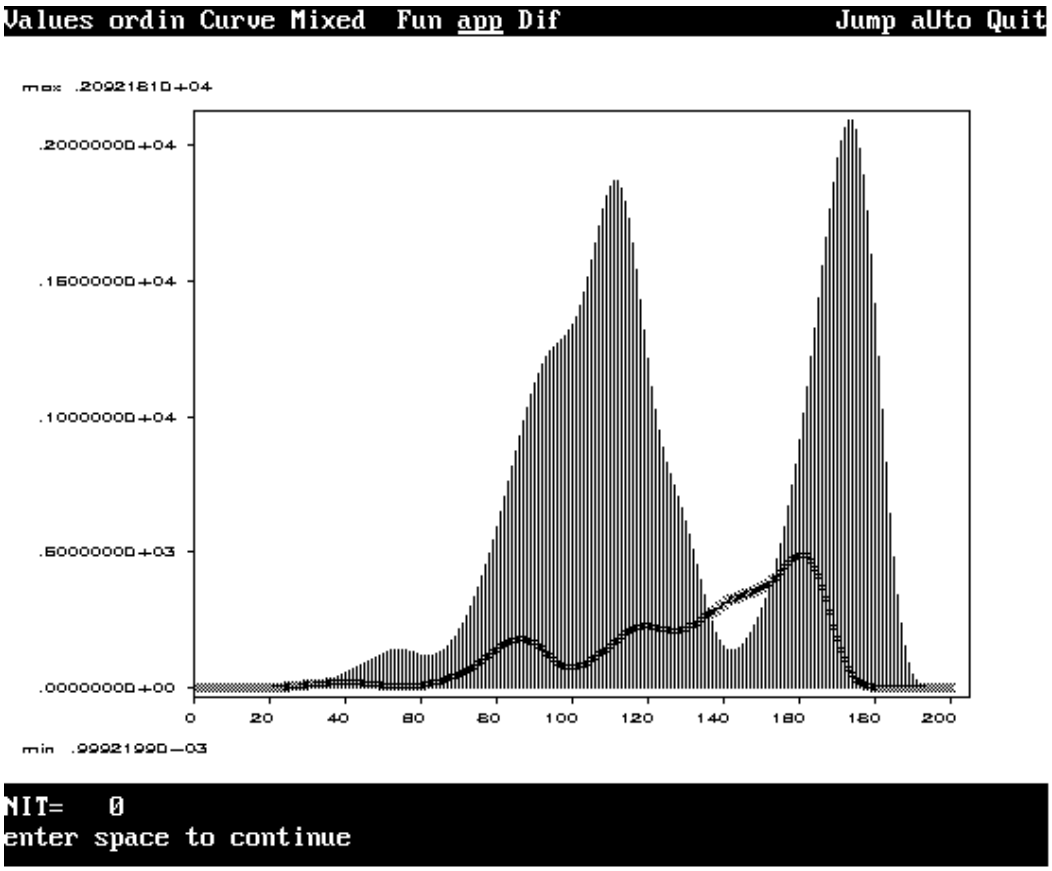
Example:

```
$SUBST('C:\UFO\PROBLEM.UFO')
```

inserts a template, written in the UFO control language, into the generated text (this template is stored in the file `C:\UFO\PROBLEM.UFO`.

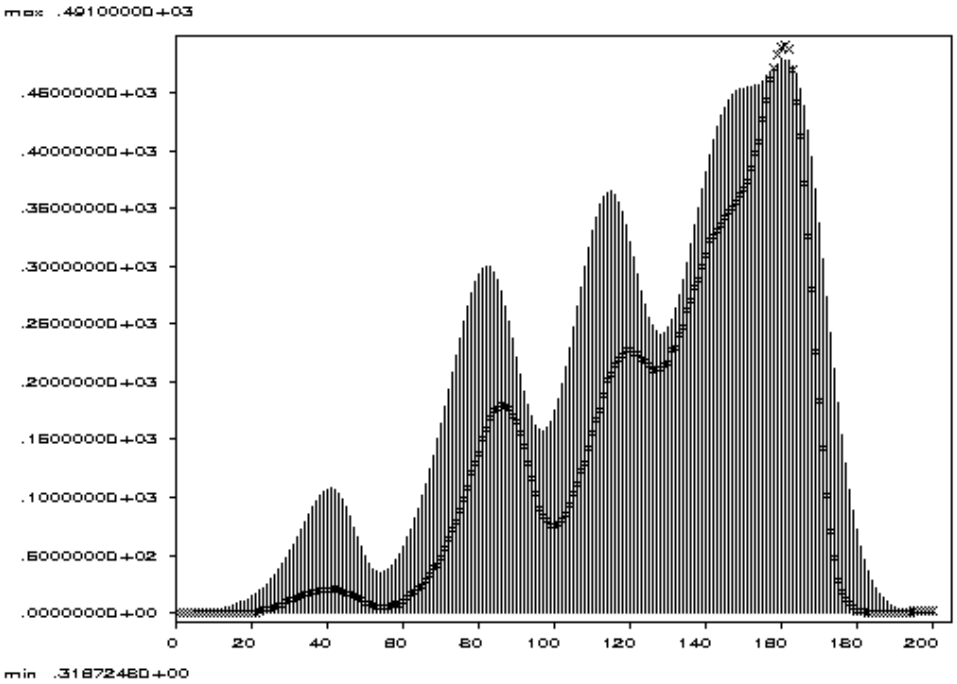
Appendix C. Graphic screen output

C.1. Nonlinear regression



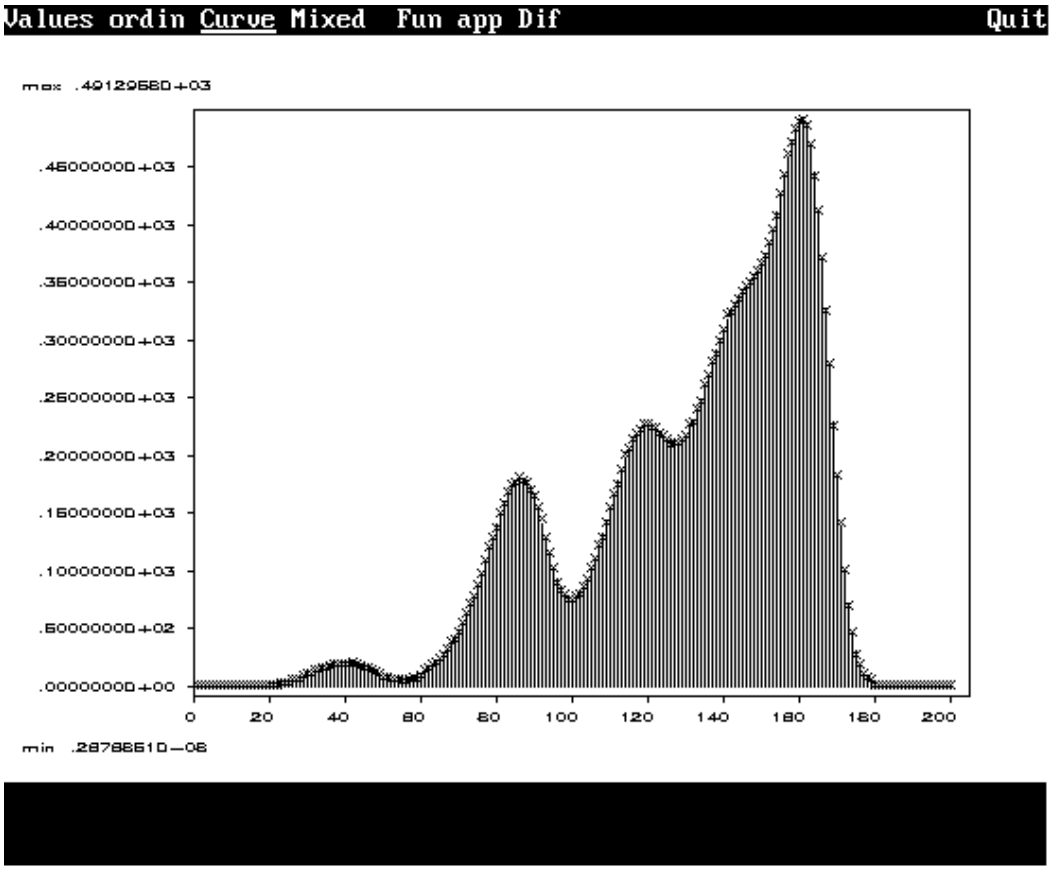
Nonlinear regression: Iteration 0

```
Values ordin Curve Mixed  Fun app Dif                    Jump aUto Quit
```

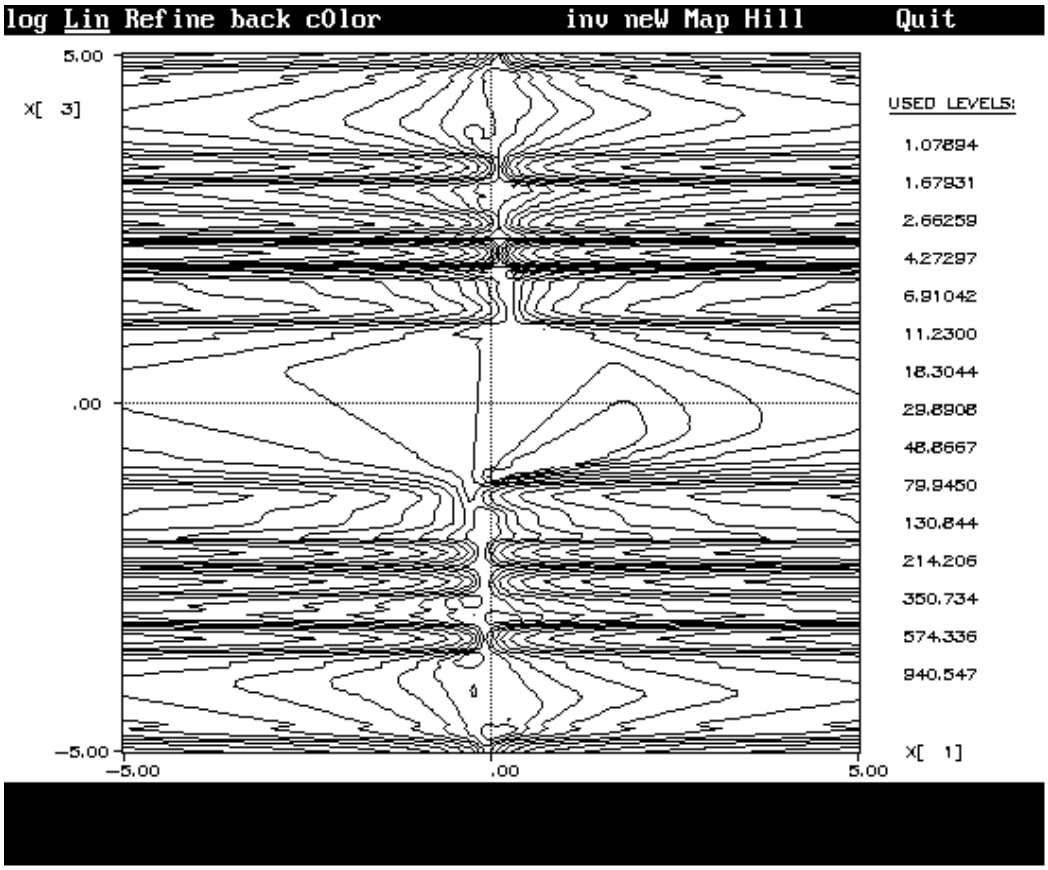



```
NIT=   6
enter space to continue
```

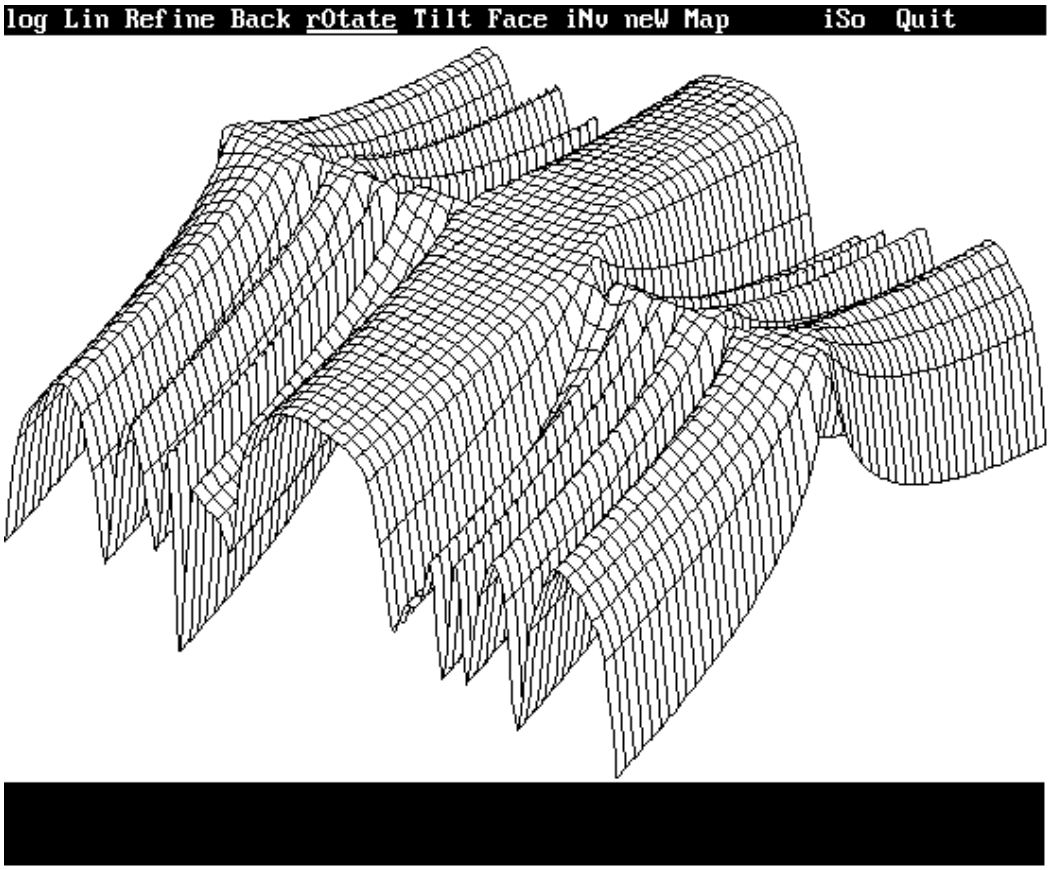
Nonlinear regression: Iteration 6



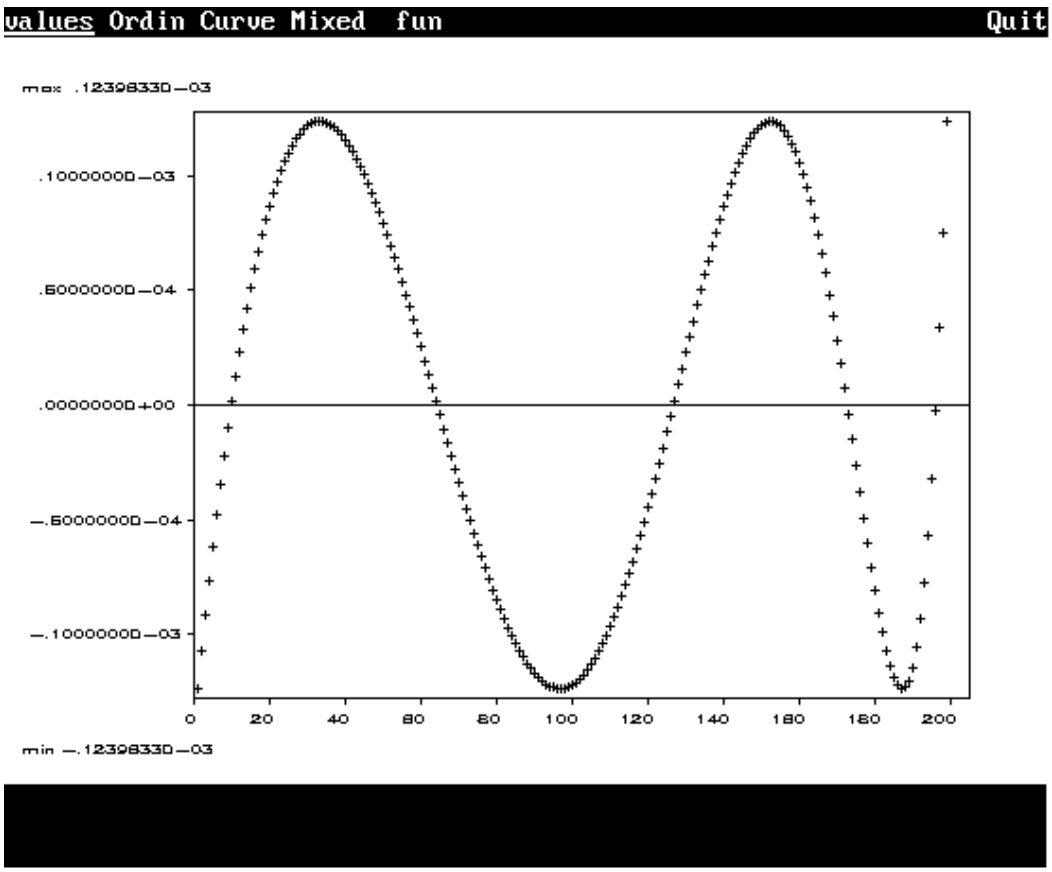
Nonlinear regression: Final solution



Nonlinear minimax optimization: Isolines

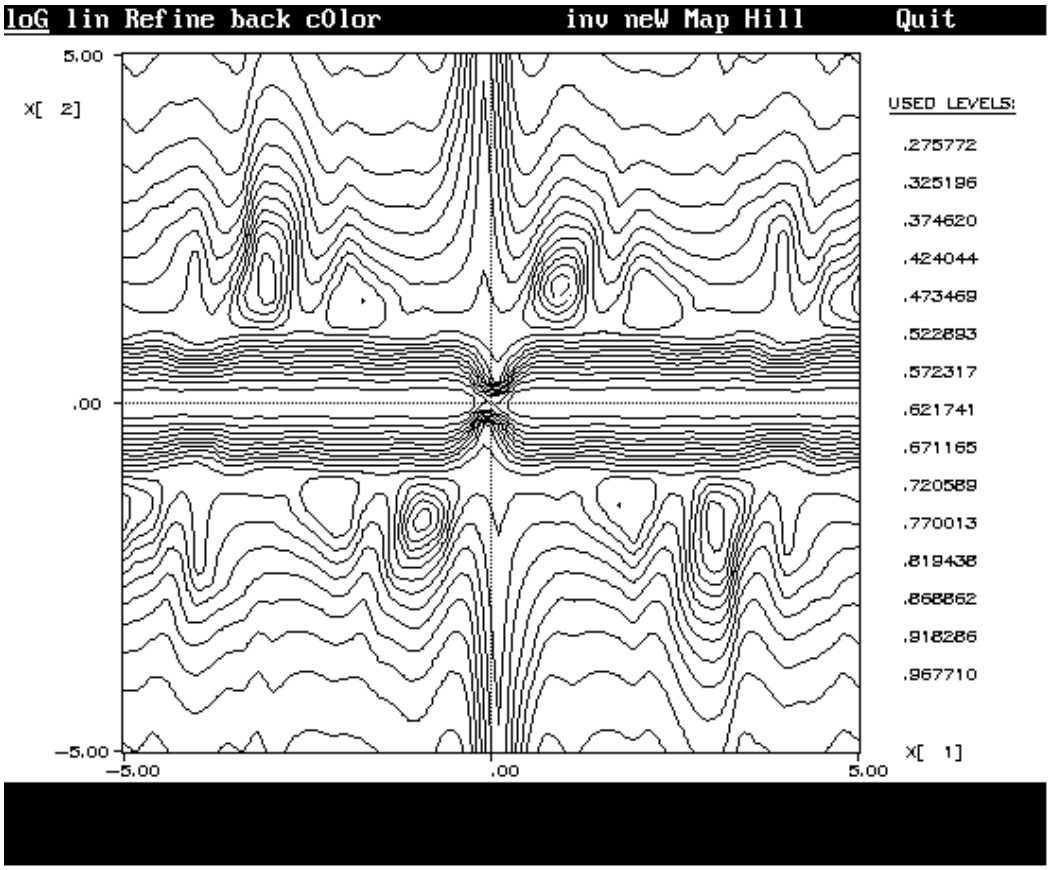


Nonlinear minimax optimization: Surface

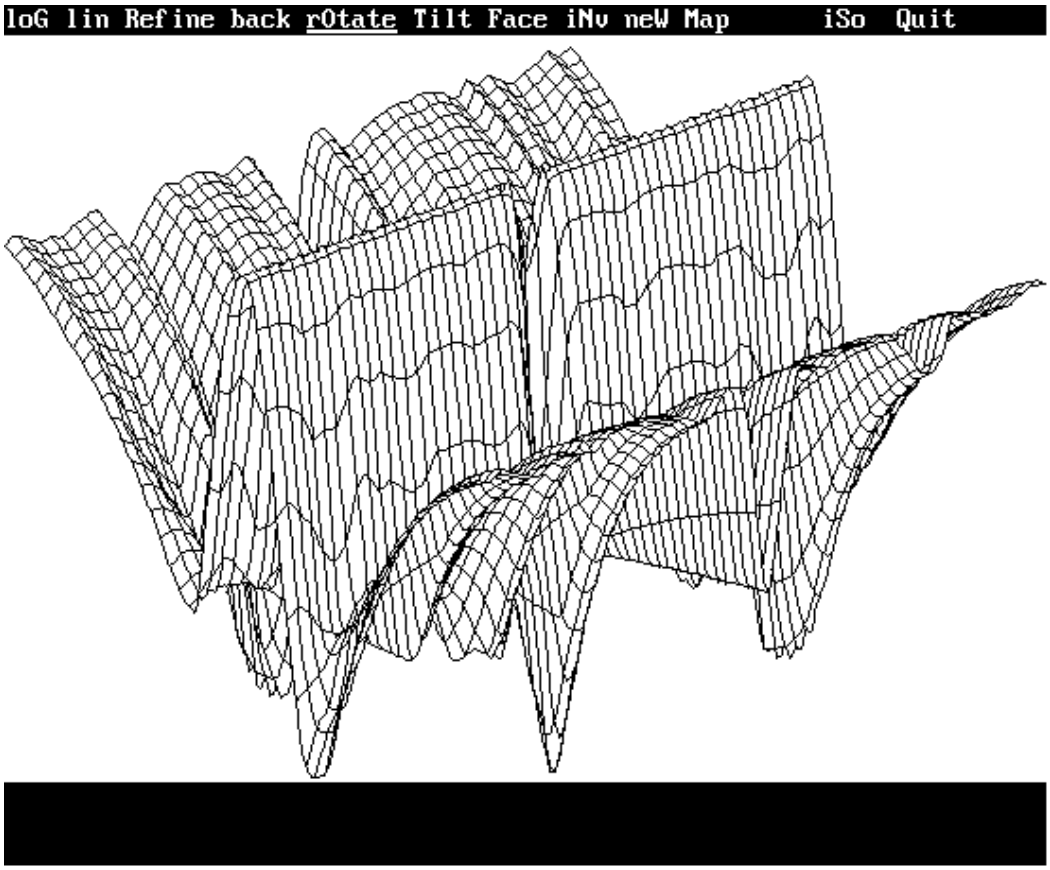


Nonlinear minimax optimization: Graph

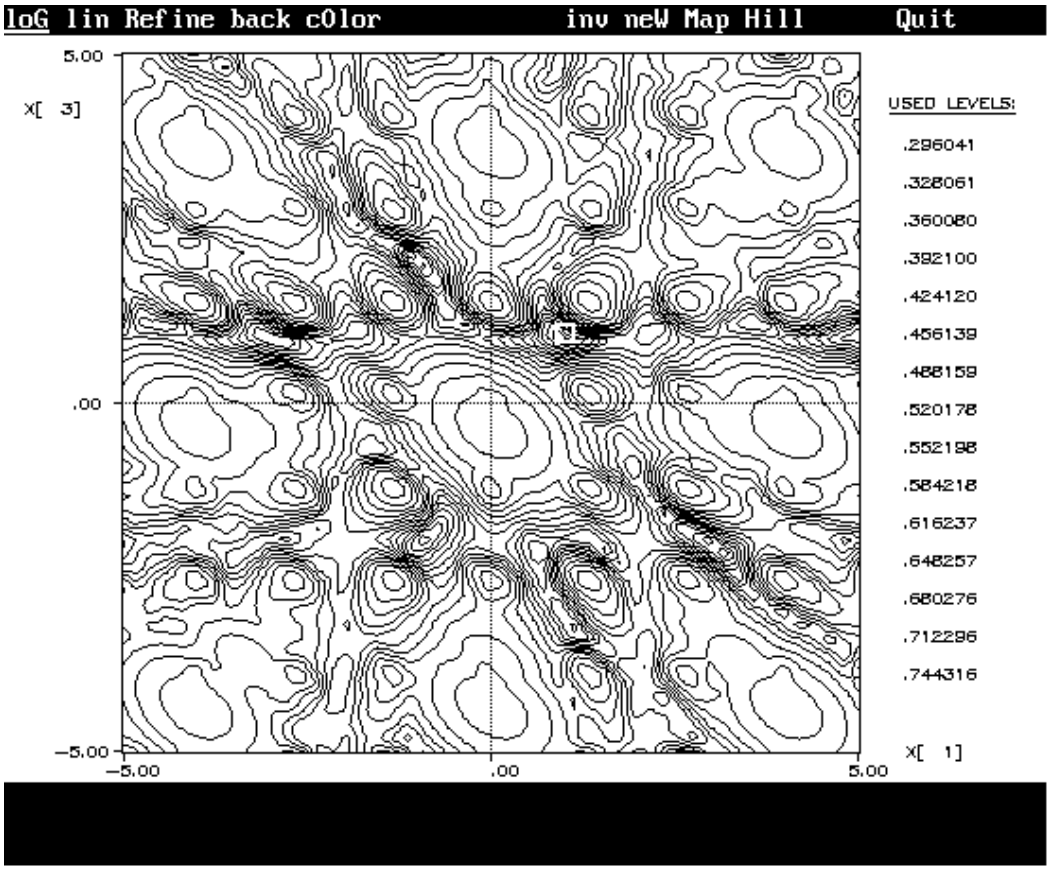
C.3. Transformer network design



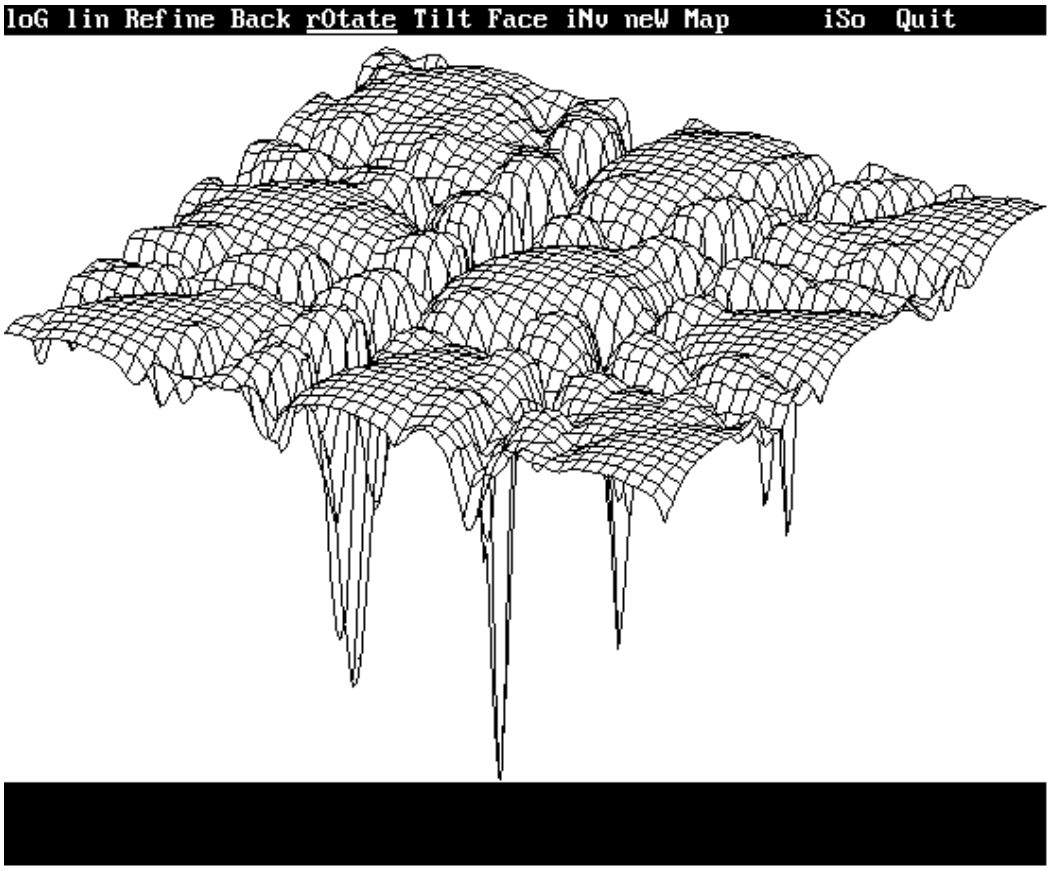
Transformer network design: Isolines



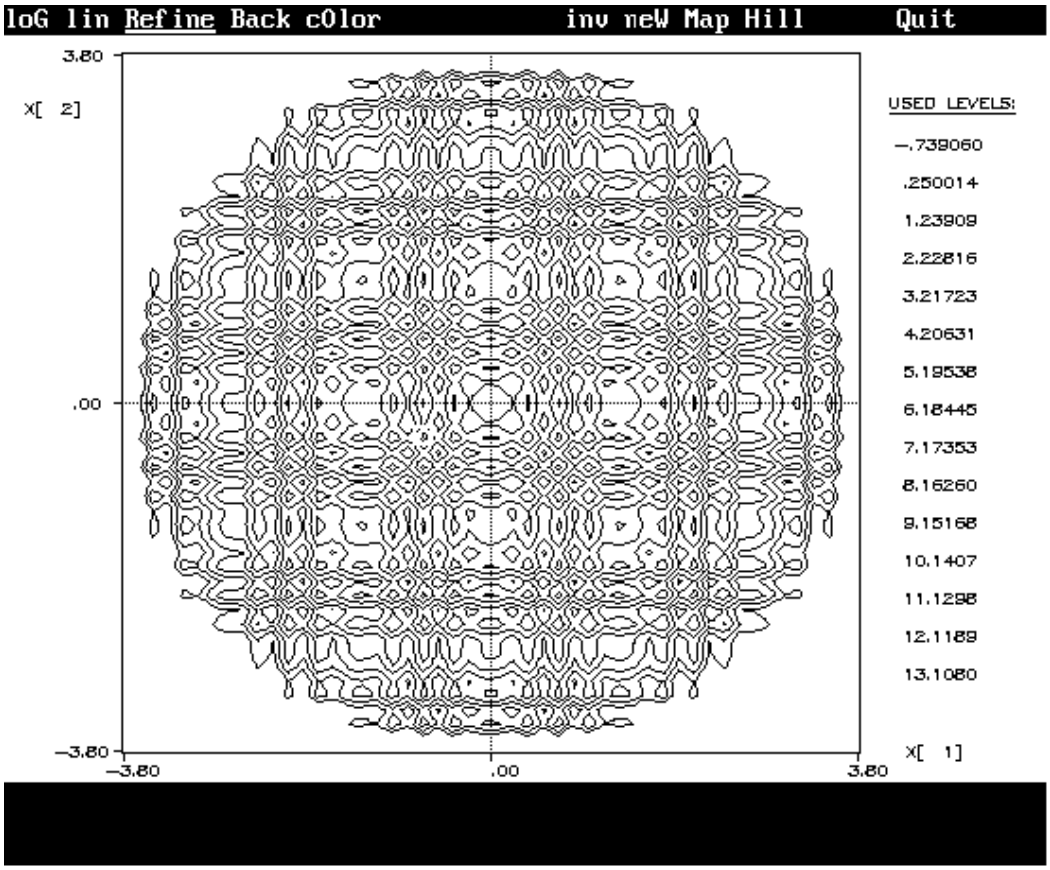
Transformer network desogn : Surface



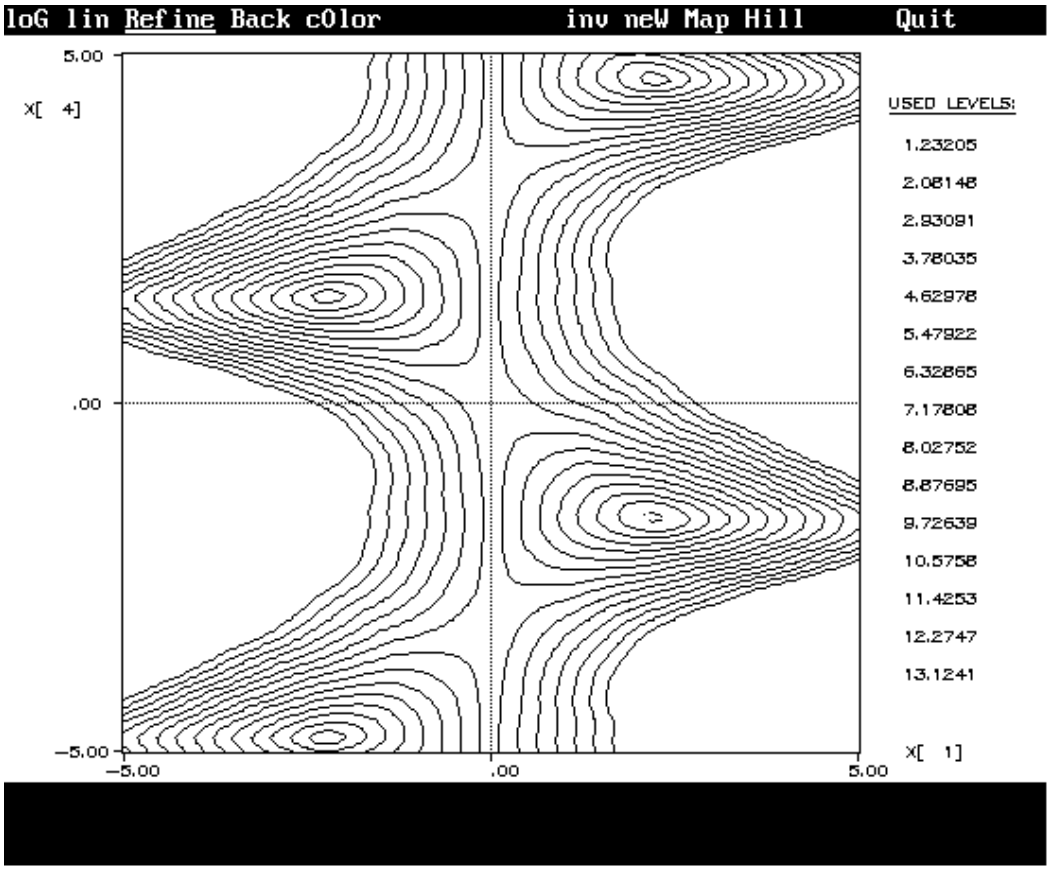
Transformer network design: Isolines



Transformer network design: Surface

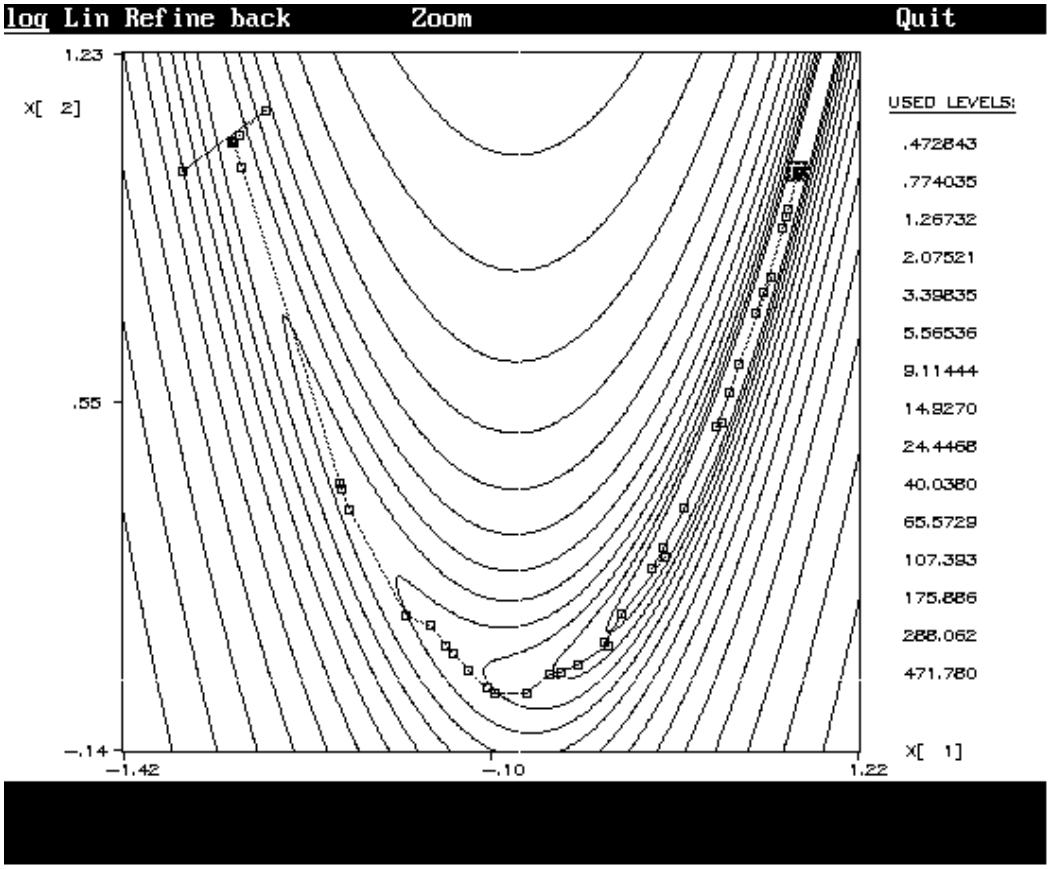


Global optimization: Isolines

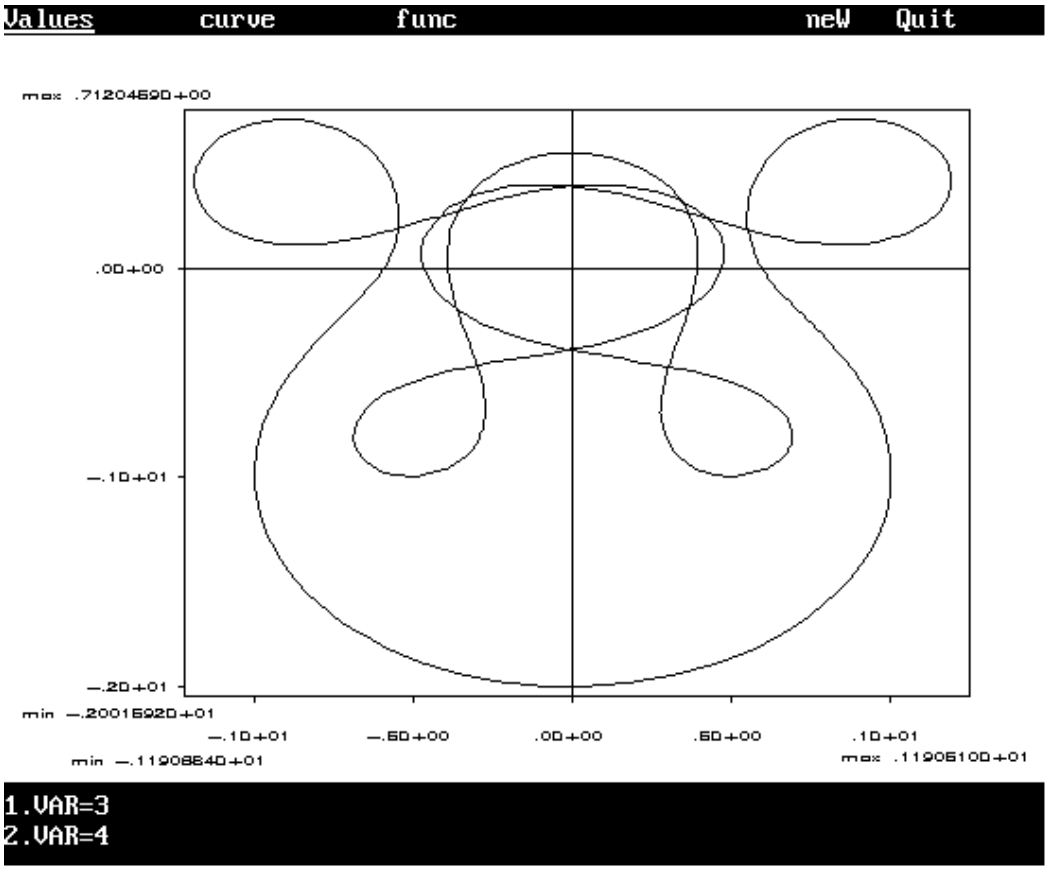


Nonsmooth optimization: Isolines

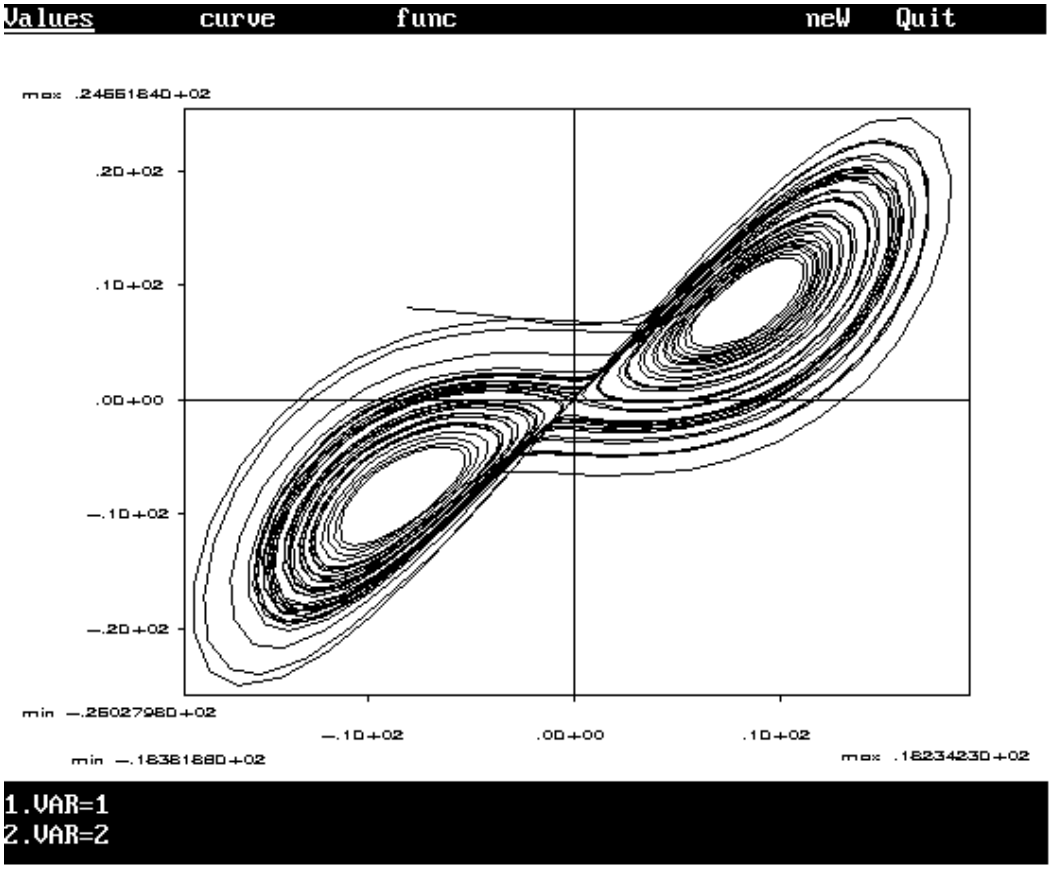
C.6. Rosenbrock function



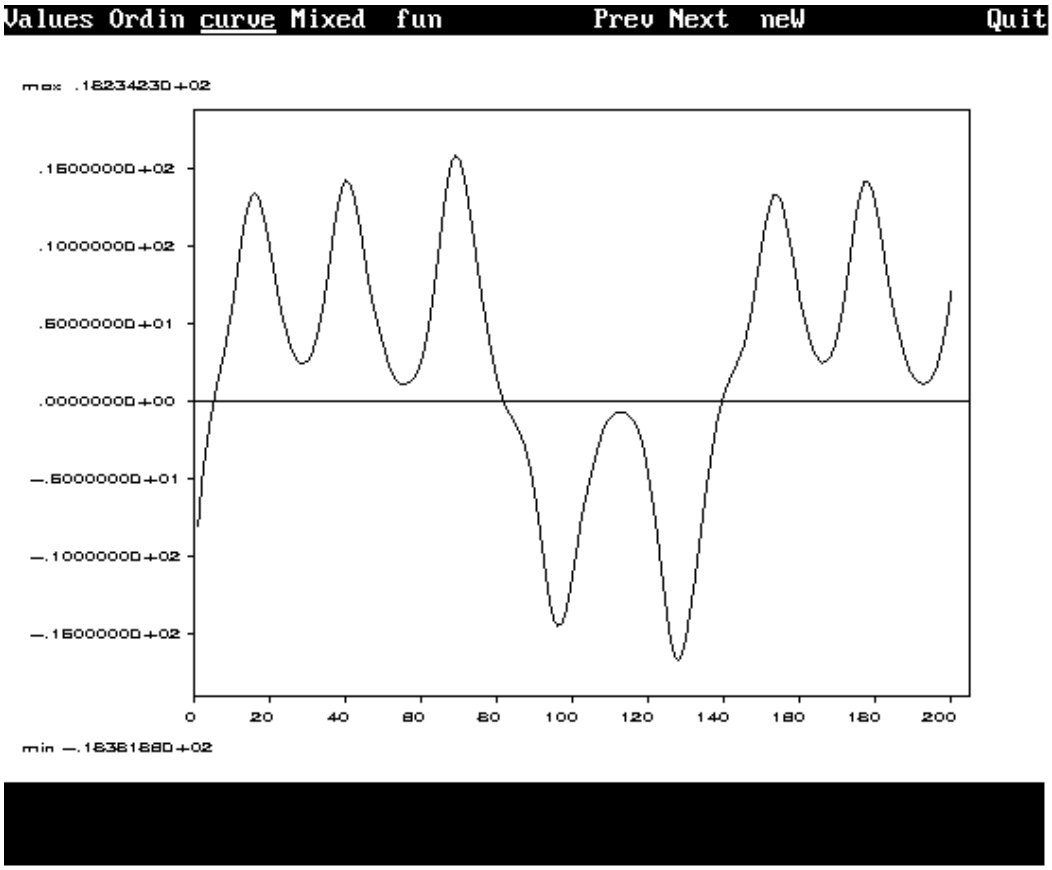
Rosenbrock function: Path of iterations



Ordinary differential equations: Orbit



The Lorenz attractor: Orbit



The Lorenz attractor: Trajectory