



národní
úložiště
šedé
literatury

Interactive System for Universal Functional Optimization (UFO) - Version 1995

Lukšan, Ladislav
1995

Dostupný z <http://www.nusl.cz/ntk/nusl-33642>

Dílo je chráněno podle autorského zákona č. 121/2000 Sb.

Tento dokument byl stažen z Národního úložiště šedé literatury (NUŠL).

Datum stažení: 04.06.2024

Další dokumenty můžete najít prostřednictvím vyhledávacího rozhraní nusl.cz .

INSTITUTE OF COMPUTER SCIENCE

ACADEMY OF SCIENCES OF THE CZECH REPUBLIC

Prague

**Interactive System for Universal Functional
Optimization (UFO) - Version 1995**

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

Technical Report No. V-662

December 1995

This work was supported under the grant 201/96/0918 given by the Czech Republic Grant Agency

Akademie věd České republiky

ÚSTAV INFORMATIKY A VÝPOČETNÍ TECHNIKY

Institute of Computer Science, Academy of Sciences of the Czech Republic

Pod vodárenskou věží 2, 182 07 Prague 8, Czech Republic

E-mail: ICS@uivt.cas.cz

Fax: (+422) 8585789 Phone: (+422) 846669, (+422) 66051111

Contents

1. Introduction to the UFO system	4
1.1. Philosophy of the UFO system	4
1.2. Execution of the UFO system	5
1.3. Input language	13
1.4. Problem description and method selection using the input language	16
1.5. The UFO environment	17
2. Problems solved using the UFO system	18
2.1. Specification of variables	21
2.2. Specification of the model function (dense problems)	22
2.3. Specification of the model function (sparse problems)	24
2.4. Objective functions for discrete approximation	25
2.5. Specification of the approximating functions (dense problems)	26
2.6. Specification of the approximating functions (sparse problems)	28
2.7. Objective functions for optimization of dynamical systems	30
2.8. Specification of the state functions	31
2.9. Specification of the initial functions	32
2.10. Specification of the subintegral function	32
2.11. Specification of the terminal function	33
2.12. Optimization with general constraints	34
2.13. Specification of the constraint functions (dense problems)	35
2.14. Specification of the constraint functions (sparse problems)	36
2.15. Additional specifications concerning optimization problems	38
3. Optimization methods in the UFO system	40
3.1. Heuristic methods	42
3.2. Conjugate direction methods	42
3.3. Variable metric methods	43
3.4. Modified Newton methods	45
3.5. Modified Gauss-Newton methods for nonlinear least squares and nonlinear equations	46
3.6. Quasi-Newton methods for nonlinear least squares and nonlinear equations	49
3.7. Quasi-Newton methods with limited storage for nonlinear equations	51
3.8. Biconjugate direction methods for nonlinear equations	52
3.9. Modified Brent method for nonlinear equations	52
3.10. Methods for linear programming problems	52
3.11. Methods for quadratic programming problems	53
3.12. Proximal bundle methods for nonsmooth optimization	53
3.13. Bundle-Newton methods for nonsmooth optimization	53
3.14. Methods for minimax problems	54
3.15. Recursive quadratic programming methods for nonlinear programming problems	55
3.16. Recursive minimax optimization methods for nonlinear programming problems	55
3.17. Inexact recursive quadratic programming methods for large sparse equality constrained nonlinear programming problems	56
3.18. Methods for initial value problems for ordinary differential equations	57
3.19. Methods for direction determination	58
3.20. Methods for stepsize selection	60
3.21. Methods for numerical differentiation	61
3.22. Methods for objective function evaluation in the case of dynamical systems optimization	62
3.23. Global optimization methods	62

4. Output specifications in the UFO system	65
4.1. Basic screen output	65
4.2. Extended screen output	65
4.3. Graphical screen output	66
4.4. Text file output	69
4.5. User supplied output	71
4.6. Tracing in the UFO control program	71
4.7. Error messages	71
5. Special tools of the UFO system	72
5.1. Checking external subroutines	72
5.2. Testing optimization methods	73
6. Applications of the UFO system (examples)	76
6.1. Optimization with simple bounds	76
6.2. Minimization of the sum of squares	77
6.3. Minimax approximation	79
6.4. Nonsmooth optimization	80
6.5. Optimization with linear constraints	81
6.6. Minimax approximation with linear constraints	83
6.7. Optimization with nonlinear constraints (nonlinear programming)	84
6.8. Global optimization	86
6.9. Large-scale optimization (sparse Hessian matrix)	88
6.10. Large-scale optimization (sparse Jacobian matrix)	89
6.11. Large-scale sum of squares optimization (sparse Jacobian matrix)	91
6.12. Large-scale nonlinear equations	93
6.13. Large-scale linear programming	94
6.14. Large-scale quadratic programming	95
6.15. Large-scale optimization with linear constraints	98
6.16. Large-scale optimization with nonlinear equality constraints	101
6.17. Optimization of dynamical systems - general integral criterion	104
6.18. Optimization of dynamical systems - special integral criterion	106
6.19. Initial value problem for ordinary differential equations	108
References	110
Appendix: Graphical screen output	117

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 the methods described in chapter 3.
3. Design and testing new optimization methods. The UFO system is a very useful tool for optimization algorithms development.

The special realization of the UFO system, which is 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 using a conversation between the user and the UFO system.
2. The batch mode. The problem specification and the method selection are realized using the input language. An input file, written in the input language, has to be prepared and stored.
3. The combined mode. Only the 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 using a convenient text editor.

The second phase is realized using the UFO macroprocessor. This macroprocessor 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 macroprocessor, 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 have 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.

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, a final program is executed and 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 macroprocessor 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 need not be compiled, are changed. Besides interface templates, which are a part of source modules, special templates controlling the UFO macroprocessor exist. An input file written in the input language is one of these special templates.

The UFO macroprocessor works with 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 five basic procedures GENER.BAT, UFOGO1.BAT, UFOGO2.BAT, COMPIL1.BAT and COMPIL2.BAT. The UFO macroprocessor is called if the statement

GENER *input_name*

is typed. Then the control program, which has to be compiled and loaded using the procedures COMPIL1.BAT or COMPIL2.BAT, is obtained. Furthermore, all the UFO system phases are performed if the statements

UFOGO1 *input_name*

or

UFOGO2 *input_name*

respectively are typed. Finally, the compilation of the control program P.FOR, which was obtained by the procedure GENER.BAT, followed by its loading and executing is started if the statements

COMPIL1 *output_name*

or

COMPIL2 *output_name*

respectively are typed. Here *input_name* is the first part of the batch file name that is used as an batch input for the control program generation and *output_name* is the first part of the text file that is used as an text output from the UFO system. The batch file name must always have the form *input_name*.UFO with the extension UFO and the text file name must have the form *output_name*.OUT with the extension OUT. If GENER, UFOGO1, UFOGO2 statements do not contain batch file specification then a full dialogue mode is considered (the batch file name is STANDARD.UFO in this case). If COMPIL1, COMPIL2 statements do not contain text file specification then the standard text file name is P.OUT. The difference between UFOGO1.BAT and UFOGO2.BAT or COMPIL1 and COMPIL2 consists in the fact that procedures with number 1 prepare programs for basic computations while procedures with number 2 prepare programs for using Microsoft CodeView debugger contained in Microsoft Fortran compiler. The UFOGO1 statement has the same meaning as the two statements GENER and COMPIL1 and the UFOGO2 statement has the same meaning as the two statements GENER and COMPIL2.

First we show how the full dialogue 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 type the statement UFOGO (without batch file specification), then the following questions appear. These we supplement together with answers.

? 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

? 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

? NF (0) ?

NUMBER OF VARIABLES

2

? IEXT (0) ?

TYPE OF EXTREMUM

0 - MINIMUM

1 - MAXIMUM

? FMODEL (*) ?

MODEL OF OBJECTIVE FUNCTION

FF = <FORTRAN_EXPRESSION>

FF = 1.0D2*(X(1)**2 - X(2))**2 + (X(1) - 1.0D0)**2

— ? GMODEL (*) ? —

MODEL OF GRADIENT OF OBJECTIVE FUNCTION

GF(1) = <FORTRAN_EXPRESSION>

GF(2) = <FORTRAN_EXPRESSION>

.

.

GF(NF) = <FORTRAN_EXPRESSION>

— ? HMODEL (*) ? —

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

NO - 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

? TEST (NO) ?

STANDARD TEST OF EXTERNAL SUBROUTINES:

- NO - NO TEST
- YES - PERFORM TEST AND CONTINUE
- ONLY - PERFORM ONLY TEST

? GRAPH (NO) ?

SPECIFICATION OF GRAPHICAL OUTPUT

- NO - GRAPHICAL OUTPUT SUPPRESSED
- YES - GRAPHICAL OUTPUT REQUIRED

? DISPLAY (NO) ?

SPECIFICATION OF EXTENDED SCREEN OUTPUT

- NO - EXTENDED SCREEN OUTPUT SUPPRESSED
- YES - EXTENDED SCREEN OUTPUT REQUIRED

? 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 - ADDITIONAL 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 (1) ?

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

? NOUT (0) ?

LEVEL OF BASIC SCREEN OUTPUT:

- 0 - BASIC FINAL OUTPUT
- 1 - EXTENDED FINAL OUTPUT

1

? MSELECT (1) ?

SELECTION OF OPTIMIZATION METHOD

- 1 - AUTOMATIC SELECTION OF METHOD
- 2 - MANUAL SELECTION OF METHOD
- 3 - MANUAL SELECTION OF METHOD AND IMPORTANT PARAMETERS
- 4 - MANUAL SELECTION OF METHOD AND ALL PARAMETERS

? OUTPUT () ?

USER SUPPLIED OUTPUT:

HERE THE RESULTS OBTAINED IN THE OPTIMIZATION PROCESS CAN BE USED FOR ADDITIONAL COMPUTATIONS AND FOR A SPECIFIC OUTPUT.

UFO PREPROCESSOR STOP

Each question is represented by one frame that contains the contents of the question (name of the macrovariable which has to be defined), the default value (in the brackets) and an explanation of the requirement. If no default value is wanted then the corresponding value or text has to be typed. The dialogue can be ended by pressing the key <! > .

The result of the UFO macroprocessor 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 adjustment:

```

*
* -----
* GLOBAL DECLARATIONS
* -----
*
INTEGER ITIME
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 XO(2)
REAL*8 GO(2)
INTEGER IMB

*
* commons placed here were omitted
* since they require a large space
*
CALL UYCLEA

*
* -----
* END OF DECLARATIONS
* -----
*
CALL UYINTP

*
* -----
* METHOD (1)
* -----
*
CALL UYINT1
X(1)=-1.2D0
X(2)=1.0D0
M=NF*(NF+1)/2
CALL UYTIM1(ITIME)
CALL UYCLST
NDECF=0
CALL UOOFU1
11000 CONTINUE
CALL U1FDU1
GOTO (11090,11010,11030,11040,11060,11080) ISB+1
11010 CONTINUE
11500 CALL UF1F01(GF,GF)
GOTO (11530,11510,11520) ISB+1
11510 CONTINUE
ASSIGN 11610 TO IMB
11600 CONTINUE
NFV=NFV+1
FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
GOTO IMB

```

```

11610 CONTINUE
      IF(TSS(NS).EQ.'UFXX') GOTO 11500
11520 CONTINUE
      CALL UFOGS2(X,IX,X,GF,HD)
      GOTO (11500,11510) ISB+1
11530 CONTINUE
      IF(TSS(NS).EQ.'UXFU') GOTO 11000
      IF(TSS(NS).EQ.'UUXX') GOTO 11040
      GOTO 11070
11030 CONTINUE
      CALL UYTRUG(X,GF,GF)
      CALL UO2FU3(X,GF,HF,X,X)
      CALL UYFUT1
      GOTO 11000
11040 CONTINUE
      CALL UUDSD1(HF)
      GOTO (11050,11010) ISB+1
11050 CONTINUE
      IF (ITERM.NE.0) GOTO 11000
      CALL UYCPSD(IX,HF,HD)
      CALL UYTRUH(X,HF)
      CALL UDDL1(HF,GF,S)
      CALL UD1TL1(GF,S)
      GOTO 11000
11060 CONTINUE
      CALL UYTRUS(X,X,XO,GF,GO,S,S)
11070 CONTINUE
      CALL USOLO1
      GOTO (11074,11072) ISB+1
11072 CONTINUE
      CALL UXVDIR(NF,R,S,XO,X)
      GOTO 11010
11074 CONTINUE
      IF (ITERS.LE.0) THEN
      CALL UYZERO(X,XO)
      ENDIF
      GO TO 11000
11080 CONTINUE
      CALL UYUPSD(X,IX,XO,GF,GO,HD)
      CALL UYTRUD(X,X,XO,GF,GO)
      CALL UUDBI1(HF,S,XO,GO)
      GOTO 11000
11090 CONTINUE
      CALL UO1FU2(X,X,X,X)
*
* -----
* END OF METHOD (1)
* -----
*
      CALL UYTIM2(ITIME)
      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 follows 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
MMAX=2*(2+1)/2
END
* -----

```

The results (screen output) obtained using this control program have the following form:

```

0 NIT= 52 NFV= 175 NFG= 0 NDC= 0 F= .308D-14 G= .818D-06
FF = .3081936474D-14
X = .9999999475D+00 .9999998968D+00
TIME= 0:00:00.06

```

Besides a dialogue, we can use the batch files written in the input language. This possibility will be explained in section 1.4. Here we note that a certain experience can be obtained using the demo-files PROB01.UFO, . . . , PROB19.UFO. These demo-files contain 19 test problems described in chapter 6. We can solve them using the statements UFOG01 PROB01, . . . , UFOG01 PROB19.

1.3. Input language

The form of the control program is determined using statements of the input language. The UFO system input language is based on the batch editing language (BEL) [119] and it contains three 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 macroprocessor.
3. Special macroinstructions. The macroinstructions control the UFO macroprocessor execution.

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

Macrovariables used in the UFO system begin with the character '\$' and they are supposed to be of the type character. Their contents are always in the form of a string which can be sometimes interpreted as an integer. The chief significance of the macrovariables is their use in substituting their contents 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 contents of \$FLOAT, \$HESF, \$TYPE, \$DECOMP, \$NUMBER and \$P are 'REAL*8' (it is standard), 'D', 'L', 'G' '1' and 'D' (it is standard) then we get REAL*8 W(100) or CALL UDDL1 or X(1)=1.0D0 respectively after the UFO preprocessor application. The contents of macrovariables can be defined by substitutions as will be shown later.

The macroinstructions are very important for the input language since they make the substitution of texts, change of macrovariables, branching, loops, etc., possible. We briefly describe the most useful of them. The more detailed description is given in [120].

1. Substitution. The substitution of a string for a macrovariable is specified by the macroinstruction \$MACRO='contents'. For example we have to set \$HESF='D', \$TYPE='L', \$DECOMP='G', \$NUMBER=1 (the integers need not to be substituted as strings) to obtain the result given above.
2. Substitution 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 can, moreover, append a new text to the old one so that it can be used repeatedly.

3. Logical macrovariables. The macrovariable \$DEF has logical contents. If we write \$DEF(MACRO) then the contents of \$DEF is either .TRUE., if the macrovariable \$MACRO was previously defined (by the substitution \$MACRO='contents' or using macroinstructions \$SET and \$ADD), or .FALSE. in the opposite case. This possibility can be used for branching. If we use the macroinstruction \$ERASE(MACRO) then the previously defined macrovariable \$MACRO becomes undefined (so that \$DEF(MACRO)=.FALSE.).
4. Branching. This possibility is very similar to the branching in the Fortran 77 language:

```

$IF(MACRO='string1')
  statements
$ELSEIF(MACRO='string2')
  statements
$ELSE
  statements
$ENDIF

```

Besides the relation '=' we can also use the other relations '<', '>', '<=', '>=', '<>' and the logical values .TRUE., .FALSE. and \$DEF(MACRO). The branching is used in the UFO macroprocessor stage and it has an influence in the form of the control program.

5. Loops. The basic looping macroinstruction has the form (similarly as in the Fortran 77 language):

```

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

```

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

```

$DO(I=1,NF,1)
  $DO(J=1,NC,1)
    CALL UKMCI1($I,$J,$I.0D0+$J.0D0,ICG,JCG,CG)
  $ENDDO
$ENDDO

```

then the UFO macroprocessor creates the sequence

```

CALL UKMCI1(1,1,1.0D0+1.0D0,ICG,JCG,CG)
CALL UKMCI1(1,2,1.0D0+2.0D0,ICG,JCG,CG)
CALL UKMCI1(1,3,1.0D0+3.0D0,ICG,JCG,CG)
CALL UKMCI1(2,1,2.0D0+1.0D0,ICG,JCG,CG)
CALL UKMCI1(2,2,2.0D0+2.0D0,ICG,JCG,CG)
CALL UKMCI1(2,3,2.0D0+3.0D0,ICG,JCG,CG)

```

6. Substitution of a file. Suppose we have a file with a name `file_name.extension`. Then we can include it into the control program using the macroinstructions

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

or

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

The main difference between these possibilities is that the macroinstruction `$INCLUDE` includes a text without change (it has to be a regular Fortran 77 text with a fixed format) while the macroinstruction `$SUBST` substitutes a text executed consecutively by the UFO macroprocessor (so it can contain the macrovariables and macroinstructions and it can be written in the free format). If the included file has the name `file_name.I` then we can use a simpler form without extension. For example the file `UZLINS.I` can be substituted using the macroinstruction `$SUBST('UZLINS')`.

7. Special macroinstructions. Besides macroinstructions of the batch editing language BEL, the input language contains special macroinstructions which control the UFO macroprocessor. These macroinstructions are the switches `$DIALOGUE` and `$BATCH` that define the dialogue mode and the batch mode respectively (an initial status is `$DIALOGUE`), as well as the special substitutions `$GLOBAL`, `$INITIATION`, `$INPUT`, `$OUTPUT`, `$METHOD`, `$TSTART`, `$TSTOP` and `$STANDARD`. At the same time, `$GLOBAL` includes the global declarations, `$INITIATION` substitutes an initiation of common variables, `$INPUT` and `$OUTPUT` insert user specified input and output respectively, `$METHOD` substitutes the part of control program corresponding to the optimization method, `$TSTART` or `$TSTOP` insert initial or final timing procedures respectively and `$STANDARD` includes the sequence:

```
$INITIATION
$INPUT
$METHOD
$IF($DEF(OUTPUT))
  $OUTPUT
$ENDIF
$END
```

The macroinstruction `$END` finishes the UFO macroprocessor execution.

8. Standard macrovariables. The macrovariables `$FLOAT` or `$P` have standard contents `'REAL*8'` or `'D'` respectively. This possibility has a meaning for a precision free notation. If we write

```
$FLOAT WA,WB
WA=2.0$(P)1
WB=1.0$(P)2
```

then after the UFO macroprocessor execution we have

```
REAL*8 WA,WB
WA=2.0D0
WB=1.0D2
```

The macrovariables `$FLOAT` and `$P` are defined in the installation template and they can be changed when we wish to use single precision computations

We have described the basic possibilities of the input language that are sufficient for preparing the batch mode input file. More details are given in [119] and [120].

1.4. Problem description and method selection using the input language

If we want to process either the batch mode or the mixed mode we have to prepare an input file written in the input language. This input file prescribes the structure of the control program. If some macrovariable is used, it has to have been previously defined. Therefore definitions of macrovariables usually lie in the beginning of the input file. Many macrovariables are used for a definition of the optimization problem. The most important among them are macrovariables defining problem functions:

\$FMODEL - definition of a model function value
 \$GMODEL - definition of a model function gradient
 \$HMODEL - definition of a model function Hessian matrix
 \$DMODEL - definition of a model function derivatives (with respect to state space variables)
 \$FMODELA - definition of approximating functions values
 \$GMODELA - definition of approximating functions gradients
 \$HMODELA - definition of approximating functions Hessian matrices
 \$DMODELA - definition of approximating functions derivatives (with respect to state space variables)
 \$FMODEL - definition of constraint functions values
 \$GMODEL - definition of constraint functions gradients
 \$HMODEL - definition of constraint functions Hessian matrices
 \$FMODELE - definition of state space functions values
 \$GMODELE - definition of state space functions gradients
 \$DMODELE - definition of state space functions derivatives (with respect to state space variables)
 \$FMODELY - definition initial value functions values
 \$GMODELY - definition initial value functions gradients

First we show a simple example which corresponds to the problem already used for a full dialogue mode demonstration (the Rosenbrock function):

```
$SET(INPUT)
  X(1)=-1.2D0; X(2)=1.0D0
$ENDSET
$SET(FMODEL)
  FF=100.0D0*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
$NF=2
$MOUT=2
$BATCH
$STANDARD
```

Using the macrovariable \$INPUT, we specify initial values of variables. Using the macrovariable \$FMODEL we specify the model function value (the model function gradient is not specified and it will be computed numerically). The macrovariable \$NF defines the number of variables and \$KOUT and \$MOUT are print specifications. The macroinstruction \$BATCH switches a mode to the batch one. The macroinstruction \$STANDARD defines a 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 using the subroutine EFFU01 or it is specified in the file FVAL.FOR. Then we can write:

```

$SET(FMODEL)
  CALL EFFU01(NF,X,FF,NEXT)
$ENDSET

```

or

```

$SET(FMODEL)
  $INCLUDE('FVAL.FOR')
$ENDSET

```

or

```

$SET(FMODEL)
  $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 have 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, two exceptions in writing source text, forced by the UFO macroprocessor, must be satisfied: All comments have to begin by the character '*' and the continuation line character have to be always '&'.

1.5. The UFO environment

The UFO environment can be used on personal computers (PC) with processors 286/386/486/586, with the operating system MS DOS version 5.0 or higher and with the Microsoft FORTRAN 77 compiler version 5.0 or higher.

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 the 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 (for example Turbo Pascal). 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 [120]) . 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 the 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 UFOGO1 (with the present input file). An easier possibility is pressing the keys <Alt-1> . Similarly, pressing the keys <Alt-2> corresponds to the statement UFOGO2 and pressing the keys <Alt-9> has the same effect as the statement GENER. Furthermore, if the control program P.FOR is loaded in the source program editor, pressing the keys <Alt-3> and <Alt-4> has the same effect as the statements COMPIL1 and COMPIL2 respectively, and pressing the keys <Alt-5> causes an exit from the UFO environment.

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 $F : R^n \rightarrow R$ 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 $f^F : R^n \rightarrow R$ 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 $f^F, g_i^F, 1 \leq i \leq n$, 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 $f^F, g_i^F, 1 \leq i \leq n, h_{ij}^F, 1 \leq i \leq n, 1 \leq j \leq n$, are real coefficients.

\$MODEL='AF' - sum of functions minimization. In this case

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

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

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

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

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

\$MODEL='AP' - sum of powers minimization. In this case

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

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

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

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

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

`$MODEL='DF'` - minimization of 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

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

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

`$MODEL='DQ'` - minimization of sum of squares 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

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

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

`$MODEL='NO'` - solving an initial value problem for a system of ordinary differential equations. In this case

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

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

The objective function defined by the choice `$MODEL='AQ'` can be used for the solution of a system of nonlinear equations

$$f_k^A(x) = 0, \quad 1 \leq k \leq n_A$$

In this case we suppose $n_A = n$. This case is considered separately, since for $n_A = n$ special methods for systems of nonlinear equations can be used.

The model function $f^F : R^n \rightarrow R$ 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='NO'` - Hessian matrix is not used.

The default option is `$HESF='D'`. The approximating functions $f_k^A : R^n \rightarrow R$, $1 \leq k \leq n_A$, 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.

$\$JACA='NO'$ - Jacobian matrix is not used.

If the approximating functions are used then we can choose several types of the Hessian matrix representation. 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
 $\$HESF='NO'$ - Hessian matrix is not used.

If $\$JACA='D'$, then it must be either $\$HESF='D'$ or $\$HESF='NO'$. If $\$JACA='S'$, then we can specify all types of Hessian matrices ($\$HESF='D'$, $\$HESF='S'$, $\$HESF='B'$, $\$HESF='NO'$). 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, terminal function, state function and initial function, 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 whole R^n (unconstrained case) or it can be defined by box constraints

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

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

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

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

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

where $f_k^C : R^n \rightarrow R$, $1 \leq k \leq n_C$, 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 $f_k^C : R^n \rightarrow R$, $1 \leq k \leq n_C$, 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 must be \$HESF='D' or \$HESF='NO'. If \$JACC='S', then must be \$HESF='S' or \$HESF='NO'.

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

1. Minimization of maximum (minimax) and nonsmooth optimization is 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 that 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, GF, HF, FA, GA, FC, GC instead of x , f^F , g^F , h^F , f^A , g^A , f^C , g^C . This new notation corresponds to the notation of the variables and 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

X(I) -	unbounded	,	if IX(I) = 0
XL(I) ≤	X(I)	,	if IX(I) = 1
	X(I) ≤ XU(I)	,	if IX(I) = 2
XL(I) ≤	X(I) ≤ XU(I)	,	if IX(I) = 3
X(I) -	constant	,	if IX(I) = 5

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

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

```
$KBF=2; $NF=4
$SET(INPUT)
  X(1)=x1
  X(2)=x2; IX(2)=1; XL(1)=x1F
  X(3)=x3; IX(3)=3; XL(3)=x3L; XU(3)=x3U
  X(4)=x4; 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 $XN(I)$, $1 \leq I \leq NF$, are determined automatically so that $X(I)/XN(I)=1$, $1 \leq I \leq NF$, 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

$$F(X) = + FF(X) \text{ if } \$IEXT = 0 \text{ (minimization)}$$

or

$$F(X) = - FF(X) \text{ if } \$IEXT = 1 \text{ (maximization)}$$

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 \$FMODEL:

```
$SET(FMODEL)
  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 \$GMODEL:

```
$SET(GMODEL)
  GF(1) = derivative ∂FF(X)/ ∂X(1)
  GF(2) = derivative ∂FF(X)/ ∂X(2)
  GF(3) = derivative ∂FF(X)/ ∂X(3)
  ———
  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 \$HMODEL. If \$HESF='D', then the Hessian matrix is assumed to be dense and we specify only its upper half:

```
$SET(HMODEL)
  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 \$GMODEL or \$HMODEL 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 include the models $\$FMODEL$ and $\$GMODEL$ into the common model $\$FGMODEL$. Similarly we can include the models $\$FMODEL$, $\$GMODEL$ and $\$HMODEL$ into the common model $\$FGHMODEL$.

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 takes at most $O(NF)$ simple operations).
- $\$KCF=2$ - evaluation of the model function $FF(X)$ is of medium complexity (it takes at least $O(NF)$ complicated operations and at most $O(NF^2)$ simple operations).
- $\$KCF=3$ - evaluation of the model function $FF(X)$ is extremely difficult (it takes at least $O(NF^2)$ complicated operations or $O(NF^3)$ 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 a 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 is linear of the form

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

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

```

$ADD(INPUT)
  FF = constant value
  GF(1) = constant derivative  $\partial FF(X)/\partial X(1)$ 
  GF(1) = constant derivative  $\partial FF(X)/\partial X(1)$ 
  GF(2) = constant derivative  $\partial FF(X)/\partial X(2)$ 
  GF(3) = constant derivative  $\partial FF(X)/\partial X(3)$ 
  —
  GF(NF) = constant derivative  $\partial FF(X)/\partial X(NF)$ 
$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 is quadratic of the form

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

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 $\$FMODEL$, $\$GMODEL$ and $\$HMODEL$ as in the general case. The coefficients FF (constant value) and $GF(I)$, $1 \leq I \leq$

NF, (constant gradient) are specified in the same way as in the linear case. The coefficients HF(K), $1 \leq K \leq NF*(NF+1)/2$, (the constant Hessian matrix) must be specified using the macrovariable \$INPUT. If \$HESF='D', then 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 that take into account the sparsity pattern of the Hessian matrix HF. This possibility decreases 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 (eg. two times) since it is used for the declaration of working fields. For example, if we have the Hessian matrix

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

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, second derivatives of the model function can be specified by using the macrovariable \$HMODEL. If \$HESF='S', then 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(HMODEL F)
HF(1)=h11F; HF(2)=h12F; HF(3)=h13F; HF(4)=h15F
HF(5)=h22F; HF(6)=h24F; HF(7)=h33F; HF(8)=h35F
HF(9)=h44F; HF(10)=h55F
$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 specification:

```

$ADD(INPUT)
HF(1)=h11F; HF(2)=h12F; HF(3)=h13F; HF(4)=h15F
HF(5)=h22F; HF(6)=h24F; HF(7)=h33F; HF(8)=h35F
HF(9)=h44F; HF(10)=h55F
$ENDADD

```

2.4. Objective functions for discrete approximation

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

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

or

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

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 the form:

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

or

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

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 simplest and the most efficient for such a choice, we can use the specification \$MODEL='AQ' in this case (minimization of sum of squares). Moreover, \$MODEL='AQ' is formally set whenever we chose \$MODEL='AP' and \$REXP=2.

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

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

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

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

for \$KBA=0, or

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

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

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

for \$KBA=1, where $FA(KA;X)$, $1 \leq KA \leq NA$, 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 at most two additional fields, AM and AW. The vector AM usually contains frequently used observations which can be included into the functions $FA(KA;X)$, $1 \leq KA \leq NA$, in the case of the simple objective function. Observations $AM(KA)$, $1 \leq KA \leq NA$, are specified by using the macrovariable \$INPUT. Their default values are $AM(KA)=0$, $1 \leq KA \leq NA$. The vector AW serves for possible scaling specified by the option \$NORMA:

- \$NORMA=0 - no scaling is performed. In this case $AW(KA)=1$, $1 \leq KA \leq NA$.
- \$NORMA=1 - scaling parameters are determined automatically so that $AW(KA)=|AM(KA)|$, $1 \leq KA \leq NA$.
- \$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 by using the statement \$NA=number_of_ functions in all the above cases.

2.5. Specification of the approximating functions (dense problems)

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

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

The first derivatives of the approximating functions are specified by using the macrovariable \$GMODELA:

```
$SET(GMODELA)
  GA(1) = derivative ∂FA(KA;X)/∂X(1)
  GA(2) = derivative ∂FA(KA;X)/∂X(2)
  GA(3) = derivative ∂FA(KA;X)/∂X(3)
  ———
  GA(NF) = derivative ∂FA(KA;X)/∂X(NF)
      (for a given index KA and given values of variables X(I), 1 ≤ I ≤ NF)
$ENDSET
```

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

```

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

```

If the macrovariables \$GMODELA or \$HMODELA 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 first derivatives of the approximating functions $FA(KA;X)$, $1 \leq KA \leq NA$, together with their values, we can collect the models \$FMODELA and \$GMODELA into the common model \$FGMODELA. Similarly we can collect the models \$FMODELA, \$GMODELA and \$HMODELA into the common model \$FGHMODELA.

To improve the efficiency of the computation, we can specify additional information about the approximating functions $FA(KA;X)$, $1 \leq KA \leq NA$. 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  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are very easy
              (they take at most  $O(NF)$  simple operations).
$KCA=2      - evaluations of the approximating functions  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are of medium
              complexity (they take at least  $O(NF)$  complicated operations and at most  $O(NF^2)$ 
              simple operations).
$KCA=3      - evaluations of the approximating functions  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are extremely
              difficult (they take at least  $O(NF^2)$  complicated or  $O(NF^3)$  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  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are smooth and well-
              conditioned.
$KSA=2      - the approximating functions  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are smooth but ill-
              conditioned.
$KSA=3      - the approximating functions  $FA(KA;X)$ ,  $1 \leq KA \leq NA$ , are nonsmooth.

```

The option \$KSA=1 is default.

If some of the approximating functions are linear having the form

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

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 $AG((KA-1)*NF+I)$, $1 \leq KA \leq NAL$, $1 \leq I \leq NF$, are specified using the macrovariable \$INPUT and the macrovariables \$FMODELA,

\$GMODELA, \$HMODELA 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 that take into account the sparsity pattern of the Jacobian matrix AG. This possibility decreases 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

$$GA(1; X) = [g_{11}^A, 0, 0, g_{14}^A],$$

$$GA(2; X) = [0, g_{22}^A, 0, g_{24}^A],$$

$$GA(3; X) = [0, 0, g_{33}^A, 0],$$

$$GA(4; X) = [g_{41}^A, g_{42}^A, g_{43}^A, 0],$$

$$GA(5; X) = [0, 0, g_{53}^A, g_{54}^A],$$

and the Jacobian matrix

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

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)=5
$ENDADD
```

As in the case of the dense problem, the first derivatives of the approximating functions can be specified by using the macrovariable \$GMODELA. If \$JACA='S', then only nonzero elements of the gradients are specified. For the above example the specification has 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

```

If some of the approximating functions are linear (i.e. if \$NAL>0) and if \$JACA='S', then the coefficients AG(K), $1 \leq K \leq \text{IAG}(\text{NAL}+1)-1$ (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 the specification:

```

$ADD(INPUT)
    AG(1)=g11A; AG(2)=g14A; AG(3)=g22A; AG(4)=g24A
    AG(5)=g33A; AG(6)=g41A; AG(7)=g42A; AG(8)=g43A
    AG(9)=g53A; AG(10)=g54A
$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 that serves for direct input of individual Jacobian matrix elements. The procedure UKMAI1 is formally called by using the statement

```
CALL UKMAI1(K,I,GA KI,IAG,JAG,AG)
```

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

```

$ADD(INPUT)
    CALL UKMAI1(1,1,g11A,IAG,JAG,AG)
    CALL UKMAI1(1,4,g14A,IAG,JAG,AG)
    CALL UKMAI1(2,2,g22A,IAG,JAG,AG)
    CALL UKMAI1(2,4,g24A,IAG,JAG,AG)
    CALL UKMAI1(3,3,g33A,IAG,JAG,AG)
    CALL UKMAI1(4,1,g41A,IAG,JAG,AG)
    CALL UKMAI1(4,2,g42A,IAG,JAG,AG)
    CALL UKMAI1(4,3,g43A,IAG,JAG,AG)
    CALL UKMAI1(5,3,g53A,IAG,JAG,AG)
    CALL UKMAI1(5,4,g54A,IAG,JAG,AG)
$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='NO' - Hessian matrix is not used.

This specification serves only 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 the form:

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

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

$$\frac{dYA(KE; TA)}{dTA} = FE(KE; X, YA(TA), TA), \quad YA(KE; TAMIN) = FY(KE; X)$$

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

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

$$F(X) = \frac{1}{2} \int_{TAMIN}^{TAMAX} \sum_{KE=1}^{NE} WE(KE; TA) * (YA(KE; TA) - YE(KE; TA))^2 dTA \\ + \frac{1}{2} \sum_{KE=1}^{NE} EW(KE) * (YA(KE; TAMAX) - EY(KE))^2$$

At the same time

$$\frac{dYA(KE; TA)}{dTA} = FE(KE; X, YA(TA), TA), \quad YA(KE; TAMIN) = FY(KE; X)$$

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

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

$$\frac{dYA(KE; TA)}{dTA} = FE(KE; YA(TA), TA), \quad YA(KE; TAMIN) \text{ is given}$$

where FE(KE; YA(TA), TA), $1 \leq KE \leq NE$, are smooth state functions.

The number of differential equations NE must be specified by using the statement \$NE=number_of_differential_equations in all the above cases.

2.8. Specification of the state functions

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

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

The first derivatives of the state functions according to the variables are specified by using the macrovariable \$GMODELE:

```
$SET(GMODELE)
  GE(1) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial X(1)$ 
  GE(2) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial X(2)$ 
  GE(3) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial X(3)$ 
  —
  GE(NF) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial X(NF)$ 
      (for a given index KE, given vector of variables X,
       given vector of state variables YA(TA) and given time TA)
$ENDSET
```

The first derivatives of the state functions according to the state variables are specified by using the macrovariable \$DMODELE:

```
$SET(DMODELE)
  DE(1) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial YA(1)$ 
  DE(2) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial YA(2)$ 
  DE(3) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial YA(3)$ 
  —
  DE(NE) = derivative  $\partial FE(KE;X,YA(TA),TA)/\partial YA(NE)$ 
      (for a given index KE, given vector of variables X,
       given vector of state variables YA(TA) and given time TA)
$ENDSET
```

If it is advantageous to compute first derivatives of the state functions $FE(KE;X,YA(TA),TA)$, $1 \leq KE \leq NE$, together with their values, we can collect the models \$FMODELE, \$GMODELE and \$DMODELE into the common model \$FGDMODELA. Partially we can collect the models \$FMODELE, \$GMODELE or \$FMODELE, \$DMODELE or \$GMODELE, \$DMODELE into the common models \$FGMODELE or \$FDMODELE or \$GDMODELE 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 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, given vector of variables X,
       given vector of state variables YA(TA) and 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. Values of the initial functions are specified by using the macrovariable \$FMODELY:

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

The first derivatives of the initial functions according to the variables are specified by using the macrovariable \$GMODELY:

```
$SET(GMODELY)
  GE(1) = derivative  $\partial FY(KE;X)/\partial X(1)$ 
  GE(2) = derivative  $\partial FY(KE;X)/\partial X(2)$ 
  GE(3) = derivative  $\partial FY(KE;X)/\partial X(3)$ 
  —
  GE(NF) = derivative  $\partial FY(KE;X)/\partial X(NF)$ 
          (for a given index KE and given vector of variables X)
$ENDSET
```

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

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(3) = initial value YA(3,TAMIN)
  —
  YA(NE) = initial value YA(NE,TAMIN)
$ENDADD
```

2.10. Specification of the subintegral function

If \$MODEL='DF', then 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. 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, given vector of state variables YA(TA)
       and given time TA)
$ENDSET
```

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

```

$SET(GMODEL A)
  GA(1) = derivative  $\partial FA(X, YA(TA), TA) / \partial X(1)$ 
  GA(2) = derivative  $\partial FA(X, YA(TA), TA) / \partial X(2)$ 
  GA(3) = derivative  $\partial FA(X, YA(TA), TA) / \partial X(3)$ 
  —
  GA(NF) = derivative  $\partial FA(X, YA(TA), TA) / \partial X(NF)$ 
  (for a given vector of variables X, given vector of state variables YA(TA)
  and given time TA)
$ENDSET

```

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

```

$SET(DMODEL A)
  DA(1) = derivative  $\partial FA(X, YA(TA), TA) / \partial YA(1)$ 
  DA(2) = derivative  $\partial FA(X, YA(TA), TA) / \partial YA(2)$ 
  DA(3) = derivative  $\partial FA(X, YA(TA), TA) / \partial YA(3)$ 
  —
  DA(NE) = derivative  $\partial FA(X, YA(TA), TA) / \partial YA(NE)$ 
  (for a given vector of variables X, given vector of state variables YA(TA)
  and given time TA)
$ENDSET

```

If it is advantageous to compute first derivatives of the subintegral function $FA(X, YA(TA), TA)$ together with its value, we can collect the models \$FMODEL A, \$GMODEL A and \$DMODELA into the common model \$FGDMODELA. Partially we can collect the models \$FMODEL A, \$GMODEL A or \$FMODEL A, \$DMODELA or \$GMODEL A, \$DMODELA into the common models \$FGDMODELA or \$FDMODEL A or \$GDMODELA respectively.

If \$MODEL='DQ' and the objective function contains an integral part, we have to set \$MODEL A='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', then 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. Value of the terminal function is specified by using the macrovariable \$FMODEL F:

```

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

```

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

```

$SET(GMODEL F)
  GF(1) = derivative  $\partial FF(X, YA(TAMAX), TAMAX) / \partial X(1)$ 
  GF(2) = derivative  $\partial FF(X, YA(TAMAX), TAMAX) / \partial X(2)$ 

```

```

GF(3) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂X(3)
-----
GF(NF) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂X(NF)
(for a given vector of variables X, given vector of state variables YA(TAMAX)
and 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(3) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂YA(3)
-----
DF(NE) = derivative ∂FF(X,YA(TAMAX),TAMAX)/∂YA(NE)
(for a given vector of variables X, given vector of state variables YA(TAMAX)
and given time TAMAX)
$ENDSET

```

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

If \$MODEL='DQ' and the objective function contains a terminal part, we have to set \$MODEL='YES' and define the coefficients EW(KE) and EY(KE), $1 \leq KE \leq NE$, 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(3) = value EW(3); EY(3) = value EY(3)
-----
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 $1 \leq KC \leq NC$. The option \$KBC=2 must be chosen if IC(KC)=3 for at least one index $1 \leq KC \leq NC$. Then two different fields XL(KC) and XU(KC), $1 \leq KC \leq NC$ are declared. In the opposite case we set \$KBC=1 and only one common field XL(KC)=XU(KC), $1 \leq KC \leq NC$ is declared.

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

```
$KBF=2; $NC=3
$ADD(INPUT)
  IC(1)=1; CL(1)=c1L
  IC(2)=1; CL(2)=c2L
  IC(3)=3; CL(3)=c3L; CU(3)=c3U
$ENDADD
```

2.13. Specification of the constraint functions (dense problems)

The constraint functions $FC(KC;X)$, $1 \leq KC \leq NC$, must be defined by the user either directly in the full dialogue mode, or by using corresponding macrovariables in the batch (or mixed) mode. Values of the constraint functions are specified by using the macrovariable \$FMODEL: C

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

The first derivatives of the constraint functions are specified by using the macrovariable \$GMODEL: C

```
$SET(GMODEL)
  GC(1) = derivative ∂FC(KC;X)/∂X(1)
  GC(2) = derivative ∂FC(KC;X)/∂X(2)
  GC(3) = derivative ∂FC(KC;X)/∂X(3)
  —
  GC(NF) = derivative ∂FC(KC;X)/∂X(NF)
  (for a given index KC and given values of variables X(I), 1 ≤ I ≤ NF)
$ENDSET
```

The second derivatives of the constraint functions are specified by using the macrovariable \$HMODEL: C. If \$JACC='D', then the Hessian matrices are assumed to be dense and we specify only their upper half:

```
$SET(HMODEL)
  HC(1) = derivative ∂2FC(KC;X)/∂X(1)2
  HC(2) = derivative ∂2FC(KC;X)/∂X(1)∂X(2)
  HC(3) = derivative ∂2FC(KC;X)/∂X(2)2
  HC(4) = derivative ∂2FC(KC;X)/∂X(1)∂X(3)
  HC(5) = derivative ∂2FC(KC;X)/∂X(2)∂X(3)
  HC(6) = derivative ∂2FC(KC;X)/∂X(3)2
  —
  HC(NF*(NF+1)/2) = derivative ∂2FC(KC;X)/∂X(NF)2
  (for a given index KC and given values of variables X(I), 1 ≤ I ≤ NF)
$ENDSET
```

If the macrovariables \$GMODEL or \$HMODEL 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 first derivatives of the constraint functions $FC(KC;X)$, $1 \leq KC \leq NC$, together with their values, we can collect the models \$FMODEL and \$GMODEL into the common model \$FGMODEL: C

Similarly we can collect the models \$FMODEL, \$GMODEL and \$HMODEL into the common model \$FGHMODEL.

To improve the efficiency of the computation, we can specify additional information about the constraint functions $FC(KC;X)$, $1 \leq KC \leq NC$. 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 $FC(KC;X)$, $1 \leq KC \leq NC$, are very easy (they take at most $O(NF)$ simple operations).
- \$KCC= 2 - evaluations of the constraint functions $FC(KC;X)$, $1 \leq KC \leq NC$, are of medium complexity (they take at least $O(NF)$ complicated operations and at most $O(NF^2)$ simple operations).
- \$KCC= 3 - evaluations of the constraint functions $FC(KC;X)$, $1 \leq KC \leq NC$, are extremely difficult (they take at least $O(NF^2)$ complicated or $O(NF^3)$ simple operations).

The option \$KCC=2 is default.

If some of the constraint functions are linear having the form

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

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 \$FMODEL, \$GMODEL, \$HMODEL 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 that take into account the sparsity pattern of the Jacobian matrix CG. This possibility decreases 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 (two times say) since it is used for declaration of working fields. For example if we have the gradients

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

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

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

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

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

and the Jacobian matrix

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

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)=5
$ENDADD

```

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

```

$SET(GMODEL)
  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

```

If some of the constraint functions are linear (i.e. if \$NCL>0) and if \$JACC='S', then the coefficients CG(K), 1 ≤ K ≤ 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 the specification:

```

$ADD(INPUT)
  CG(1)=g11C; CG(2)=g14C; CG(3)=g22C; CG(4)=g24C
  CG(5)=g33C; CG(6)=g41C; CG(7)=g42C; CG(8)=g43C
  CG(9)=g53C; CG(10)=g54C
$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 that serves for direct input of individual Jacobian matrix elements. The procedure UKMCI1 is formally called by using the statement

```
CALL UKMCI1(K,I,GCKI,ICG,JCG,CG)
```

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

```
$ADD(INPUT)
  CALL UKMCI1(1,1,g11C,ICG,JCG,CG)
  CALL UKMCI1(1,4,g14C,ICG,JCG,CG)
  CALL UKMCI1(2,2,g22C,ICG,JCG,CG)
  CALL UKMCI1(2,4,g24C,ICG,JCG,CG)
  CALL UKMCI1(3,3,g33C,ICG,JCG,CG)
  CALL UKMCI1(4,1,g41C,ICG,JCG,CG)
  CALL UKMCI1(4,2,g42C,ICG,JCG,CG)
  CALL UKMCI1(4,3,g43C,ICG,JCG,CG)
  CALL UKMCI1(5,3,g53C,ICG,JCG,CG)
  CALL UKMCI1(5,4,g54C,ICG,JCG,CG)
$ENDADD
```

The main advantage of the last possibility is the fact that it is not necessary to specify beforehand the fields ICG and JCG. If the number of 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,ICG,JCG,CG,LCG)
```

where K is an index of a given constraint function (row of the Jacobian matrix), I is an index of a given variable (column of the Jacobian matrix), GCKI is a numerical value of the element $\partial FC(K;X)/\partial X(I)$ 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 a 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 exponentials. 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, that usually contains the starting point in its region of attractivity is found.
 \$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 \$FGMODEL and \$FGHMODEL for a common specification of the value, the gradient and the Hessian matrix

of the model function. Similarly we cannot use the common models \$FGMODELA and \$FGHMODELA for a 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 the objective function evaluation, penalty function definition, direction determination, quadratic programming solution, stepsize selection, and variable metric update. 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='SM' - sequential (or recursive) minimax optimization methods for general dense problems.
\$FORM='SE' - inexact sequential (or recursive) quadratic programming methods for sparse equality constrained problems.

Sections 3.1 - 3.14 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 3.15 - 3.17. Basic parts of optimization methods are described in Sections 3.18 - 3.22. Section 3.23 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 that use no matrices. This class contains conjugate direction methods, variable metric methods with limited storage and difference versions of the truncated Newton method.
\$CLASS='VM' - variable metric methods that use an approximation of the Hessian matrix which is updated in each iteration.
\$CLASS='MN' - modified Newton methods that use the Hessian matrix computed either analytically or numerically.
\$CLASS='GN' - modified Gauss-Newton methods for nonlinear least squares problems that 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='BD' - biconjugate direction methods for nonlinear equations.
\$CLASS='BR' - modified Brent method for nonlinear equations.
\$CLASS='LP' - special methods for linear programming problems.
\$CLASS='QP' - special methods for quadratic programming problems.
\$CLASS='BM' - proximal bundle methods for nonsmooth optimization.
\$CLASS='BN' - bundle-Newton 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 type of the method, kind of the matrix decomposition and 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.
`$TYPE='P'` - pattern search method of Hooke and Jeeves.
`$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 [43].
`$DECOMP='S'` - the LDL^T decomposition with permutations is used as an input for the direction determination. This decomposition is usually obtained by the Schnabel-Eskow algorithm [108].
`$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 [12].
`$DECOMP='I'` - the inverse of a symmetric matrix is used as an input for the direction determination.
`$DECOMP='R'` - the $R^T R$ decomposition without permutation is used as an input for the direction determination. This decomposition is usually obtained by the recursive QR factorization [62].
`$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 [15].
`$DECOMP='A'` - the rectangular matrix is used as an input for the direction determination.
`$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 .
`$DECOMP='E'` - the general square matrix is used as an input for the direction determination in the case `NA=NF` (system of nonlinear equations).
`$DECOMP='K'` - the indefinite Karush-Kuhn-Tucker matrix is used as an input for the direction determination in the equality constrained case.

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 3.19.

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 that are familiar with optimization methods.

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

3.1. Heuristic methods

Heuristic (or comparative) methods are specified by the statement `$CLASS='HM'`. These methods can be used only for small-size problems (with at most 10 variables). 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`:

`$TYPE='P'` - pattern search method of Hooke and Jeeves [56].
`$TYPE='S'` - simplex method of Nelder and Mead [93].

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 `$KCA=1`). The main advantage of conjugate direction methods is that matrices are not used (implicitly `$HESF='NO'`). This fact highly decreases storage requirements.

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

`$DECOMP='C'` - conjugate gradient methods. These methods are the simplest ones from all conjugate direction methods and they require the fewest storage requirements. However, they usually consume a greater number of function evaluations than other conjugate direction methods.
`$DECOMP='V'` - variable metric methods with limited storage. These methods allow us to prescribe storage requirements using the number of VM steps (the number of necessary used vectors is approximately two times greater than the number of VM steps). The number of VM steps is specified by the macrovariable `$MF`. Variable metric methods with limited storage usually consume fewer function evaluations than conjugate gradient methods.
`$DECOMP='M'` - inexact difference version of the modified Newton method [25]. This method is implemented either as the line search method or as the trust region method. It can be very efficient but, since it consumes a greater number of gradient evaluations, it can be slower than other conjugate direction methods, particularly if the objective function is more complicated (`$KCF>1` or `$KCA>1`).

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

`$NUMBER=1` - basic conjugate gradient methods described in [71]. The individual methods are specified by using the macrovariables `$MET`, `$MET1` and `$MET2`.
`$NUMBER=2` - generalized conjugate gradient methods introduced in [58]. The individual methods are specified by using the macrovariable `$MET1`.

If `$MET=0`, then the steepest descent method is used. If `$MET=1`, then the Fletcher-Reeves method [36] is used. If `$MET=2`, then the Polak-Ribiere method [99] is used. If `$MET=3`, then the Hestenes-Stiefel method [54] is used. The macrovariable `$MET1` specifies the restart procedure as it is described in [71]. If `$MET1=1`, then a restarted CG method with positive parameter is used. If `$MET1=2`, then a bounded CG method with positive parameter is used. If `$MET1=3`, then a bounded CG method with positive lower bound is used. If `$MET1=4`, then a CG method with the Powell restart is used. If `$MET1=5`, then a CG method with the test on conjugacy is used. If `$MET1=6`, then a CG method with the test on orthogonality is used. The macrovariable `$MET2` specifies the scaling parameter as it is described in [71] (`$MET2=1` for suppressed scaling and `$MET2=2` for scaling in each iteration).

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

`$NUMBER=1` - The BFGS method with limited storage described in [94]. The default number of VM steps is `$MF=5`.

`$NUMBER=2` - The extended BFGS method with limited storage described in [59]. The default number of VM steps is `$MF=3`.

Both these methods are realized by using various scaling techniques [66], specified by the macrovariable `$MET1`. If `$MET1=1`, then scaling is suppressed. If `$MET1=2`, then scalar scaling is used. If `$MET1=3`, then diagonal scaling is used. If `$MET1=4`, then scalar and diagonal scalings are used simultaneously.

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

L-C-1, L-V-1,
L-C-2, L-V-2,
L-M-3,
G-M-3.

The default choice is L-C-1. In both the box constrained and the linearly constrained cases we cannot use specifications with `$DECOMP='M'`. Conjugate direction methods can be used also for sparse linear constraints when `$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`:

`$UPDATE='B'` - the Broyden family [9]. Variable metric methods from this family are the most commonly used ones since they are very robust and efficient.
`$UPDATE='D'` - the Davidon family [22]. 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`, and `$MET2`. The macrovariable `$MET` determines the variable metric update. If `$MET=1`, then the BFGS method [9], [31], [46], [110] is used. If `$MET=2`, then the DFP method [21], [35] is used. If `$MET=3`, then the Hoshino method [57] is used. If `$MET=4`, then the safeguarded rank-one method [70] is used. If `$MET=5`, then the optimally conditioned method [22] is used. If `$MET=6`, then the rank-one based method [70] from the preconvex part of the Broyden family is used. If `$MET=7`, then the variationally derived method [73] from the preconvex part of the Broyden family is used. If `$MET=8`, then the heuristic method [76] is used. If `$MET=9`, then the method [129] derived from the matrix decomposition is used. If `$MET=10`, then the method [130] which minimizes the angle between the direction vector and the negative gradient is used. If `$MET=11`, then the method [76] which minimizes the norm of the direction vector is used. If `$MET=12`, then the least prior deviation method [89] is used. The default value is `$MET=1`. If we specify `$DECOMP='M'`, then we can use only the values `$MET=1,2,3,4`.

The macrovariable `$MET1` determines the Oren (scaling) parameter [96]. If `$MET1=1`, then no scaling is used. If `$MET1=2`, then initial scaling [111] is used. If `$MET1=3`, then controlled scaling [73] is used. If `$MET1=4`, then simple controlled scaling [81] is used. If `$MET1=5`, then scaling in each

iteration is used. The default value is \$MET1=3. The scaling parameter is determined by using heuristic rules given in [76].

The macrovariable \$MET2 determines a value of the Biggs (nonquadratic model) parameter [4]. If \$MET2=1, then the unit value is used. If \$MET2=2, then the Spedicato value [112] is used. If \$MET2=3, then the modified Spedicato value [76] is used. If \$MET2=4, then the value determined from the homogeneous model [76] is used. If \$MET2=5, then the value determined from the cubic model [5] is used. The default value is \$MET2=2.

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

```

L-G-1,  L-B-1,  L-I-1,
                                     L-M-3,
G-G-1,  G-B-1
                                     G-M-3,
                                     G-M-5,
T-G-1,
T-G-2.
```

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'), then the sparse variable metric methods, that preserve this pattern, are used. The individual variable metric updates (or families) are specified by using the macrovariable \$UPDATE:

```

$UPDATE='M' - the simple Marwill projection [86]. This update can be used only if $DECOMP='M'.
$UPDATE='G' - the fractioned Marwill projection [126]. This update can be used only if $DE-
               COMP='M' and $NUMBER=3.
$UPDATE='T' - the fractioned Toint projection (the best method given in [126]). This update can be
               used only if $DECOMP='M' and $NUMBER=3.
$UPDATE='B' - the partitioned variable metric updates from the Broyden family [50]. These updates
               can be used only if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.
```

The default value is \$UPDATE='M'.

Fractioned updates with the specifications \$UPDATE='G' or \$UPDATE='T' can be used only in the unconstrained case. If \$UPDATE='B', then the particular update is specified by using the macrovariable \$MET. If \$MET=1, then the BFGS method is used. If \$MET=2, then the DFP method is used. If \$MET=3, then the Hoshino method is used. If \$MET=4, then the safeguarded rank-one method is used. The default value is \$MET=1.

If \$DECOMP='G', then less efficient sparse product form updates from the Broyden family are used. In this case, the values \$MET=1,2,3 can be used.

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

```

L-G-1,  L-M-1,
                                     L-M-3,
G-G-1,
                                     G-M-3,
                                     G-M-5.
```

The default choice is L-M-3. In both the box constrained and the linearly constrained cases we can use only specifications with \$DECOMP='M' and \$NUMBER=3. Similarly, if the fractioned updates (\$UPDATE='T' and \$UPDATE='G') are required, then only specifications with \$DECOMP='M' and \$NUMBER=3 can be used.

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

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

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

The default choice is L-M-3.

3.4. 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 $\$HMODEL$ (or $\$FGHMODEL$) 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 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 dense ($\$HESF='D'$), then all second derivatives have to be given analytically or they are computed numerically by using differences of gradients. The possible specifications (type-decomposition-number) for dense modified Newton methods in the unconstrained case are:

L-G-1, L-S-1, L-B-1,
L-G-2,

L-M-3,
G-G-1, G-S-1, G-B-1,
G-M-3,
G-M-5,
G-M-6.

The default choice is G-M-5. 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 $\$MODEL='FF'$, then only the structurally nonzero second order derivatives have to be given analytically by using the prescribed pattern. 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 some graph coloring problem [16], [17]. If $\$MODEL='AF'$ or $\$MODEL='AQ'$ or $\$MODEL='AP'$, then only the nonzero second derivatives of the approximating functions have to be given analytically by using the prescribed pattern. 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), then the combination [79] of both the modified Newton and the modified Gauss-Newton methods can be used. This choice is possible by using the macrovariable $\$MET$. If $\$MET=1$, then the modified Newton method is used. If $\$MET=2$, then the combined method is used. The default value is $\$MET=2$.

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

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

The default choice is G-M-3. In the box constrained case we can only use specifications with \$DECOMP='M' and \$NUMBER=3. The choice L-M-1 differs from the choice L-M-2. The last one corresponds to the incomplete Gill-Murray decomposition.

If the Hessian matrix is sparse with a partitioned pattern (\$HESF='B'), then 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 more expensive matrix operations.

If \$MODEL='AQ' (sum of squares), then 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. The possible specifications (type-decomposition-number) for partitioned modified Newton methods in the unconstrained case are:

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

The default choice is G-M-3.

3.5. 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='NO') depending on the Hessian matrix specification. Even if 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='NO' - 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 [26] 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], [37] 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 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 [79].

The default value is \$UPDATE='NO'.

Individual variable metric updates from the above families are specified by using the macrovariable \$MET. If \$MET=1, then the BFGS method is used. If \$MET=2, then the DFP method is used. If \$MET=3, then the Hoshino method is used. If \$MET=4, then the original (unsafeguarded) rank-one method is used. The value \$MET=4 is allowed only if \$UPDATE='S' and it 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 it is described in [79]. Decision between these possibilities is mediated by the macrovariable \$MOT1. If \$MOT1=0, then the cumulative update is used. If \$MOT1=1, then the simple update is used.

In the dense case, the modified Gauss-Newton methods can be realized with additional special matrix decompositions that cannot be used in other cases. If \$DECOMP='R', then the recursive QR decomposition [99] is used with an additional correction of the upper triangular matrix R . If \$DECOMP='C' then, moreover, the upper triangular matrix R is changed by using the rank revealing algorithm [15] that can improve its conditioning. The possible specifications (type-decomposition-number) for dense modified Gauss-Newton methods in the unconstrained case are:

L-G-1,	L-S-1,	L-R-1,	L-C-1,	L-M-1,
				L-M-2,
				L-M-3,
G-G-1,	G-S-1,	G-R-1,	G-C-1,	G-M-1,
				G-M-2,
				G-M-3,
				G-M-5,
				G-M-6,
T-G-1,	T-S-1,	T-R-1,	T-C-1,	
T-G-2,				
	T-S-5,		T-C-5,	T-M-5,
				M-M-1,
				M-M-2.

The default choice is G-M-5. 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='NO'). The specification \$UPDATE='S' can be used only if \$DECOMP='M'.

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

- \$UPDATE='NO' - 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 [26] 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 [11] 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 [79].

\$UPDATE='M' - a sparse update based on the Marwill 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 [79].

The default value is \$UPDATE='NO'.

Individual variable metric updates from the above families are specified by using the macrovariable \$MET like the dense case. The value \$MET=4 is allowed only if either \$UPDATE='S' or \$UPDATE='D' and it 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). Decision between these possibilities is mediated by the macrovariable \$MOT1 similarly as in the dense case.

If \$UPDATE='D', then we can use several switches for utilizing variable metric updates specified by the macrovariable \$MOT2. If \$MOT2=0, then the Fletcher and Xu switch [37] is used. If \$MOT2=1, then a modification of the Fletcher and Xu switch is used. If \$MOT2=2, then the Denis and Welsch switch [29] is used. If \$MOT2=3, then the Ramsin and Wedin switch [105] is used. The default value is \$MOT2=0.

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

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

The default choice is G-M-3. In the box constrained case we can use only specifications with \$DECOMP='M' and \$NUMBER=3.

If the Hessian matrix is specified to be sparse with a partitioned pattern (\$HESF='B') then the normal equation matrix has the same structure. If that is the case, then we can use hybrid methods with partitioned updates \$UPDATE='NO', \$UPDATE='S', \$UPDATE='D', \$UPDATE='F', \$UPDATE='B', whose details were already explained above. Note that the partitioned realization is usually less efficient than the general one due to more expensive matrix operations.

The possible specifications (type-decomposition-number) for partitioned Gauss-Newton methods are:

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

The default choice is G-M-3.

If the Hessian matrix is not specified (\$HESF='NO'), then the normal equation matrix is not used. Instead of that the Jacobian matrix, defining a linear least squares problem, is utilized in each iteration. 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 be used only in the unconstrained case.

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

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

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 be used only if NA=NF (system of nonlinear equations).

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

\$UPDATE='NO' - no update is used. The method utilizes original Jacobian matrix.
 \$UPDATE='V' - the simple factorized BFGS update [79] 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 [79] 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', then we can use several switches for utilizing variable metric updates, specified by the macrovariable \$MOT2, like the case with 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 has often a lower number of nonzero elements than the Jacobian one. As a result, the classical Gauss-Newton methods are more efficient in this case.

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

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

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 be used only if NA=NF (system of nonlinear equations). In the box constrained case we can use only specifications with either \$NUMBER=3 or \$NUMBER=4. The choice L-E-1 differs from the choice L-E-2. The last one corresponds to the incomplete LU decomposition.

3.6. 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 in the case when the first derivatives are not specified analytically (the macrovariable \$GMODEL 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='NO'- no update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.
 \$UPDATE='B' - the Broyden family [10] of rank-one updates is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.

When \$UPDATE='B', then the individual quasi-Newton methods are specified by using the macrovariable \$MET. If \$MET=1, then the first (good) Broyden update [10] is used. If \$MET=2, then the second Broyden update [10] is used. If \$MET=3, then the second Greenstadt update [114] is used. If \$MET=4, then the first Greenstadt update [114] is used. If \$MET=5, then the first Todd OC update [60] is used. If \$MET=6, then the first Todd OCX update [60] is used. If \$MET=7, then the second Todd OC update [60] is used. If \$MET=8, then the second Todd OCX update [60] is used. The default value is \$MET=1. Dense quasi-Newton methods can be used only in the unconstrained case.

The possible specifications (type-decomposition-number) for dense quasi-Newton methods are:

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

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

If the Jacobian matrix is sparse with a general pattern (\$JACA='S'), then 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 [18] graph coloring algorithm is used.

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

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

\$UPDATE='NO'- no update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.
 \$UPDATE='B' - the Schubert update [109] is used in almost all iterations. Only after restart, the Jacobian matrix is approximated numerically by using differences.
 \$UPDATE='S' - row scaling update [87] is used in almost all iterations. Only after restart, the Jacobian matrix is approximated numerically by using differences.

The specification \$UPDATE='S' can be used only if the options \$DECOMP='E' and either \$NUMBER=1 or \$NUMBER=2 are chosen.

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

\$UPDATE='NO'- no update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.
 \$UPDATE='B' - the Schubert update [109] is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.
 \$UPDATE='S' - row scaling update [87] is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.

- `$UPDATE='C'` - cyclic column scaling update [64] is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.
- `$UPDATE='M'` - the combination [64] of both the cyclic column scaling and the Schubert updates is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.

The specification `$UPDATE='S'` can be used only if the options `$DECOMP='E'` and either `$NUMBER=1` or `$NUMBER=2` are chosen.

The possible specifications (type-decomposition-number) for sparse quasi-Newton methods are:

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

The default choice is G-A-3 for least squares problems and G-E-3 for systems of nonlinear equations. The specification `$DECOMP='E'` can be used only if `NA=NF` (system of nonlinear equations). In the box constrained case we can use only specifications with either `$NUMBER=3` or `$NUMBER=4`. The choice L-E-1 differs from the choice L-E-2. The later one corresponds to the incomplete LU decomposition.

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

Quasi-Newton methods with limited storage are specified by the statement `$CLASS='QL'`. These methods are special methods for solving sparse systems of nonlinear equations (`$MODEL='AQ'`) in a case in which the first derivatives are not specified analytically (the macrovariable `$GMODEL` is not defined). Therefore, only the case `NA=NF` is permitted. Quasi-Newton methods with limited storage use an initial approximation of the sparse Jacobian matrix together with several small-size matrices which are updated in every iteration in such a way that they approximate the Jacobian matrix as precisely as possible. There are two possibilities which are distinguished by using the macrovariable `$UPDATE`:

- `$UPDATE='NO'` - no update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.
- `$UPDATE='B'` - the Broyden good update of rank-one with limited storage [14] is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.

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

L-A-3, L-E-3.

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`:

- `$UPDATE='NO'` - no update is used. Every approximation of the Jacobian matrix is computed numerically by using differences.
- `$UPDATE='C'` - the inverse column scaling update [88] is used in almost all iterations. Only after restart the Jacobian matrix is approximated numerically by using differences.

The possible specifications (type-decomposition-number) for inverse column scaling methods are:

L-I-1,
L-I-2,
G-I-1.

If `$NUMBER=1`, then a tridiagonal decomposition is used. If `$NUMBER=2`, then an incomplete LU decomposition is used. The default choice is G-I-1.

3.8. Biconjugate direction methods for nonlinear equations

Biconjugate direction methods are specified by the statement `$CLASS='BD'`. These methods are special methods for solving systems of nonlinear equations (`$MODEL='AQ'`) in the case when the first derivatives are not specified analytically (the macrovariable `$GMODEL` is not defined). Therefore only the case `NA=NF` is permitted. Biconjugate direction methods are very efficient for large problems with computationally simple functions in nonlinear equations (`$KCA=1`). The main advantage of biconjugate direction methods is that matrices are not used. This fact highly decreases storage requirements.

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

`$DECOMP='E'` - inexact difference version of the Newton method for systems of nonlinear equations [77]. This method is implemented either as the line search method or as the trust region method and it is based on smoothed CGS algorithm.

Iterative methods for solving linearized equations can be modified by using tridiagonal decomposition. This possibility is determined by the macrovariable `$MOS2`. If `$MOS2=0`, then tridiagonal decomposition is not used. If `$MOS2=1`, then tridiagonal decomposition is used before the iterative process. If `$MOS2=2`, then tridiagonal decomposition is used as a preconditioner. If `$MOS2=3`, then both previous cases are assumed. The default value is `$MOS2=0`.

The possible specifications (type-decomposition-number) for the biconjugate direction methods are:

L-E-3,
G-E-3.

The default choice is G-E-3.

3.9. Modified Brent method for nonlinear equations

The Brent method is specified by the statement `$CLASS='BR'`. This method is a special method for solving dense systems of nonlinear equations (`$MODEL='AQ'`) in the case when the first derivatives are not specified analytically (the macrovariable `$GMODEL` is not defined). Therefore, only the case `NA=NF` is permitted. The Brent method does not need any additional specifications (macrovariables `$TYPE`, `$DECOMP`, `$NUMBER` are not used).

3.10. Methods for linear programming problems

Linear programming methods 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'`), then we can use two different linear programming methods based on the active set strategy:

`$NUMBER=1` - primal reduced gradient (null-space) method (like the method proposed in [42]), which is a special implementation of the steepest descent reduced gradient method.
`$NUMBER=2` - primal projected gradient (range-space) method which is a special implementation of the steepest descent projected gradient method.

The possible specifications (type-number) for dense linear programming methods are L-1 and L-2.

If the constraint Jacobian matrix is sparse ($\$JACC='S'$), then we can use one linear programming method based on the simplex algorithm:

$\$NUMBER=1$ - primal reduced gradient (null-space) method which is described in [125].

The possible specification (type-number) for sparse linear programming methods is L-1.

3.11. Methods for quadratic programming problems

Quadratic programming methods are specified by 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 ($\$JACC='D'$), then 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 [44]) which is a special implementation of the Newton reduced gradient method.

$\$NUMBER=2$ - primal projected gradient (range-space) method (like the method proposed in [33]) which is a special implementation of the Newton projected gradient method.

$\$NUMBER=3$ - dual projected gradient (range-space) method (like the method proposed in [47]).

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

If the constraint Jacobian matrix is sparse ($\$JACC='S'$), then we can use one quadratic programming method based on the simplex algorithm:

$\$NUMBER=1$ - primal reduced gradient (null-space) method which is described in [125].

The possible specification (type-number) for sparse linear programming methods is L-1.

3.12. 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. This subproblem is in fact the same as in 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 [67].

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

Proximal bundle methods are realized only as line search methods in two modifications, which are specified by the macrovariable $\$MEX$. If $\$MEX=0$, then a convex version is assumed. If $\$MEX=1$, then a nonconvex version is assumed and we can define a measure of nonconvexity using the macrovariable $\$ETA5$. The default value is $\$ETA5=0.25$. The possible specifications (type-number) for bundle methods are L-1 and L-2. The default choice is L-1. Proximal bundle methods are used whenever $\$KSF=3$ or $\$KSA=3$. They can be also used for minimax problems as it is shown in Section 3.14.

3.13. 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. This subproblem is in fact the same as in recursive quadratic programming methods for minimax problems. Bundle-Newton 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 [67].
- $\$NUMBER=2$ - primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

A nonconvex version is assumed and we can define a measure of nonconvexity using the macrovariable $\$ETA5$. The default value is $\$ETA5=0.25$. The possible specifications (type-number) for bundle 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 be also used for minimax problems as it is shown in Section 3.14.

3.14. Methods for minimax problems.

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

- $\$CLASS='BM'$ - proximal bundle methods.
- $\$CLASS='BN'$ - bundle-Newton methods.
- $\$CLASS='LP'$ - recursive linear programming methods.
- $\$CLASS='VM'$ - recursive quadratic programming variable metric methods. An approximation of 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.

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 $\$METS=1$ - $\$MET=12$ can be used). Similarly, modified Newton methods are the same as in Section 3.4 with the choice $\$DECOMP='G'$ (the Gill-Murray decomposition is used).

Even if minimax problems can be solved by using bundle methods described in Sections 3.12 - 3.13, it is more efficient to use recursive linear programming or recursive quadratic programming methods that utilize a special structure of minimax problems.

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.

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 [38].

If $\$TYPE='L'$, then The special line search method ($\$MES=5$) described in [68] can be used successfully.

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 [67].
- $\$NUMBER=2$ - primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

All of the above methods are realized only for dense unconstrained or linearly constrained problems. The possible specification (type-number) for recursive linear programming methods is G-1. The possible specifications (type-number) for recursive quadratic programming methods are:

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

The default choice is L-1.

3.15. Recursive quadratic programming methods for nonlinear programming problems.

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

- \$CLASS='VM' - recursive quadratic programming variable metric methods. An approximation of 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 3.4 with the choice \$DECOMP='G' (the Gill-Murray decomposition is used).

Recursive quadratic programming methods for 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 [103]. 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 [47]).
- \$NUMBER=2 - primal projected gradient (range-space) method (like the method proposed in [33]) which is a special implementation of the Newton projected gradient method.

Recursive quadratic programming methods are realized only for dense nonlinear programming problems. The possible specifications (type-number) for these methods are L-1 and L-2. The default choice is L-1.

3.16. Recursive minimax optimization methods for nonlinear programming problems.

Recursive minimax optimization methods for nonlinear programming problems are specified by the statement \$FORM='SM'. These methods belong to two following classes:

- \$CLASS='VM' - recursive minimax optimization variable metric methods. An approximation of Lagrangian function Hessian matrix is updated in each iteration using variable metric updates.
- \$CLASS='MN' - recursive minimax optimization 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, mod-

ified Newton methods are the same as in Section 3.4 with the choice `$DECOMP='G'` (the Gill-Murray decomposition is used).

Recursive minimax optimization methods for nonlinear programming problems are based on the transformation of a nonlinear programming problem to a sequence of minimax problems with l_∞ -exact penalty function (see [68]). These methods are realized as line search methods (`$TYPE='L'`). The special line search method (`$MES=5`) for l_∞ -exact penalty function can be used successfully. The special quadratic programming subproblem, derived from the minimax formulation, can be solved by using two different methods:

- `$NUMBER=1` - dual projected gradient (range-space) method proposed in [67].
- `$NUMBER=2` - primal projected gradient (range-space) method which is a special implementation of the Newton projected gradient method.

Recursive quadratic programming methods are realized only for dense nonlinear programming problems. The possible specifications (type-number) for these methods are L-1 and L-2. The default choice is L-1.

3.17. Inexact recursive quadratic programming methods for large sparse equality constrained nonlinear programming problems.

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

- `$CLASS='CD'` - inexact recursive quadratic programming conjugate direction methods. The Lagrangian function Hessian matrix is represented by several small-size matrices which are updated in each iteration using variable metric updates with limited storage.
- `$CLASS='VM'` - inexact recursive quadratic programming variable metric methods. An approximation of Lagrangian function Hessian matrix is updated in each iteration using variable metric updates.
- `$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'`.

Variable metric methods are realized in two different forms which are specified using the macrovariable `$UPDATE`:

- `$UPDATE='M'` - the simple Marwill projection [86].
- `$UPDATE='B'` - the partitioned variable metric updates from the Broyden family [50]. These updates can be used only if `$MODEL='AF'` or `$MODEL='AQ'` or `$MODEL='AP'`.

The default value is `$UPDATE='B'`, if `$MODEL='AF'` or `$MODEL='AQ'` or `$MODEL='AP'`, and `$UPDATE='M'`, otherwise. If `$UPDATE='B'`, then the particular update is specified by the macrovariable `$MET`. If `$MET=1`, then the BFGS method is used. If `$MET=2`, then the DFP method is used. If `$MET=3`, then the Hoshino method is used. If `$MET=4`, then the safeguarded rank-one method is used. The default value is `$MET=1`.

Inexact recursive quadratic programming methods for equality constrained nonlinear programming problems are based either on an inexact solution of the Karush-Kuhn-Tucker system or on a decomposition of Lagrangian function Hessian matrix followed by an inexact solution of a range space system for the Lagrange multipliers. The first approach, specified by the choice `$DECOMP='K'`, is realized in three variants:

- \$NUMBER=1 - exact sparse Bunch-Parlett decomposition [30] of the indefinite Karush-Kuhn-Tucker system.
- \$NUMBER=3 - inexact smoothed conjugate gradient method for the indefinite Karush-Kuhn-Tucker system with a precision control based on various penalty functions.
- \$NUMBER=4 - inexact MINRES method for the indefinite Karush-Kuhn-Tucker system with a precision control based on various penalty functions.

A particular realization of both inexact smoothed conjugate gradient method and inexact MINRES method depends on specifications given by the macrovariables \$MOS1, \$MOS2, \$MOS3. The macrovariable \$MOS1 specifies a termination criterion for an iterative process. If \$MOS1=1, then a common termination criterion is used. If \$MOS1=2, then a separated termination criterion is used. The macrovariable \$MOS2 specifies a precision control and a choice of penalty parameter. If \$MOS2=0, then a precision control is suppressed. If \$MOS2=1, then a precision control, together with a basic choice of penalty parameter, is used. If \$MOS2=2, then a precision control, together with an extended choice of penalty parameter, based on condition of positive definiteness, is used. The macrovariable \$MOS3 specifies residual smoothing of the conjugate gradient method. If \$MOS3=0, then a residual smoothing is suppressed. If \$MOS3=1, then a one-dimensional residual smoothing is used.

The second approach, specified by the choice \$DECOMP='G', is realized in two variants:

- \$NUMBER=3 - sparse Gill-Murray decomposition of the Lagrangian function Hessian matrix followed by the inexact smoothed conjugate gradient method for positive definite range space system with a precision control based on various penalty functions.
- \$NUMBER=4 - sparse Bunch-Parlett decomposition of the Lagrangian function Hessian matrix followed by inexact MINRES method for an indefinite range space system with a precision control based on various penalty functions.

Individual penalty functions are determined by using the macrovariable \$PENF. If \$PENF=1, then the l_1 exact penalty function is used. If \$PENF=2, then the augmented Lagrangian function with constant Lagrange multipliers is used.

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, \$MEP2, \$MEP3. The macrovariable \$MEP1 specifies a second order correction. If \$MEP1=1, then the second order correction is suppressed. If \$MEP1=2, then the second order correction is determined as being a least squares solution of the shifted constraint system. The macrovariable \$MEP2 specifies estimates of Lagrange multipliers at the beginning of each iteration. If \$MEP2=1, then the initial estimate is taken from the previous iteration. If \$MEP2=2, then the initial estimate is determined as being a least squares solution of the first part of the Karush-Kuhn-Tucker system. The macrovariable \$MEP3 specifies updates of the Lagrange multipliers after a line search. If \$MEP3=1, then a full step is taken. If \$MEP3=2, then a proportional step is taken.

The possible specifications (type-decomposition-number) for inexact recursive quadratic programming methods for equality constrained nonlinear programming problems are

L-K-1,
L-K-3, L-G-3,
L-K-4, L-G-4.

The default choice is L-K-3.

3.18. 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 [52], use a stepsize control based on a local truncation error.

A solution to an initial value problem for ordinary differential equations can be stored for subsequent processing. An extent of stored data is determined by using the macrovariable `$MED`. If `$MED=0`, then no data are stored. If `$MED=1`, then data in all solution steps are stored. If `$MED=2`, then 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.

3.19. 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 `$TYPE` was explained above. Now we will explain the specification `$NUMBER`.

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

`$NUMBER=1` - direct methods for solving linear systems based on various matrix decompositions. The Gill-Murray decomposition [43] is applied if `$DECOMP='M'`. These decompositions are interesting, especially in the sparse case. The Gill-Murray decomposition [43] of the sparse Hessian matrix is used if `$DECOMP='G'`. The orthogonal QR decomposition [127] of the sparse Jacobian matrix is used if `$DECOMP='A'`. The triangular LU decomposition [23] of the sparse Jacobian matrix is used if `$DECOMP='E'`. The Bunch-Parlett decomposition [30] of the sparse Karush-Kuhn-Tucker matrix is used if `$DECOMP='K'`. Moreover, symbolic decomposition is always determined before the iterative process, so that only numerical computations with known factors are carried out in the subsequent iterations.
`$NUMBER=2` - an alternative possibility to the previous case. The Schnabel-Eskow decomposition [108] is applied if `$DECOMP='M'`. In the sparse case, the incomplete Gill-Murray decomposition is used if `$DECOMP='M'` and the incomplete LU decomposition is used if `$DECOMP='E'`
`$NUMBER=3` - inexact iterative methods. The conjugate gradient method [25] for solving linear systems with the Hessian matrix is used if `$DECOMP='M'`. The CGLS method [98] for solving linear systems with the normal equation matrix is used if `$DECOMP='A'`. The smoothed CGS method [123] for solving linear systems with the Jacobian matrix is used if `$DECOMP='E'`. The smoothed conjugate gradient method [83] for a linear system with the Karush-Kuhn-Tucker matrix is used if `$DECOMP='K'`. The precision is specified by the macrovariable `$MOS`. If `$MOS=1`, then the simple strategy is used. If `$MOS=2`, then the geometric decreasing strategy is used. If `$MOS=3`, then the harmonic decreasing strategy is used. If `$DECOMP='M'` and `'S'`, then the conjugate gradient method can be modified by using the incomplete Gill-Murray (IGM) decomposition. Similarly, if `$DECOMP='E'` and `$JACA='S'`, then the smoothed CGS method can be modified by using the incomplete LU (ILU) decomposition. These possibilities are specified by the macrovariable `$MOS2`. If `$MOS2=0` then neither IGM nor ILU decompositions are used. If `$MOS2=1`, then either IGM or ILU decompositions are used before the iterative process. If `$MOS2=2`, then either IGM

or ILU decompositions are used as a preconditioner. If `$MOS2=3`, then both previous cases are assumed.

`$NUMBER=4` - inexact iterative methods. The MINRES method for solving linear systems with the Karush-Kuhn-Tucker matrix if `$DECOMP='K'`. 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

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

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. 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`, then no scaling is performed. If `$MOS1=2`, then the scaling coefficients are derived from the normal equation matrix diagonal elements [75]. There are six possibilities:

- `$NUMBER=1` - so-called dog-leg methods based on various matrix decompositions. The Gill-Murray decomposition [43] is utilized if `$DECOMP='M'`. These decompositions are interesting especially in the sparse case. The Gill-Murray decomposition [43] of the sparse Hessian matrix is used if `$DECOMP='G'`. The orthogonal QR decomposition [127] of the sparse Jacobian matrix is used if `$DECOMP='A'`. The triangular LU decomposition [23] of the sparse Jacobian matrix is used if `$DECOMP='E'`. Moreover, symbolic decomposition is always determined before the iterative process, so that only numerical computations with known factors are carried out in the subsequent iterations. The individual dog-leg methods are specified by the macrovariable `$MOS`. If `$MOS=1`, then the single dog-leg method [100] is used. If `$MOS=2`, then the double dog-leg method [27] is used. If `$MOS=3`, then the triple dog-leg method is used. If `$MOS=4`, then the optimum dog-leg method [13] is used.
- `$NUMBER=2` - an alternative possibility to the previous case. The Schnabel-Eskow decomposition [108] is applied if `$DECOMP='M'`. If `$DECOMP='G'` or `$DECOMP='Q'` or `$DECOMP='E'` then so-called multiple dog-leg methods [78] are supposed. The number of dog-leg steps is specified by the statement `$MOS=number_of_steps` in the last case.
- `$NUMBER=3` - iterative trust region methods. The conjugate gradient trust region method [116] with the Hessian matrix if `$DECOMP='M'`. The CGLS trust region method [74] with the normal equation matrix if `$DECOMP='A'`. The smoothed CGS trust region method [82] with the Jacobian matrix if `$DECOMP='E'`. The precision is specified by the macrovariable `$MOS`. If `$MOS=1`, then the simple strategy is used. If `$MOS=2`, then the geometric decreasing strategy is used. If `$MOS=3`, then the harmonic decreasing strategy is used. If `$DECOMP='M'` and `$HESF='S'`, then the conjugate gradient method can be modified by using the incomplete Gill-Murray (IGM) decomposition. Similarly, if `$DECOMP='E'` and `$JACA='S'`, then the smoothed CGS method can be modified by using the incomplete LU (ILU) decomposition. These possibilities are specified by the macrovariable `$MOS2`. If `$MOS2=0`, then neither IGM nor ILU decompositions are used. If `$MOS2=1`, then either IGM or ILU decompositions are used before the iterative process. If `$MOS2=2`, then either IGM or ILU decompositions are used as a preconditioner. If `$MOS2=3`, then both previous cases are assumed.

- \$NUMBER=4 - iterative trust region methods. The combined Lanczos and CG trust region method [78] with the Hessian matrix if \$DECOMP='M'. The LSQR trust region method [74] with the normal equation matrix if \$DECOMP='A'. The GMRES trust region method [82] with the Jacobian matrix if \$DECOMP='E'. The precision is specified by the macrovariable \$MOS as in the previous case. If \$DECOMP='E' and \$JACA='S', then the GMRES method can be modified by using the incomplete LU (ILU) decomposition. This possibility is specified by the macrovariable \$MOS2 as in the previous case.
- \$NUMBER=5 - an optimum locally constrained trust region method [92]. The Gill-Murray decomposition [43] is utilized if \$DECOMP='M'.
- \$NUMBER=6 - an alternative possibility to the previous case. The Schnabel-Eskow decomposition [108] is applied if \$DECOMP='M'.

If \$TYPE='T', then only the specifications \$NUMBER=1, \$NUMBER=2 and \$NUMBER=5 can be used. These specifications have the same meaning as in the case \$TYPE='G', but the implementation is simpler. If \$NUMBER=5, then the simplified optimum locally constrained trust region method [75] is used.

If \$TYPE='M' then Levenberg-Marquardt type methods are supposed:

- \$NUMBER=1 - a modified Marquardt method proposed by Fletcher [32].
- \$NUMBER=2 - a spiral algorithm proposed by Steen and Byrne [117].

3.20. 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, then only the function values are used during the line search. If \$KDS=1, then 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 then 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, then 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 [21] is used.
- \$MES=5 - conic extrapolation or interpolation [6] is used.

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

- \$KTERS<0 - nonmonotone line search procedure proposed in [51] is used. The absolute value of the macrovariable \$KTERS, which cannot be greater than 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 [48]. The termination precision is given by the value \$EPS1.
- \$KTERS=3 - the Curry-Altman stepsize [20] (Wolfe conditions). The termination precision is given by the values \$EPS1 and \$EPS2.
- \$KTRES=4 - the extended Curry-Altman stepsize [34] (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 an 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 the 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' - 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 [95].

Trust region methods are also influenced using the macrovariable \$KTERS. If \$KTERS<0, then nonmonotone trust region procedure proposed in [24] is used. The absolute value of the macrovariable \$KTERS, which cannot be greater than 10, gives the number of nonmonotone steps.

3.21. Methods for numerical differentiation

The UFO system computes derivatives of the model function (of the approximating functions, of the constraint functions) numerically whenever they are not given analytically. This is made possible by the macroprocessor that 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 using the macrovariable \$MCG:

- \$MCG=0 - a simple difference determination described in [28] is used.
- \$MCG=1 - an optimum difference determination proposed in [45] is used.
- \$MCG=2 - an optimum difference determination proposed in [118] is used.

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

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

If either `$MODEL='DF'` or `$MODEL='DQ'`, then 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 `$SOLVER` as it is described above. If the partial derivatives of all the used functions are given analytically, then the gradient of the objective function is computed by integration methods. There are two possibilities specified by the macrovariable `$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.

3.23. Global optimization methods

Global optimization methods are used if `$EXTREM='G'` is specified. Global optimization methods use local optimization ones for finding local minima. Therefore the particular local optimization method has to be chosen by using the macrovariables `$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 that 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 clustering methods.

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

`$GTYPE=1` - singlestart methods. Random points, uniformly distributed in the given region, are generated and a local minimization method is started from the point with the lowest function value.
`$GTYPE=2` - multistart methods. Random points, uniformly distributed in a given region, are generated and a local minimization is started from every point. Obtained local minima are compared and selected.
`$GTYPE=3` - modified multistart methods. Random points, distributed in a given region uniformly, are generated and a local minimization is started whenever a point is found which has a lower function value than that reached up to date.

\$GTYPE=3 - Bayesian reduced multistart methods [7]. Random samples of points are repeatedly generated. Every random sample is reduced and a 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 [63]. 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 minimization of a tunneling function has failed to find a new starting point.

\$GTYPE=3 - filled function methods [39], [40]. 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 saddle point in a basin of a lower minimizer of the objective function.

If \$GCLASS=3, then 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 less than the threshold distance are linked. A cluster is recognised as a set of points linked together.

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

\$GTYPE=1 - multi level single linkage method [107]. 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 to 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 [107]. This method is a generalization of the mode analysis method. Region is partitioned into cells. After 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 that are described in [107].

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 the use of global optimization methods up to 20 variables only. If we use clustering or multi level single linkage methods (\$GCLASS=3 or \$GCLASS=4), then 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). 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. Output specifications in the UFO system

The UFO system has many output possibilities including the graphical pictures. These output possibilities can be divided into five basic groups.

4.1. Basic screen output

The basic screen output can be used only if `$GRAPH='NO'` and `$DISPLAY='NO'`. In this case, individual rows corresponding to iterations and 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.
`$MOUT=± 2` - Extended output. Additional information from every iteration appear on the screen.
`$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 appear on the screen.

If `$MOUT>0`, then a standard line of final results is printed, while if `$MOUT<0` then a modified line of final results, containing 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.

4.2. Extended screen output

If we want to use an extended screen output, we have to set `$DISPLAY='YES'` (the default value is `$DISPLAY='NO'`). This type of screen output consists of text pages which correspond to individual iterations and final results. 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 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 `$SCAN='YES'` (the default value is `$SCAN='NO'`) If `$SCAN='NO'`, then the output of iterations is suppressed. Scanning of the iterative process can be terminated by using the character `'!'` from the keyboard.

4.3. Graphical screen output

The graphical 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 graphical output, we have to set \$GRAPH='YES' (the default value is \$GRAPH='NO'). In this case, both iterations and final results appear in the graphical mode. 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 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 graphical 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='YES' (the default value is \$SCAN='NO'). Scanning of the iterative process can be terminated by using the character '!' from the keyboard.

Besides text representations in the graphical mode, which are essentially like the ones in the extended screen output (with the choice \$DISPLAY='YES'), we can chose several types of graphical pictures.

a) Graphical picture:

If we have chosen either V (variables) or A (approximation) or C (constraints), then results can be displayed graphically by typing G (graph) from the keyboard. A graphical picture appears on the screen in this case. It contains either values of variables with indices I , $1 \leq I \leq NF$, or values of the approximating functions with indices KA , $1 \leq KA \leq NA$, or values of the constraint functions with indices KC , $1 \leq KC \leq NC$. If we have chosen A (approximation) in the case of $NE > 0$, then the graphical 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 graphical picture can be changed by typing particular characters from the keyboard.

Change of representation:

- V - (values) : Values are drawn.
- O - (ordinates) : Values and ordinates from zero axis are drawn.
- C - (curves) : 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 approximating functions $AF(KA)$, $1 \leq KA \leq NA$, or values of 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 approximating functions $AF(KA)$ together with their prescribed values $AM(KA)$, $1 \leq KA \leq NA$, or values of constraint functions $CF(KC)$ together with their bounds $CL(KC)$ and $CU(KC)$, $1 \leq KC \leq NC$, are demonstrated.

D - (differences) : Either differences between variables and their bounds or differences between approximating functions and their prescribed values or differences between 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.

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.

After typing each character we must use ENTER.

If we have chosen F (function) as a group of final results, we can use additional graphical 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). 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.

After typing each character we must use ENTER.

c) Three dimensional orbit:

If $NE > 2$, we can draw an orbit of three components of a solution of the set of ordinary differential equations by typing P (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). 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 $D\phi$.

T - (tilt) : Tilting rotated values or curves by a subsequently entered angle $D\theta$.

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.

After typing each character we must use ENTER.

d) Colored map of the objective function:

If we have specified either `$MAP='YES'` or `$MAP='EXTENDED'` (default value is `$MAP='NO'`), we can draw a colored 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 colored map.
G - (logarithmic) :	Logarithmic scale of the colored map.
R - (refinement) :	Refinement of the colored map.
B - (back) :	Back refinement of the colored map.
N - (negate) :	Colored 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.
I - (isolines) :	Drawing objective function contours.
W - (new) :	Selection of new variables and drawing a new colored map.
Q - (quit) :	Return to the displayed group of final results.

After typing each character we must use ENTER.

If we set `$MAP='YES'`, then one picture for two variables is drawn. If we set `$MAP='EXTENDED'`, then 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 used variable (according to the text appeared on the screen). Note that the choice `$MAP='EXTENDED'` excludes the choices `$HIL='YES'` and `$ISO='YES'`, so that the other pictures cannot be used.

e) Objective function surface:

If we have specified `$HIL='YES'` (default value is `$HIL='NO'`), 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 - (negate) :	Surface of the objective function negation.

Another type of picture, new surface or return:

M - (map) :	Drawing a colored map of the objective function.
I - (isolines) :	Drawing objective function contours.
W - (new) :	Selection of new variables and drawing new surface.
Q - (quit) :	Return to the displayed group of final results.

After typing each character we must use ENTER.

Before drawing the objective function surface we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every used variable (according to the text appeared on the screen).

f) Objective function contours:

If we have specified `$ISO='YES'` (default value is `$ISO='NO'`), we can draw an objective function contours by typing I (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.

Another type of picture, new contours or return:

M - (map) :	Drawing a colored map of the objective function.
H - (hills) :	Drawing an objective function surface with respect to visibility.
W - (new) :	Selection of new variables and drawing a new surface.
Q - (quit) :	Return to the displayed group of final results.

After typing each character we must use ENTER.

Before drawing the objective function contours we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every used variable (according to the text appeared on the screen).

4.4. 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 to the extent of the print.

The macrovariable `$KOUT` can have the following values:

<code>\$KOUT= 0</code> -	Text file output is suppressed (the file P.OUT is empty)
<code>\$KOUT= ± 1</code> -	Standard output. The heading and the final results are printed together with selected information on each accepted iteration.
<code>\$KOUT= ± 2</code> -	Extended output. Additional information, obtained from stepsize selection, is printed.
<code>\$KOUT= ± 3</code> -	Extended output. Additional information, obtained from direction determination and variable metric update, is printed.
<code>\$KOUT= ± 4</code> -	Extended output. Additional information, obtained from linear constraint addition and deletion, is printed.
<code>\$KOUT= ± 5</code> -	Extended output. Additional information, obtained from numerical differentiation, is printed.

If `$KOUT>0`, then a standard heading is printed, while if `$KOUT<0` then an extended heading, containing problem specifications and optimization options, is printed.

A selection of iterations, accepted for print, is controlled by the contents of the macrovariables `$KOUT1`, `$KOUT2`, `$KOUT3`. If `KOUT1 ≤ KOUT2` then only the iterations whose numbers are between `KOUT1` and `KOUT2` are assumed, but `KOUT3-1` ones are always omitted (`KOUT1` is a lower bound, `KOUT2` is an upper bound and `KOUT3` is a step). Similarly, if `KOUT1 > KOUT2`, then only the iterations whose numbers are less than `KOUT2` or greater than `KOUT1` are assumed, but `KOUT3-1` ones are always omitted. If `$KOUT3=0`, then no iterations are assumed.

While the macrovariable `$KOUT` specifies which information is printed, the macrovariable `$LOUT` specifies how much information is printed:

<code>\$LOUT= 0</code> -	Basic output. The basic information (1 row if <code>\$KOUT=1</code>) is printed in each accepted iteration.
--------------------------	--

\$LOUT= \pm 1 - Extended output. Additional scalars, together with vector of variables, are printed.
 \$LOUT= \pm 2 - Extended output. Additional vectors (usually gradients) are printed.
 \$LOUT= \pm 3 - Extended output. Additional matrices (usually Hessian matrices) are printed.
 \$LOUT= \pm 4 - The most extended output. All useful data are printed.

If \$LOUT>0, then basic part of the information is printed. If \$LOUT<0, then a more extensive part of the information is printed.

The macrovariable \$LOUT has an additional significance. If \$KOUT=0 and \$LOUT>0, then a copy of the basic screen output is provided. If \$KOUT=0 and \$LOUT<0, then 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 : UDDL11
STEP SIZE DETERMINATION : USOLO1
FUNCTION DETERMINATION : UF1FO1
GRADIENT DETERMINATION : UFOGS2
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 contain 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.

4.5. User supplied output

The UFO system allows to the utilization of both the user supplied output subroutines and the post-processing subroutines. These subroutines can be included into the control program by using the macrovariable \$OUTPUT:

```
$SET(OUTPUT)
    Calling the user supplied output subroutines.
    Calling the post-processing subroutines.
$ENDSET
```

Parameters of the user supplied output subroutines and post-processing subroutines must satisfy the UFO conventions. For example, the vector of variables, the model function value, the model function gradient must be denoted X, FF, GF respectively (see chapter 2).

4.6. Tracing in the UFO control program

Tracing in the control program is a useful tool for debugging optimization algorithms on main-frames. If this is the case, then we will specify \$TRACE='YES'. Besides simple tracing, we can prescribe scalar integer or real variables whose values will be printed together with labels. This possibility can be specified by using the macrovariables \$IDEB and \$RDEB:

```
$IDEB = 'list of integer variables separated by commas'
$RDEB = 'list of real variables separated by commas'
```

If the macrovariables \$IDEB or \$RDEB are not specified, then no integer or real variables are printed.

Tracing is executed only in the accepted iterations whose numbers are determined by using the macrovariables \$KOUT1, \$KOUT2, \$KOUT3 (see Section 4.5).

4.7. 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. This possibility serves especially for a debugging and no details are given here.

5. 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.

5.1. 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 \$FMODEL, \$GMODEL, \$HMODEL or \$FMODELA, \$GMODELA, \$HMODELA or \$FMODEL, \$GMODEL, \$HMODEL, respectively. Sometimes checking the correctness of these models is needed. If this is the case, then both the analytical and the numerical differentiation can be compared. Checking optimization problems can be specified by using the macrovariable \$TEST. If \$TEST='NO', then no checking is performed. If \$TEST='YES', then both the analytical and the numerical differentiation is executed before optimization is started and the derivatives obtained are printed. Only the derivatives that are analytically specified (the first, the second) are checked. Finally, if \$TEST='ONLY', then only checking is performed and optimization is not started. An 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.

5.2. Testing optimization methods

The UFO system contains a great number of subroutines (collections of test problems) that serve for testing optimization methods. All of these subroutines begin with the letter 'E' (external). Input subroutines have the second letter 'I' and the third letter 'U' or 'L' or 'N' for an 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, respectively. 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 selected method, we use the specifications \$COLLECTION='YES' 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 (15 dense problems from [19], [70]). External subroutines EIUD01, EFFU01, EFGU01, EFHD01 are used.
- TEST02*.UFO - Tests for sum of squares minimization (30 dense problems from [91]). External subroutines EIUD02, EAFU02, EAGU02, EAHD02 are used.
- TEST03*.UFO - Tests for linearly constrained optimization (16 dense problems from [55]). 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 (7 dense problems from [68]). External subroutines EIUD06, EAFU06, EAGU06, EAHD06 are used.
- TEST07*.UFO - Tests for inequality constrained nonlinear programming (34 dense problems from [55]). External subroutines EIND07, EFFU07, EFGU07, ECFU07, ECGU07 are used.
- TEST08*.UFO - Tests for equality constrained nonlinear programming (31 dense problems from [55]). External subroutines EIND08, EFFU08, EFGU08, ECFU08, ECGU08 are used.
- TEST09*.UFO - Tests for unconstrained global optimization (13 problems from [131]). External subroutines EIUD09, EFFU09, EFGU09 are used.
- TEST10*.UFO - Tests for unconstrained optimization (15 sparse problems from [70], [126]). 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 (8 sparse problems). External subroutines EILS13, EFFU13, EFGU13 are used.
- TEST14*.UFO - Tests for sum of functions minimization (15 sparse problems from [70], [126]). External subroutines EIUB14, EAFU14, EAGU14 are used.
- TEST15*.UFO - Tests for sum of squares minimization (24 sparse problems from [74]). External subroutines EIUB15, EAFU15, EAGU15 are used.
- TEST16*.UFO - Extended tests for unconstrained optimization (25 dense problems from [19], [70], [91]). External subroutines EIUD16, EFFU16, EFGU16, EFHD16 are used.
- TEST17*.UFO - Tests for nonlinear equations solutions (30 dense problems). External subroutines EIUD17, EAFU17, EUGU17 are used.

TEST18*.UFO - Tests for nonlinear equations (20 sparse problems from [77]). External subroutines EIUS18, EAFU18, EAGU18 are used.

TEST19*.UFO - Tests for nonsmooth unconstrained optimization (22 dense problems from [85], [61]). External subroutines EIUD19, EFFU19, EFGU19, EFHD19 are used.

TEST20*.UFO - Tests for equality constrained sparse nonlinear programming (18 sparse problems from [83]). External subroutines EIUB20, EIUS20, EIND20, EINS20, EFFU20, EFGU20, EAFU20, EAGU20, ECFU20, ECGU20 are used.

TEST21*.UFO - Tests for optimization of dynamical systems (4 dense problems). External subroutines EIUD21, EFFU21, EEGU21, EYFU21, EYGU21 are used.

TEST22*.UFO - Tests for linearly constrained minimax optimization (6 dense problems from [68]). External subroutines EIUD22, EAFU22, EAGU22, EAHD22 are used.

TEST32*.UFO - Tests for sum of squares minimization (6 dense problems from [72]). External subroutines EIUD32, EAFU32, EAGU32 are used.

TEST33*.UFO - Tests for sum of squares minimization (6 dense problems from [72]). External subroutines EIUD33, EAFU33, EAGU33 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, . . . , 23 are briefly described in the text files E01.TXT, . . . , E23.TXT.

To demonstrate the use of the test input file we perform a test of sum of squares minimization by using hybrid method realized as a trust region method. The test input file TEST02.UFO has the 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
$SET(GMODELA)
  CALL EAGU02(NF,KA,X,GA,NEXT)
$ENDSET
$NF=12
$NA=400
$KOUT=0
$LOUT=1
$MOUT=1
$MIT=500
$MFV=1000
$MODEL='AQ'
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=5
$UPDATE='F'
$TOLX='1.0$P-16'
$TOLF='1.0$P-16'
$TOLB='1.0$P-16'
$TOLG='1.0$P-6'
$COLLECTION='YES'
$NEXT=25
$BATCH

```

\$STANDARD

The result (screen output) obtained has the following form (each row corresponds to one test problem and the last row is a summary):

1	NIT=	12	NFV=	15	NFG=	13	NDC=	28	F=	.247D-31	G=	.222D-15
2	NIT=	21	NFV=	28	NFG=	22	NDC=	51	F=	.245D+02	G=	.678D-06
3	NIT=	33	NFV=	34	NFG=	34	NDC=	64	F=	.204D-22	G=	.581D-06
4	NIT=	13	NFV=	16	NFG=	14	NDC=	27	F=	.399D-19	G=	.282D-03
5	NIT=	6	NFV=	7	NFG=	7	NDC=	8	F=	.142D-15	G=	.807D-07
6	NIT=	11	NFV=	17	NFG=	12	NDC=	28	F=	.622D+02	G=	.790D-07
7	NIT=	7	NFV=	8	NFG=	8	NDC=	11	F=	.203D-26	G=	.622D-12
8	NIT=	5	NFV=	6	NFG=	6	NDC=	8	F=	.411D-02	G=	.261D-08
9	NIT=	1	NFV=	2	NFG=	2	NDC=	1	F=	.564D-08	G=	.177D-07
10	NIT=	126	NFV=	134	NFG=	127	NDC=	277	F=	.440D+02	G=	.816D-04
11	NIT=	30	NFV=	68	NFG=	30	NDC=	150	F=	.923D-14	G=	.114D+03
USOG01: MAXIMUM NUMBER OF REDUCTIONS												
12	NIT=	12	NFV=	14	NFG=	13	NDC=	29	F=	.114D-20	G=	.601D-10
13	NIT=	10	NFV=	11	NFG=	11	NDC=	14	F=	.169D-09	G=	.247D-06
14	NIT=	41	NFV=	50	NFG=	42	NDC=	82	F=	.142D-23	G=	.280D-10
15	NIT=	11	NFV=	14	NFG=	12	NDC=	29	F=	.154D-03	G=	.511D-06
16	NIT=	23	NFV=	54	NFG=	23	NDC=	102	F=	.429D+05	G=	.530D-04
USOG01: MAXIMUM NUMBER OF REDUCTIONS												
17	NIT=	22	NFV=	24	NFG=	23	NDC=	61	F=	.273D-04	G=	.476D-07
18	NIT=	15	NFV=	20	NFG=	16	NDC=	63	F=	.335D-21	G=	.419D-10
19	NIT=	13	NFV=	15	NFG=	14	NDC=	29	F=	.219D-01	G=	.373D-07
20	NIT=	7	NFV=	8	NFG=	8	NDC=	27	F=	.373D-09	G=	.285D-10
21	NIT=	12	NFV=	15	NFG=	13	NDC=	28	F=	.403D-29	G=	.225D-13
22	NIT=	10	NFV=	11	NFG=	11	NDC=	14	F=	.506D-09	G=	.247D-06
23	NIT=	20	NFV=	25	NFG=	21	NDC=	41	F=	.439D-04	G=	.197D-06
24	NIT=	24	NFV=	34	NFG=	25	NDC=	107	F=	.308D-03	G=	.354D-06
25	NIT=	10	NFV=	11	NFG=	11	NDC=	10	F=	.125D-25	G=	.190D-11
TOTAL	NIT=	495	NFV=	641	NFG=	518	IDC=	1289	*		23	
TIME= 0:00:02.36												

6. Application of the UFO system (examples)

Before the solution of a given problem, the input file containing the problem description and other specifications for 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 a 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 a 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 less than 10000 for a 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, that depend on a compiler, can be incorrect. Instead of the constants 0.0D0, 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 13 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). All input files contain necessary data and can be used in the batch mode. These input files are included to the UFO system as the demo-files PROB01.UFO,...,PROB19.UFO.

6.1. Optimization with simple bounds

a) Problem description:

Suppose we have to find a maximum of the objective function

$$F(x) = \frac{1}{n!} \left(\prod_{i=1}^n x_i \right) - 2$$

with simple bounds $0 \leq x_i \leq i$ for $1 \leq i \leq n$, where $n = 5$. The starting point is $x_i = 2$ for $1 \leq i \leq n$. The solution point is $x_i = i$ for $1 \leq i \leq n$ 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.DO ; XL(I)=0.DO ; XU(I)=DBLE(I) ; IX(I)=3
1 CONTINUE
$ENDSET
$SET(FGMODEL)
  W=1.DO
  DO 2 I=1,NF
    W=W*X(I)/DBLE(I)
2 CONTINUE
  FF=W-2.DO
```

```

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 initial values and simple bounds for variables. By using the macrovariable \$FGMODEL 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):

```

      NIT=    0    NFV=    1    NFG=    1    F=   .187D+01    G=   .667D-01
      NIT=    1    NFV=    4    NFG=    4    F=   .155D+01    G=   .150D+00
      NIT=    2    NFV=    7    NFG=    7    F=   .120D+01    G=   .200D+00
      NIT=    3    NFV=    9    NFG=    9    F=   .100D+01    G=   .000D+00
0  NIT=    3    NFV=    9    NFG=    9    NDC=    0    F=   .100D+01    G=   .000D+00
FF =-.1000000000D+01
X  = .1000000000D+01 .2000000000D+01 .3000000000D+01 .4000000000D+01
      .5000000000D+01

```

6.2. Minimization of the sum of squares

a) problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \frac{1}{2} \sum_{i=1}^m (x_4 e^{-x_1 t_i} + x_5 e^{-x_2 t_i} + x_6 e^{-x_3 t_i} - y_i)^2$$

where $m = 20$, $t_i = i/10$ and $y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}$ for $1 \leq i \leq m$. 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.DO ; X(2)=2.DO ; X(3)=1.DO
X(4)=1.DO ; X(5)=1.DO ; X(6)=1.DO
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(FMODEL)
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
$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 \$FGMODEL 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):

NIT=	0	NFV=	7	NFG=	0	F=	.465D+00	G=	.675D+00
NIT=	1	NFV=	14	NFG=	0	F=	.264D+00	G=	.323D+00
NIT=	2	NFV=	21	NFG=	0	F=	.142D+00	G=	.433D-01
NIT=	3	NFV=	28	NFG=	0	F=	.115D+00	G=	.628D-01
NIT=	4	NFV=	35	NFG=	0	F=	.734D-01	G=	.780D-01
NIT=	5	NFV=	43	NFG=	0	F=	.492D-01	G=	.397D-01
NIT=	6	NFV=	50	NFG=	0	F=	.383D-01	G=	.140D+00
NIT=	7	NFV=	57	NFG=	0	F=	.326D-01	G=	.291D+00
NIT=	8	NFV=	65	NFG=	0	F=	.170D-01	G=	.114D-01
NIT=	9	NFV=	72	NFG=	0	F=	.128D-01	G=	.106D-01
NIT=	10	NFV=	79	NFG=	0	F=	.639D-02	G=	.163D-01
NIT=	11	NFV=	86	NFG=	0	F=	.960D-03	G=	.387D-01
NIT=	12	NFV=	93	NFG=	0	F=	.947D-03	G=	.781D-01
NIT=	13	NFV=	100	NFG=	0	F=	.136D-04	G=	.956D-02
NIT=	14	NFV=	107	NFG=	0	F=	.414D-11	G=	.565D-05
NIT=	15	NFV=	114	NFG=	0	F=	.531D-23	G=	.485D-11

```

0 NIT= 15 NRV= 114 NFG= 0 NDC= 65 F= .531D-23 G= .485D-11
F = .5306095250D-23
X = .1000000000D+01 .1000000000D+02 .4000000000D+01 .1000000000D+01
.5000000000D+01 .3000000000D+01

```

6.3. Minimax approximation

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \max_{1 \leq i \leq m} \left| \frac{x_1 + t_i x_2}{1 + t_i x_3 + t_i^2 x_4 + t_i^3 x_5} - y_i \right|$$

where $m = 21$, $t_i = (i - 1)/10 - 1$ and $y_i = e^{-t_i}$ for $1 \leq i \leq m$. 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(FMODEL)
  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 \$FMODEL 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):

```

NIT=    0   NRV=    6   NFG=    0   F= .222D+01   G= .100D+61
NIT=    1   NRV=   13   NFG=    0   F= .203D+01   G= .783D+00

```



```

      NIT=   2   NFV=   20   NFG=   0   F=   .192D+01   G=   .571D+00
      NIT=   3   NFV=   27   NFG=   0   F=   .121D+01   G=   .520D+00
      NIT=   4   NFV=   33   NFG=   0   F=   .274D+00   G=   .333D+00
      NIT=   5   NFV=   39   NFG=   0   F=   .115D-01   G=   .615D-01
      NIT=   6   NFV=   45   NFG=   0   F=   .367D-02   G=   .207D-01
      NIT=   7   NFV=   51   NFG=   0   F=   .531D-03   G=   .172D-01
      NIT=   8   NFV=   57   NFG=   0   F=   .179D-03   G=   .439D-02
      NIT=   9   NFV=   63   NFG=   0   F=   .123D-03   G=   .120D-03
      NIT=  10   NFV=   69   NFG=   0   F=   .122D-03   G=   .913D-07
0  NIT=  10   NFV=   69   NFG=   0   NDC=   0   F=   .122D-03   G=   .913D-07
F = .1223712519D-03
X = .9998776287D+00 .2535884404D+00 -.7466075718D+00 .2452015019D+00
   -.3749029102D-01

```

6.4. Nonsmooth optimization

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = -x_1 + 2 * (x_1^2 + x_2^2 - 1) + \frac{7}{4}|x_1^2 + x_2^2 - 1|$$

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.DO
  X(2)=-1.DO
$ENDSET
$SET(FGMOLELF)
  W=X(1)**2+X(2)**2-1.DO
  FF=-X(1)+2.DO*W+1.75DO*ABS(W)
  W=SIGN(3.5$P 0,W)+4.DO
  GF(1)=W*X(1)-1.DO
  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 \$FGMODEL 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):

```

NIT=    0    NFV=    1    NFG=    1    F=   .475D+01    G=   .100D+61
NIT=    1    NFV=    3    NFG=    3    F=  -.379D+00    G=   .850D+01
NIT=    2    NFV=    4    NFG=    4    F=  -.606D+00    G=   .933D+00
NIT=    3    NFV=    5    NFG=    5    F=   .925D+01    G=   .802D+00
NIT=    4    NFV=    6    NFG=    6    F=  -.728D+00    G=   .802D+00
NIT=    5    NFV=    7    NFG=    7    F=   .871D+00    G=   .348D+00
NIT=    6    NFV=    8    NFG=    8    F=  -.828D+00    G=   .722D+00
NIT=    7    NFV=    9    NFG=    9    F=  -.844D+00    G=   .162D+00
NIT=    8    NFV=   10    NFG=   10    F=  -.999D+00    G=   .984D-01
NIT=    9    NFV=   11    NFG=   11    F=  -.998D+00    G=   .114D+00
NIT=   10    NFV=   12    NFG=   12    F=  -.999D+00    G=   .501D+00
NIT=   11    NFV=   13    NFG=   13    F=  -.100D+01    G=   .530D-01
NIT=   12    NFV=   14    NFG=   14    F=  -.100D+01    G=   .454D-05
NIT=   13    NFV=   15    NFG=   15    F=  -.100D+01    G=   .986D-07
0 NIT=  13    NFV=  15    NFG=  15    NDC=   0    F=  -.100D+01    G=   .268D-08
FF =-.1000000000D+01
X   = .1000000000D+01  .6971317425D-18

```

6.5. Optimization with linear constraints

a) problem specification:

Suppose we have to find a 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.DO ; CG(7)=0.DO ; CG(8)=1.DO
CG(9)=0.DO ; CG(10)=5.DO
$ENDSET
$SET(FMODEL)
FF=(X(1)-X(2))**2+(X(3)-1.DO)**2+ &
(X(4)-1.DO)**4+(X(5)-1.DO)**6
$ENDSET
$SET(GMODEL)
GF(1)= 2.DO*(X(1)-X(2))
GF(2)=-2.DO*(X(1)-X(2))
GF(3)= 2.DO*(X(3)-1.DO)
GF(4)= 4.DO*(X(4)-1.DO)**3
GF(5)= 6.DO*(X(5)-1.DO)**5
$ENDSET
$NF=5
$NC=2
$NCL=2
$KBC=1
$MOUT=2
$NOUT=1
$FMIN=0
$BATCH
$STANDARD

```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of variables and types and values of the general linear constraints. Since there are only the 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 less than zero. By using the macrovariable \$FMODEL we specify analytically the value of the model function. By using the macrovariable \$GMODEL we specify analytically the gradient of the model function.

d) Problem solution (basic screen output):

NIT=	0	NFV=	1	NFG=	1	F=	.266D+03	G=	.853D+02
NIT=	1	NFV=	2	NFG=	2	F=	.234D+02	G=	.911D+01
NIT=	2	NFV=	3	NFG=	3	F=	.900D+01	G=	.571D+01
NIT=	3	NFV=	4	NFG=	4	F=	.564D+00	G=	.100D+01
NIT=	4	NFV=	5	NFG=	5	F=	.270D+00	G=	.435D+00
NIT=	5	NFV=	6	NFG=	6	F=	.160D+00	G=	.300D+00
NIT=	6	NFV=	7	NFG=	7	F=	.322D-01	G=	.101D+00
NIT=	7	NFV=	8	NFG=	8	F=	.101D-01	G=	.422D-01
NIT=	8	NFV=	9	NFG=	9	F=	.323D-02	G=	.521D-01
NIT=	9	NFV=	10	NFG=	10	F=	.293D-02	G=	.957D-01
NIT=	10	NFV=	11	NFG=	11	F=	.665D-03	G=	.264D-01
NIT=	11	NFV=	12	NFG=	12	F=	.439D-03	G=	.201D-01
NIT=	12	NFV=	13	NFG=	13	F=	.242D-03	G=	.554D-02
NIT=	13	NFV=	14	NFG=	14	F=	.465D-04	G=	.101D-01
NIT=	14	NFV=	15	NFG=	15	F=	.298D-04	G=	.112D-01

```

NIT= 15   NFV= 16   NFG= 16   F= .104D-04   G= .266D-02
NIT= 16   NFV= 17   NFG= 17   F= .591D-05   G= .113D-02
NIT= 17   NFV= 18   NFG= 18   F= .258D-05   G= .103D-02
NIT= 18   NFV= 19   NFG= 19   F= .803D-06   G= .994D-03
NIT= 19   NFV= 20   NFG= 20   F= .198D-06   G= .510D-03
NIT= 20   NFV= 21   NFG= 21   F= .768D-07   G= .249D-03
NIT= 21   NFV= 22   NFG= 22   F= .480D-07   G= .451D-03
NIT= 22   NFV= 23   NFG= 23   F= .108D-07   G= .622D-04
NIT= 23   NFV= 24   NFG= 24   F= .488D-08   G= .143D-04
NIT= 24   NFV= 25   NFG= 25   F= .113D-08   G= .200D-04
NIT= 25   NFV= 26   NFG= 26   F= .377D-09   G= .209D-04
NIT= 26   NFV= 27   NFG= 27   F= .969D-10   G= .199D-05
NIT= 27   NFV= 28   NFG= 28   F= .293D-10   G= .300D-05
NIT= 28   NFV= 30   NFG= 30   F= .166D-10   G= .297D-05
NIT= 29   NFV= 31   NFG= 31   F= .430D-11   G= .166D-05
NIT= 30   NFV= 32   NFG= 32   F= .193D-11   G= .262D-05
NIT= 31   NFV= 33   NFG= 33   F= .499D-12   G= .260D-06
0 NIT= 31   NFV= 33   NFG= 33   NDC= 0   F= .499D-12   G= .260D-06
FF = .4994938390D-12
X = .1001660928D+01 .1001660854D+01 .9999998639D+00 .9991695885D+00
.1000000027D+01

```

6.6. Minimax approximation with linear constraints

a) Problem description:

Suppose we have to find a 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

$$\frac{5}{100} x_1 - x_2 + \frac{1}{2} \geq 0.$$

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.DO)-1.DO
  IF (KA.EQ.3) FA=-LOG(X(2))-1.DO
$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 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):

```

      MIT=   0   NFV=   3   NFG=   0   F=   .361D+01   G=   .100D+61
      MIT=   1   NFV=   6   NFG=   0   F=   .198D+01   G=   .363D+00
      MIT=   2   NFV=   9   NFG=   0   F=   .682D+00   G=   .487D+00
      MIT=   3   NFV=  12   NFG=   0   F=  -.277D+00   G=   .595D+00
      MIT=   4   NFV=  15   NFG=   0   F=  -.368D+00   G=   .371D+00
      MIT=   5   NFV=  18   NFG=   0   F=  -.375D+00   G=   .161D+00
      MIT=   6   NFV=  21   NFG=   0   F=  -.448D+00   G=   .143D+00
      MIT=   7   NFV=  24   NFG=   0   F=  -.449D+00   G=   .184D-01
      MIT=   8   NFV=  27   NFG=   0   F=  -.449D+00   G=   .285D-05
      MIT=   9   NFV=  30   NFG=   0   F=  -.449D+00   G=   .128D-10
0  MIT=   9   NFV=  30   NFG=   0   NDC=   0   F=  -.449D+00   G=   .128D-10
F  =-.4489107861D+00
X   = .1526434615D+01   .5763217307D+00

```

6.7. Optimization with nonlinear constraints (nonlinear programming)

a) Problem description:

Suppose we have to find a 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

$$(x_4 - x_6)^2 + (x_5 - x_7)^2 \geq 4$$

$$\frac{x_3 x_4 - x_2 x_5}{\sqrt{x_2^2 + x_3^2}} \geq 1$$

$$\frac{x_3 x_6 - x_2 x_7}{\sqrt{x_2^2 + x_3^2}} \geq 1$$

$$\frac{x_1 x_3 + (x_2 - x_1) x_5 - x_3 x_4}{\sqrt{(x_2 - x_1)^2 + x_3^2}} \geq 1$$

$$\frac{x_1 x_3 + (x_2 - x_1) x_7 - x_3 x_6}{\sqrt{(x_2 - x_1)^2 + x_3^2}} \geq 1$$

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(FMODEL F)
  FF=X(1)*X(3)
$ENDSET
$SET(FMODEL C)
  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 initial values and simple bounds for variables and 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 \$FMODEL we specify analytically the value of the model function. The gradient of the model function is computed numerically.

d) Problem solution (basic screen output):

```

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= .332D+02 C= .961D+00 G= .267D+01
NIC= 0 NIT= 2 NFV= 31 NFG= 0 F= .290D+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= .321D-12 G= .177D-05
0 NIC= 0 NIT= 13 NFV= 158 NFG= 0 F= .233D+02 C= .321D-12 G= .792D-07
FF = .2331370850D+02
X = .4828427089D+01 .3838703303D-07 .4828427160D+01 .100000019D+01
    .2414213578D+01 .2414213565D+01 .1000000000D+01

```

6.8. Global optimization

a) Problem description:

Suppose we have to find a global minimum of the objective function

$$F(x) = (x_1 - 3)^2(x_1 + 5)^2 + (x_2 - 2)^2(x_2 + 3)^2 - x_1^2x_2^2$$

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.DO ; XU(1)=10.DO
  XL(2)=-12.DO ; XU(2)=10.DO
$ENDSET
$SET(FMODEL)
  FF=((X(1)-3.DO)*(X(1)+5.DO))**2+ &
    ((X(2)-2.DO)*(X(2)+3.DO))**2-(X(1)*X(2))**2
$ENDSET
$NF=2
$MOUT=1
$EXTREM='G'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify bounds defining the investigated region. By using the macrovariable \$FMODEL we specify analytically the value of the model function. The gradient of the model function is computed numerically. Since we require to find a global minimum we set \$EXTREM='G'.

d) Problem solution (basic screen output):

```
0  NIT= 10  NFV= 39  NDC= 0  F= -.806D+03
0  NIT= 8   NFV= 29  NDC= 0  F= -.307D+03
0  NIT= 7   NFV= 27  NDC= 0  F= -.150D+03
0  NIT= 11  NFV= 44  NDC= 0  F= -.806D+03
0  NIT= 8   NFV= 30  NDC= 0  F= -.580D+02
0  NIT= 13  NFV= 55  NDC= 0  F= -.307D+03

1.EXTREM : F =  -.8060772623D+03
           X =  -.7329989930D+01  -.6447506505D+01

2.EXTREM : F =  -.3072281498D+03
           X =  -.6228926461D+01  .4363683078D+01

3.EXTREM : F =  -.1504539067D+03
           X =  .3836710475D+01  -.4317610487D+01

4.EXTREM : F =  -.5795091449D+02
```



```
X = .3368245214D+01 .2827173224D+01
```

```
STATISTICS:
```

```
NEXTREM = 4 NFVT = 660 NFGT = 0 NFHT = 0
```

6.9. Large scale optimization (sparse Hessian matrix)

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \sum_{i=1}^n ((3 - 2x_i)x_i - x_{i-1} - x_{i+1} + 1)^2, \quad x_{n+1} = x_0 = 0$$

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(FMODEL)
  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(GMODEL)
  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=500
$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 \$FMODEL we specify analytically the value of the model function. By using the macrovariable \$GMODEL we specify analytically the gradient of the model function.

d) problem solution (basic screen output):

```

      NIT=   0   NFV=   1   NFG=   4   F=  .410D+03   G=  .380D+02
      NIT=   1   NFV=   2   NFG=   8   F=  .513D+02   G=  .123D+02
      NIT=   2   NFV=   3   NFG=  12   F=  .616D+01   G=  .694D+01
      NIT=   3   NFV=   4   NFG=  16   F=  .218D+01   G=  .414D+01
      NIT=   4   NFV=   5   NFG=  20   F=  .883D+00   G=  .281D+01
      NIT=   5   NFV=   6   NFG=  24   F=  .375D+00   G=  .160D+01
      NIT=   6   NFV=   7   NFG=  28   F=  .165D+00   G=  .108D+01
      NIT=   7   NFV=   8   NFG=  32   F=  .736D-01   G=  .706D+00
      NIT=   8   NFV=   9   NFG=  36   F=  .200D-03   G=  .653D-01
      NIT=   9   NFV=  10   NFG=  40   F=  .196D-04   G=  .818D-02
      NIT=  10   NFV=  11   NFG=  44   F=  .137D-06   G=  .174D-02
      NIT=  11   NFV=  12   NFG=  48   F=  .434D-09   G=  .737D-04
      NIT=  12   NFV=  13   NFG=  52   F=  .163D-11   G=  .451D-05
      NIT=  13   NFV=  14   NFG=  56   F=  .990D-14   G=  .361D-06
0  NIT=  13  NFV=  14  NFG=  56  NDC=   0  F=  .990D-14  G=  .361D-06

```

6.10. Large-scale optimization (sparse Jacobian matrix)

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \sum_{i=1}^n f_i^A(x)$$

where $n=100$ and

$$\begin{aligned}
 f_i^A(x) &= ((3 - 2x_i)x_i - x_{i+1} + 1)^2 & , i = 1 \\
 f_i^A(x) &= ((3 - 2x_i)x_i - x_{i-1} - x_{i+1} + 1)^2 & , 2 \leq i \leq n - 1 \\
 f_i^A(x) &= ((3 - 2x_i)x_i - x_{i-1} + 1)^2 & , i = n
 \end{aligned}$$

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):

```

$FLOAT A
$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(FMODEL)
  A=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0
  IF (KA.GT.1) A=A-X(KA-1)
  IF (KA.LT.NF) A=A-X(KA+1)
  FA=A*A
$ENDSET
$SET(GMODEL)
  A=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0
  IF (KA.GT.1) A=A-X(KA-1)
  IF (KA.LT.NF) A=A-X(KA+1)
  A=A+A
  GA(KA)=A*(3.0D0-4.0D0*X(KA))
  IF (KA.GT.1) GA(KA-1)=-A
  IF (KA.LT.NF) GA(KA+1)=-A
$ENDSET
$NF=100
$NA=100
$MA=300
$M=600
$MOUT=2
$MODEL='AF'
$JACA='S'
$HESF='B'
$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):

```

NIT=    0   NFV=    1   NFG=    1   F=   .410D+03   G=   .380D+02
NIT=    1   NFV=    2   NFG=    2   F=   .562D+02   G=   .887D+01
NIT=    2   NFV=    3   NFG=    3   F=   .335D+02   G=   .597D+01
NIT=    3   NFV=    5   NFG=    5   F=   .821D+01   G=   .709D+01
NIT=    4   NFV=    7   NFG=    7   F=   .182D+01   G=   .696D+01
NIT=    5   NFV=    8   NFG=    8   F=   .666D+00   G=   .239D+01
NIT=    6   NFV=    9   NFG=    9   F=   .431D-01   G=   .131D+01
NIT=    7   NFV=   10   NFG=   10   F=   .169D-02   G=   .176D+00
NIT=    8   NFV=   11   NFG=   11   F=   .736D-04   G=   .335D-01
NIT=    9   NFV=   12   NFG=   12   F=   .655D-05   G=   .116D-01
NIT=   10   NFV=   13   NFG=   13   F=   .585D-07   G=   .882D-03
NIT=   11   NFV=   14   NFG=   14   F=   .536D-08   G=   .487D-03
NIT=   12   NFV=   15   NFG=   15   F=   .540D-10   G=   .272D-04
NIT=   13   NFV=   16   NFG=   16   F=   .297D-11   G=   .695D-05
NIT=   14   NFV=   17   NFG=   17   F=   .832D-13   G=   .123D-05
NIT=   15   NFV=   18   NFG=   18   F=   .982D-15   G=   .104D-06
0 NIT=  15   NFV=  18   NFG=  18   NDC=    0   F=   .982D-15   G=   .104D-06

```

6.11. Large-scale sum of squares optimization (sparse Jacobian matrix)

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \sum_{i=1}^n (f_i^A(x))^2$$

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
$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):

```

      NIT=    0   NFV=    1   NFG=    1   F=   .205D+03   G=   .190D+02
      NIT=    1   NFV=    2   NFG=    2   F=   .636D+01   G=   .224D+01
      NIT=    2   NFV=    3   NFG=    3   F=   .233D-01   G=   .145D+00
      NIT=    3   NFV=    4   NFG=    4   F=   .186D-05   G=   .445D-02
      NIT=    4   NFV=    5   NFG=    5   F=   .107D-10   G=   .937D-05
      NIT=    5   NFV=    6   NFG=    6   F=   .138D-18   G=   .871D-09
0  NIT=    5   NFV=    6   NFG=    6   NDC=    0   F=   .138D-18   G=   .871D-09

```

6.12. Large-scale nonlinear equations

a) Problem description:

Suppose we have to solve a system of the nonlinear equations

$$\begin{aligned}
 f_i^A(x) &= (3 - 2x_i)x_i - x_{i+1} + 1 = 0 & , i = 1 \\
 f_i^A(x) &= (3 - 2x_i)x_i - x_{i-1} - x_{i+1} + 1 = 0 & , 2 \leq i \leq n - 1 \\
 f_i^A(x) &= (3 - 2x_i)x_i - x_{i-1} + 1 = 0 & , i = n
 \end{aligned}$$

where $n=100$. The starting point is $x_i = -1$ for $1 \leq i \leq n$. 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
$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='AQ'
$JACA='NO'
$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):

```

      NIT=    0    NFV=    1    F=   .205D+03
      NIT=    1    NFV=    5    F=   .527D+01
      NIT=    2    NFV=    9    F=   .167D-01
      NIT=    3    NFV=   13    F=   .152D-05
      NIT=    4    NFV=   19    F=   .956D-11
      NIT=    5    NFV=   27    F=   .389D-18
0  NIT=    5    NFV=   27    NDC=    0    F=   .389D-18

```

6.13. Large-scale linear programming

a) Problem description:

Suppose we have to find a 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

$$-x_i + x_{i+1} - x_{i+2} = i, \quad 1 \leq i \leq n_C$$

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)
    CALL UKMCI1(KC,KC,-1.0D0,ICG,JCG,CG)
    CALL UKMCI1(KC,KC+1,1.0D0,ICG,JCG,CG)
    CALL UKMCI1(KC,KC+2,-1.0D0,ICG,JCG,CG)
2 CONTINUE
$ENDSET
$IEXT=1
$NF=20

```

```

$NC=18
$NCL=18
$MC=200
$KBF=2
$KBC=1
$MOUT=2
$NOOUT=1
$MODEL='FL'
$JACC='S'
$BATCH
$STANDARD

```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify 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 $3*(NF-2)=54$. We set \$MC=200 as a sufficiently large dimension for auxiliary fields. The option \$MODEL='FL' indicates the linear programming problem.

d) Problem solution (basic screen output):

```

NUMITR= 1 IJNEW= 20 IJOLD= 15 KINP= 0 IU= 48 F= .980D+04
NUMITR= 2 IJNEW= 19 IJOLD= 20 KINP= 0 IU= 49 F= .208D+04
NUMITR= 3 IJNEW= 0 IJOLD= 20 KINP= 0 IU= 49 F= .000D+00
0 NUMITR= 3 NEL= 3 NREF= 1 KINP= 0 IU= 49 F= .000D+00 ITERL= 1
NUMITR= 1 IJNEW= 15 IJOLD= 19 KINP= 0 IU= 49 F=-.900D+01
NUMITR= 2 IJNEW= 20 IJOLD= 18 KINP= 0 IU= 48 F=-.700D+01
NUMITR= 3 IJNEW= 0 IJOLD= 18 KINP= 0 IU= 48 F=-.700D+01
0 NUMITR= 3 NEL= 3 NREF= 1 KINP= 0 IU= 48 F=-.700D+01 ITERL= 2
0 NIT= 0 NFV= 0 NFG= 0 NDC= 200 F= .700D+01 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

```

6.14. Large-scale quadratic programming

a) Problem description:

Suppose we have to find a minimum of the quadratic function

$$F(x) = \sum_{i=1}^{k-2} (x_{k+i+1} - x_{k+i})^2$$

with simple bounds $\alpha_i \leq x_i \leq \alpha_{i+1}$, $0.4(\alpha_{i+2} - \alpha_i) \leq x_{k+i} \leq 0.6(\alpha_{i+2} - \alpha_i)$, $1 \leq i \leq k-1$, $\alpha_k \leq x_k \leq \alpha_{k+1}$, and linear constraints

$$x_{k+i} - x_{i+1} + x_i = 0, 1 \leq i \leq k - 1$$

where $\alpha_i = 1 + (101/100)^i$, $1 \leq i \leq k + 1$, and where $n = 2k - 1 = 41$, $n_C = k - 1 = 20$. The starting point is not given. The minimum value of the quadratic objective function is $F = 2.29133$.

b) Problem specification (input file):

```

$FLOAT WA,WB,WC
$SET(INPUT)
  WA=1.00D0; WB=2.01D0
  DO 1 I=1,NC
    J=I+NC+1
    WC=1.0D0+(1.01D0)**(I+1)
    IX(I)=3; XL(I)=WA; XU(I)=WB
    IX(J)=3; XL(J)=0.4D0*(WC-WA); XU(J)=0.6D0*(WC-WA)
    GF(I)=0.0D0
    GF(J)=0.0D0
    WA=WB; WB=WC
    IC(I)=5; CL(I)=0.0D0
    CALL UKMCI1(I,J,1.0D0,ICG,JCG,CG)
    CALL UKMCI1(I,I,1.0D0,ICG,JCG,CG)
    CALL UKMCI1(I,I+1,-1.0D0,ICG,JCG,CG)
    IH(I)=1
1 CONTINUE
  IX(NC+1)=3; XL(NC+1)=WA; XU(NC+1)=WB
  GF(NC+1)=1
  IH(NC+1)=1; IH(NC+2)=1
  K=NC+2
  DO 2 I=K,NF-1
    IH(I+1)=IH(I)+2
2 CONTINUE
  IH(NF+1)=IH(NF)+1
  J=1
  DO 3 I=K,NF
    JH(J)=I; JH(J+1)=I+1
    HF(J)=2.0D0; HF(J+1)=-2.0D0
    IF (I.EQ.K.OR.I.EQ.NF) HF(J)=1.0D0
    J=J+2
3 CONTINUE
$ENDSET
$NF=41
$NC=20
$NCL=20
$MC=500
$M=100
$KBF=2
$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 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, indicated by the statement \$HESF='S', is very simple and the number of its upper half nonzero elements is $2*(N-NC)-3=39$. We set $M=100$ as a sufficiently large dimension for working fields. The sparse Jacobian matrix, indicated by the statement \$JACC='S', is tridiagonal and the number of its nonzero elements is $3*NC=60$. We set $MC=200$ as a sufficiently large dimension for working fields. The option \$MODEL='FQ' indicates the linear programming problem.

d) Problem solution (basic screen output):

```

NUMITR= 1 IJNEW= 33 IJOLD= 33 KINP= 0 IU= 39 F= .496D+00
NUMITR= 2 IJNEW= 36 IJOLD= 36 KINP= 0 IU= 39 F= .393D+00
NUMITR= 3 IJNEW= 35 IJOLD= 35 KINP= 0 IU= 39 F= .261D+00
NUMITR= 4 IJNEW= 39 IJOLD= 39 KINP= 0 IU= 39 F= .164D+00
NUMITR= 5 IJNEW= 31 IJOLD= 31 KINP= 0 IU= 39 F= .905D-01
NUMITR= 6 IJNEW= 26 IJOLD= 26 KINP= 0 IU= 39 F= .416D-01
NUMITR= 7 IJNEW= 37 IJOLD= 10 KINP= 0 IU= 39 F= .123D-01
NUMITR= 8 IJNEW= 28 IJOLD= 28 KINP= 0 IU= 39 F= .435D-02
NUMITR= 9 IJNEW= 0 IJOLD= 28 KINP= 0 IU= 39 F= .000D+00
  0 NUMITR= 9 NEL= 6 NREF= 1 KINP= 0 IU= 39 F= .000D+00 ITERL= 1
  NAQ= 1 NIQ= 0 NSBSP= 14 NCGR= 0 IU= 39 F= .2291583D+01 G= .686D-01
  NAQ= 1 NIQ= 1 NSBSP= 13 NCGR= 3 IU= 39 F= .2291563D+01 G= .683D-01
  NAQ= 1 NIQ= 2 NSBSP= 12 NCGR= 6 IU= 39 F= .2291553D+01 G= .668D-01
  NAQ= 1 NIQ= 3 NSBSP= 11 NCGR= 10 IU= 39 F= .2291537D+01 G= .668D-01
  NAQ= 1 NIQ= 4 NSBSP= 10 NCGR= 14 IU= 39 F= .2291512D+01 G= .455D-01
  NAQ= 1 NIQ= 5 NSBSP= 9 NCGR= 19 IU= 38 F= .2291501D+01 G= .291D-01
  NAQ= 2 NIQ= 0 NSBSP= 14 NCGR= 19 IU= 37 F= .2291496D+01 G= .436D-01
  NAQ= 2 NIQ= 1 NSBSP= 13 NCGR= 24 IU= 37 F= .2291495D+01 G= .435D-01
  NAQ= 2 NIQ= 2 NSBSP= 12 NCGR= 30 IU= 37 F= .2291495D+01 G= .432D-01
  NAQ= 2 NIQ= 3 NSBSP= 11 NCGR= 36 IU= 37 F= .2291495D+01 G= .429D-01
  NAQ= 2 NIQ= 4 NSBSP= 10 NCGR= 41 IU= 36 F= .2291482D+01 G= .434D-01
  NAQ= 2 NIQ= 5 NSBSP= 9 NCGR= 47 IU= 38 F= .2291360D+01 G= .125D-01
  NAQ= 3 NIQ= 0 NSBSP= 10 NCGR= 47 IU= 38 F= .2291359D+01 G= .131D-01
  NAQ= 3 NIQ= 1 NSBSP= 9 NCGR= 50 IU= 38 F= .2291354D+01 G= .108D-01
  NAQ= 3 NIQ= 2 NSBSP= 8 NCGR= 53 IU= 38 F= .2291348D+01 G= .621D-02
  NAQ= 4 NIQ= 0 NSBSP= 9 NCGR= 53 IU= 38 F= .2291348D+01 G= .621D-02
  NAQ= 4 NIQ= 1 NSBSP= 8 NCGR= 55 IU= 40 F= .2291345D+01 G= .373D-02
  NAQ= 5 NIQ= 0 NSBSP= 10 NCGR= 55 IU= 40 F= .2291345D+01 G= .437D-02
  NAQ= 6 NIQ= 0 NSBSP= 11 NCGR= 55 IU= 40 F= .2291345D+01 G= .411D-02
  NAQ= 6 NIQ= 1 NSBSP= 11 NCGR= 56 IU= 40 F= .2291344D+01 G= .396D-02
  NAQ= 6 NIQ= 2 NSBSP= 10 NCGR= 57 IU= 40 F= .2291343D+01 G= .318D-02
  NAQ= 6 NIQ= 3 NSBSP= 9 NCGR= 58 IU= 40 F= .2291342D+01 G= .253D-02

```

```

NAQ= 6 NIQ= 4 NSBSP= 8 NCGR= 59 IU= 41 F= .2291342D+01 G= .246D-02
NAQ= 6 NIQ= 5 NSBSP= 7 NCGR= 61 IU= 41 F= .2291341D+01 G= .233D-02
NAQ= 6 NIQ= 6 NSBSP= 6 NCGR= 67 IU= 41 F= .2291341D+01 G= .197D-02
NAQ= 6 NIQ= 7 NSBSP= 5 NCGR= 72 IU= 40 F= .2291339D+01 G= .638D-03
NAQ= 7 NIQ= 0 NSBSP= 7 NCGR= 72 IU= 40 F= .2291339D+01 G= .194D-02
NAQ= 7 NIQ= 1 NSBSP= 6 NCGR= 77 IU= 39 F= .2291335D+01 G= .190D-02
NAQ= 7 NIQ= 2 NSBSP= 5 NCGR= 82 IU= 39 F= .2291335D+01 G= .186D-02
NAQ= 7 NIQ= 3 NSBSP= 4 NCGR= 86 IU= 36 F= .2291334D+01 G= .149D-02
NAQ= 8 NIQ= 0 NSBSP= 6 NCGR= 86 IU= 36 F= .2291334D+01 G= .178D-02
NAQ= 8 NIQ= 1 NSBSP= 5 NCGR= 90 IU= 36 F= .2291331D+01 G= .187D-02
NAQ= 8 NIQ= 2 NSBSP= 4 NCGR= 93 IU= 36 F= .2291331D+01 G= .944D-03
NAQ= 8 NIQ= 3 NSBSP= 4 NCGR= 94 IU= 36 F= .2291330D+01 G= .000D+00
0 NAQ= 8 NIQ= 3 NSBSP= 4 NCGR= 94 IU= 36 F= .2291330D+01 G= .000D+00
0 NIT= 0 NFV= 0 NFG= 0 NDC= 500 F= .229D+01 G= .000D+00
FF= .2291330455D+01
X = .1601960000D+01 .2010000000D+01 .2022180600D+01 .2034483006D+01
.2046647093D+01 .2056885037D+01 .2065335158D+01 .2073869780D+01
.2082856706D+01 .2093685273D+01 .2106259114D+01 .2119580857D+01
.2133035817D+01 .2146625327D+01 .2160350732D+01 .2172205073D+01
.2181910478D+01 .2191338011D+01 .2200859818D+01 .2210476844D+01
.2220190040D+01 .4080400000D+00 .1218060000D-01 .1230240600D-01
.1216408687D-01 .1023794428D-01 .8450120803D-02 .8534622011D-02
.8986925666D-02 .1082856706D-01 .1257384126D-01 .1332174283D-01
.1345496026D-01 .1358950986D-01 .1372540496D-01 .1185434141D-01
.9705405081D-02 .9427532305D-02 .9521807628D-02 .9617025705D-02
.9713195962D-02

```

6.15. 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
CALL UKMCI1(1, 1, 0.22D0,ICG,JCG,CG)
CALL UKMCI1(1, 2, 0.20D0,ICG,JCG,CG)
CALL UKMCI1(1, 3, 0.19D0,ICG,JCG,CG)
CALL UKMCI1(1, 4, 0.25D0,ICG,JCG,CG)
CALL UKMCI1(1, 5, 0.15D0,ICG,JCG,CG)
CALL UKMCI1(1, 6, 0.11D0,ICG,JCG,CG)
CALL UKMCI1(1, 7, 0.12D0,ICG,JCG,CG)
CALL UKMCI1(1, 8, 0.13D0,ICG,JCG,CG)
CALL UKMCI1(1, 9, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(2, 1,-1.46D0,ICG,JCG,CG)
CALL UKMCI1(2, 3,-1.30D0,ICG,JCG,CG)
CALL UKMCI1(2, 4, 1.82D0,ICG,JCG,CG)
CALL UKMCI1(2, 5,-1.15D0,ICG,JCG,CG)
CALL UKMCI1(2, 7, 0.80D0,ICG,JCG,CG)
CALL UKMCI1(2,10, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(3, 1, 1.29D0,ICG,JCG,CG)
CALL UKMCI1(3, 2,-0.89D0,ICG,JCG,CG)
CALL UKMCI1(3, 5,-1.16D0,ICG,JCG,CG)
CALL UKMCI1(3, 6,-0.96D0,ICG,JCG,CG)
CALL UKMCI1(3, 8,-0.49D0,ICG,JCG,CG)
CALL UKMCI1(3,11, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(4, 1,-1.10D0,ICG,JCG,CG)
CALL UKMCI1(4, 2,-1.06D0,ICG,JCG,CG)
CALL UKMCI1(4, 3, 0.95D0,ICG,JCG,CG)
CALL UKMCI1(4, 4,-0.54D0,ICG,JCG,CG)
CALL UKMCI1(4, 6,-1.78D0,ICG,JCG,CG)
CALL UKMCI1(4, 7,-0.41D0,ICG,JCG,CG)
CALL UKMCI1(4,12, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(5, 4,-1.43D0,ICG,JCG,CG)
CALL UKMCI1(5, 5, 1.51D0,ICG,JCG,CG)

```

```

CALL UKMCI1(5, 6, 0.59D0,ICG,JCG,CG)
CALL UKMCI1(5, 7,-0.33D0,ICG,JCG,CG)
CALL UKMCI1(5, 8,-0.43D0,ICG,JCG,CG)
CALL UKMCI1(5,13, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(6, 2,-1.72D0,ICG,JCG,CG)
CALL UKMCI1(6, 3,-0.33D0,ICG,JCG,CG)
CALL UKMCI1(6, 5, 1.62D0,ICG,JCG,CG)
CALL UKMCI1(6, 6, 1.24D0,ICG,JCG,CG)
CALL UKMCI1(6, 7, 0.21D0,ICG,JCG,CG)
CALL UKMCI1(6, 8,-0.26D0,ICG,JCG,CG)
CALL UKMCI1(6,14, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(7, 1, 1.12D0,ICG,JCG,CG)
CALL UKMCI1(7, 4, 0.31D0,ICG,JCG,CG)
CALL UKMCI1(7, 7, 1.12D0,ICG,JCG,CG)
CALL UKMCI1(7, 9,-0.36D0,ICG,JCG,CG)
CALL UKMCI1(7,15, 1.00D0,ICG,JCG,CG)
CALL UKMCI1(8, 2, 0.45D0,ICG,JCG,CG)
CALL UKMCI1(8, 3, 0.26D0,ICG,JCG,CG)
CALL UKMCI1(8, 4,-1.10D0,ICG,JCG,CG)
CALL UKMCI1(8, 5, 0.58D0,ICG,JCG,CG)
CALL UKMCI1(8, 7,-1.03D0,ICG,JCG,CG)
CALL UKMCI1(8, 8, 0.10D0,ICG,JCG,CG)
CALL UKMCI1(8,16, 1.00D0,ICG,JCG,CG)
$ENDSET
$SET(FGMODEL F)
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=100
$NC=8
$NCL=8
$MC=200
$KBF=2
$KBC=1
$MOUT=2
$NOUT=1
$JACC='S'

```

```

$HESF='S'
$BATCH
$STANDARD

```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify 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 \$FGMODEL we specify analytically the value and the gradient of the model function.

d) Problem solution (basic screen output):

```

      NIT=    0   NFV=    2   NFG=    2   F=   .421D+04   G=   .100D+01
      NIT=    1   NFV=    3   NFG=    3   F=   .325D+03   G=   .000D+00
      NIT=    2   NFV=    4   NFG=    4   F=   .252D+03   G=   .000D+00
      NIT=    3   NFV=    5   NFG=    5   F=   .246D+03   G=   .152D+02
      NIT=    4   NFV=    6   NFG=    6   F=   .245D+03   G=   .174D+01
      NIT=    5   NFV=    7   NFG=    7   F=   .245D+03   G=   .182D+01
      NIT=    6   NFV=    8   NFG=    8   F=   .245D+03   G=   .502D-01
      NIT=    7   NFV=    9   NFG=    9   F=   .245D+03   G=   .170D-01
      NIT=    8   NFV=   10   NFG=   10   F=   .245D+03   G=   .952D-04
      NIT=    9   NFV=   11   NFG=   11   F=   .245D+03   G=   .000D+00
0  NIT=    9   NFV=   11   NFG=   11   NDC= 200   F=   .245D+03   G=   .000D+00
FF = .2448996975D+03
X   = .3984822768D-01 .7919832074D+00 .2028707289D+00 .8443603623D+00
      .1269907517D+01 .9347380182D+00 .1681960181D+01 .1553019234D+00
      .1567869438D+01 .0000000000D+00 .0000000000D+00 -.1066732118D-19
      .6602062241D+00 .0000000000D+00 .6742558679D+00 .0000000000D+00

```

6.16. Large-scale optimization with nonlinear equality constraints

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \sum_{i=1}^n (f_i^A(x))^2$$

where $n = 100$ and

$$\begin{aligned}
f_1^A(x) &= (3 - 2x_1)x_1 - x_{1+1} + 1 & , i = 1 \\
f_i^A(x) &= (3 - 2x_i)x_i - x_{i-1} - x_{i+1} + 1 & , 2 \leq i \leq n - 1 \\
f_n^A(x) &= (3 - 2x_n)x_n - x_{n-1} + 1 & , i = n
\end{aligned}$$

over the set given by the nonlinear equality constraints

$$8x_i(x_i^2 - x_{i-1}) - 2(1 - x_i) + 4(x_i - x_{i+1}^2) + x_{i-1}^2 - x_{i-2} + x_{i+1} - x_{i+2}^2 = 0, \quad 3 \leq i \leq n - 2$$

The starting point is $x_i = -1$, $1 \leq i \leq n$. 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.DO
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.DO
4 CONTINUE
$ENDSET
$SET(FMODEL)
  FF=0.DO
```

```

DO 5 J=1,NF
WA=(3.DO-2.DO*X(J))*X(J)+1.DO
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(GMODEL F)
DO 6 J=1,NF
GF(J)=0.DO
6 CONTINUE
DO 7 J=1,NF
WA=(3.DO-2.DO*X(J))*X(J)+1.DO
IF (J.GT. 1) WA=WA-X(J-1)
IF (J.LT.NF) WA=WA-X(J+1)
WB=2.DO*WA
GF(J)=GF(J)+WB*(3.DO-4.DO*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(FMODEL C)
K=KC+2
FC=8.DO*X(K)*(X(K)**2-X(K-1))-2.DO*(1.DO-X(K))+
& 4.DO*(X(K)-X(K+1)**2)+X(K-1)**2-X(K-2)+X(K+1)-
& X(K+2)**2
$ENDSET
$SET(GMODEL C)
K=KC+2
GC(K-2)=-1.DO
GC(K-1)=-8.DO*X(K)+2.DO*X(K-1)
GC(K)=24.DO*X(K)**2-8.DO*X(K-1)+6.DO
GC(K+1)=-8.DO*X(K+1)+1.DO
GC(K+2)=-2.DO*X(K+2)
$ENDSET
$NF=100
$M=1500
$NC=96
$NCL=0
$MC=500
$KBC=1
$MOUT=2
$JACC='S'
$HESF='S'
$FORM='SE'
$FMIN=0
$BATC
$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 $2 \cdot NF - 1 = 199$. 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 only the left sides CL(KC), $1 \leq KC \leq NC$, and we can set \$KBC=1. The specification \$FMIN=0 is used, since the objective function value cannot be less than zero. By using the macrovariable \$FMODEL we specify analytically the value of the model function. By using the macrovariable \$GMODEL we specify analytically the gradient of the model function. By using the macrovariable \$FMODEL we specify analytically the values of the constraint functions. By using the macrovariable \$GMODEL we specify analytically the gradients of the constraint functions. The choice \$FORM='SE' correspond to inexact recursive quadratic programming methods for equality constrained problems.

d) problem solution (basic screen output):

```

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= .328D+03 C= .122D+02 G= .112D+02
NIC=  0 NIT=  2 NFV=  3 NFG=  30 F= .270D+03 C= .369D+01 G= .220D+01
NIC=  0 NIT=  3 NFV=  4 NFG=  40 F= .240D+03 C= .181D+01 G= .244D+01
NIC=  0 NIT=  4 NFV=  5 NFG=  50 F= .140D+03 C= .535D+01 G= .200D+02
NIC=  0 NIT=  5 NFV=  7 NFG=  60 F= .212D+04 C= .482D+01 G= .183D+02
NIC=  0 NIT=  6 NFV=  8 NFG=  70 F= .275D+03 C= .236D+01 G= .389D+02
NIC=  0 NIT=  7 NFV=  9 NFG=  80 F= .166D+02 C= .453D+00 G= .911D+01
NIC=  0 NIT=  8 NFV= 10 NFG=  90 F= .561D+01 C= .138D+00 G= .211D+01
NIC=  0 NIT=  9 NFV= 11 NFG= 100 F= .533D+01 C= .274D-01 G= .343D+00
NIC=  0 NIT= 10 NFV= 12 NFG= 110 F= .529D+01 C= .159D-02 G= .220D-01
NIC=  0 NIT= 11 NFV= 13 NFG= 120 F= .529D+01 C= .837D-04 G= .359D-02
NIC=  0 NIT= 12 NFV= 14 NFG= 130 F= .529D+01 C= .364D-05 G= .180D-03
NIC=  0 NIT= 13 NFV= 15 NFG= 140 F= .529D+01 C= .368D-07 G= .549D-06
0 NIC=  0 NIT= 14 NFV= 15 NFG= 140 F= .529D+01 C= .368D-07 G= .549D-06

```

6.17. 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

$$\frac{dy_1(t)}{dt} = y_2(t), \quad y_1(0) = x_1$$

$$\frac{dy_2(t)}{dt} = (1 - y_1^2(t))y_2(t) - y_1(t), \quad y_2(0) = 1$$

b) Problem specification (input field):

```

$SET(INPUT)
  X(1)=ZERO
  TA=ZERO
  TAMAX=1.5D 0
$ENDSET
$SET(FMODEL F)
  FF=HALF*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODEL F)
  DF(1)=YA(1)
  DF(2)=YA(2)
$ENDSET
$SET(FMODEL A)
  FA=HALF*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODEL A)
  DA(1)=YA(1)
  DA(2)=YA(2)
$ENDSET
$SET(FMODEL E)
  GO TO (1,2) KE
1 FE=YA(2)
  GO TO 3
2 FE=(ONE-YA(1)**2)*YA(2)-YA(1)
3 CONTINUE
$ENDSET
$SET(DMODEL E)
  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(FMODEL Y)
  GO TO (7,8) KE
7 FE=X(1)
  GO TO 9
8 FE=ONE
9 CONTINUE
$ENDSET
$SET(GMODEL Y)
  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 as the initial value of the variable x_1 as the initial and terminal times 0 and T, respectively. By using the macrovariables \$FMODELA and \$DMODELA we specify subintegral function and by using the macrovariables \$FMODEL and \$DMODEL we specify terminal function. Right hand sides of the differential equations are specified by using the macrovariables \$FMODELE and \$DMODELE, while initial values and their derivatives are given by using the macrovariables \$FMODELY and \$GMODELY. The option \$MODEL='DF' indicates general integral criterion.

d) Problem solution (basic screen output):

```

      NIT=    0   NFV=    1   NFG=    0   F=   .276D+01   G=   .242D+01
      NIT=    1   NFV=    3   NFG=    0   F=   .197D+01   G=   .513D+00
      NIT=    2   NFV=    4   NFG=    0   F=   .194D+01   G=   .468D-02
      NIT=    3   NFV=    5   NFG=    0   F=   .194D+01   G=   .122D-03
      NIT=    4   NFV=    6   NFG=    0   F=   .194D+01   G=   .205D-07
0  NIT=    4   NFV=    6   NFG=    0   NDC=    0   F=   .194D+01   G=   .205D-07
FF = .7671653645D+00
X   = .6169838477D+00

```

6.18. 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

$$\frac{dy_1(t)}{dt} = -x_1 y_1(t), \quad y_1(0) = x_2$$

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
$MODEL= 'DQ'
$MODEL= 'YES'
$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 as the initial values of the variables x_1 and x_2 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 initial values and their derivatives are given by using the macrovariables \$FMODELY and \$GMODELY. The option \$MODEL='DQ' together with \$MODEL='YES' indicates special integral criterion.

d) Problem solution (basic screen output):

```

    NIT=    0   NFV=    1   NFG=    1   F=   .250D+00   G=   .693D+00
    NIT=    1   NFV=    3   NFG=    2   F=   .338D-01   G=   .114D+00
    NIT=    2   NFV=    5   NFG=    3   F=   .160D-02   G=   .613D-02
    NIT=    3   NFV=    7   NFG=    4   F=   .120D-04   G=   .225D-02
    NIT=    4   NFV=    9   NFG=    5   F=   .191D-08   G=   .300D-04
    NIT=    5   NFV=   11   NFG=    6   F=   .279D-15   G=   .200D-08
0  NIT=    5   NFV=   11   NFG=    6   NDC=    7   F=   .279D-15   G=   .200D-08

```

F = .2793082861D-15
 X = .9999999725D+00 .9999999990D+00

6.19. Initial value problem for ordinary differential equations

a) Problem description:

Suppose we have to find a solution of the Van der Pol equation

$$\frac{dy_1(t)}{dt} = y_2(t), \quad y_1(0) = 2$$

$$\frac{dy_2(t)}{dt} = (1 - y_1^2(t))y_2(t) - y_1(t), \quad y_2(0) = 0$$

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='NO'
$MED=2
$NOUT=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify as the initial values of the variables y_1 and y_2 as the initial and terminal times 0 and T, respectively. Right hand sides of the differential equations are specified by using the macrovariable \$FMODELE. The option \$MODEL='NO' indicates integration of a system of ordinary differential equations.

d) Problem solution (basic screen output):

```

0  NSTP= 47  NACC= 35  NREJ= 12  NEV =1228
1  AT = .000000D+00
   AY = .200000D+01 .000000D+00
2  AT = .500000D+00
   AY = .183772D+01 -.534523D+00
3  AT = .100000D+01
   AY = .150814D+01 -.780218D+00
4  AT = .150000D+01
   AY = .104093D+01 -.112432D+01
5  AT = .200000D+01
   AY = .323317D+00 -.183297D+01
6  AT = .250000D+01
   AY = -.840966D+00 -.267748D+01
7  AT = .300000D+01
   AY = -.186607D+01 -.102106D+01
8  AT = .350000D+01
   AY = -.198111D+01 .278099D+00
9  AT = .400000D+01
   AY = -.174177D+01 .624666D+00
10 AT = .450000D+01
   AY = -.136968D+01 .874905D+00
11 AT = .500000D+01
   AY = -.837077D+00 .130709D+01
12 AT = .550000D+01
   AY = .149210D-01 .218756D+01
13 AT = .600000D+01
   AY = .127904D+01 .243781D+01
14 AT = .650000D+01
   AY = .198166D+01 .383461D+00
15 AT = .700000D+01
   AY = .192015D+01 -.435839D+00
16 AT = .750000D+01
   AY = .163005D+01 -.702764D+00
17 AT = .800000D+01
   AY = .121323D+01 -.987814D+00
18 AT = .850000D+01
   AY = .597989D+00 -.154333D+01
19 AT = .900000D+01
   AY = -.412915D+00 -.252690D+01
20 AT = .950000D+01
   AY = -.163855D+01 -.178124D+01
21 AT = .100000D+02
   AY = -.200834D+01 .329065D-01

```

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: Bayesian 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] 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.
- [10] C.G.Broyden: A class of methods for solving nonlinear simultaneous equations. *Math. of Comput.* 19 (1965) 577-593.
- [11] 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.
- [12] J.R. Bunch, B.N. Parlett: Direct methods for solving symmetric indefinite systems of linear equations. *SIAM J. Numer. Anal.* 8 (1971) 639-655.
- [13] 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.
- [14] 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.
- [15] T.F.Chan: Rank revealing QR factorizations. *Linear Algebra Appl.* 88/89 (1987) 67-82.
- [16] T.F.Coleman, B.S.Garbow J.S.Moré: Software for estimation sparse Hessian matrices. *ACM Trans. of Math. Software* 11 (1985) 363-367.
- [17] T.F.Coleman: Large sparse numerical optimization. Springer-Verlag, Berlin, 1984.
- [18] T.F.Coleman, B.S.Garbow, J.S.Moré: Software for estimating sparse Jacobian matrices. *ACM Trans. of Math. Software* 10 (1984) 329-345.
- [19] 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.
- [20] H.Curry: The method of steepest descent for nonlinear minimization problems. *Quart. Appl. Math.* 2 (1944) 258-261.

- [21] W.C.Davidon: Variable metric method for minimisation. A.E.C. Research and Development Report ANL-5990, 1959.
- [22] W.C.Davidon: Optimally conditioned optimization algorithms without line searches. *Math. Programming* 9 (1975) 1-30.
- [23] T.A.Davis, I.S.Duff: An unsummetric pattern multifrontal method for sparse LU factorization. Report No. TR-93-018, CIS Department, University of Florida, Gainesville 1993.
- [24] N.Y.Deng, Y.Xiao, F.J.Zhou: Nonmonotonic trust region algorithm. *JOTA* 76 (1993) 259-285.
- [25] R.S.Dembo, T.Steihaug: Truncated-Newton algorithms for large-scale unconstrained minimization. *Math. Programming* 26 (1983) 190-212.
- [26] 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.
- [27] 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.
- [28] J.E.Dennis, R.B.Schnabel: Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall, Englewood Cliffs, New Jersey 1983.
- [29] J.E.Dennis, R.E.Welsch: Techniques for Nonlinear Least Squares and Robust Regression. *Communications in Statistics B* 7 (1978) 345-359.
- [30] I.S.Duff, J.K.Reid: The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. of Math. Software* 9 (1983) 302-325.
- [31] R.Fletcher: A new approach to variable metric algorithms. *Computer J.* 13 (1970) 317-322.
- [32] R.Fletcher: A modified Marquardt subroutine for nonlinear least squares. Report No. R-6799, Theoretical Physics Division, A.E.R.E. Harwell, 1971.
- [33] R.Fletcher: A general quadratic programming algorithm. *J. Inst. Math. Appl.* 7 (1971) 76-91.
- [34] R.Fletcher: Practical methods of optimization (Second edition). Wiley, New York, 1987.
- [35] R.Fletcher, M.J.D.Powell: A rapidly convergent descent method for minimization. *Computer J.* 6 (1963) 163-168.
- [36] R.Fletcher, C.M.Reeves: Function minimization by conjugate gradients. *Computer J.* 7 (1964) 149-154.
- [37] R.Fletcher, C.Xu: Hybrid methods for nonlinear least squares. *IMA J. Numer. Anal.* 7 (1987) 371-389.
- [38] R.Fletcher: Second order corrections for nondifferentiable optimization. In: "Numerical analysis, Dundee 1981" (G.A.Watson ed.), Lecture Notes in Mathematics 912, Springer-Verlag, Berlin 1982.
- [39] R.P.Ge: A filled function method for finding a global minimizer of a function of several variables. *Math. Programming* 46 (1990) 191-204.
- [40] 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.
- [41] J.C.Gilbert, C.Lemarechal: Some numerical experiments with variable-storage quasi-Newton algorithms. *Math. Programming*, 45 (1989) 407-435.

- [42] P.E.Gill, W.Murray: A numerically stable form of the simplex algorithm. *Linear Algebra Appl.* 7 (1973) 99-138.
- [43] P.E.Gill, W.Murray: Newton type methods for unconstrained and linearly constrained optimization. *Math. Programming* 7 (1974) 311-350.
- [44] P.E.Gill, W.Murray: Numerically stable methods for quadratic programming. *Math. Programming* 14 (1978) 349-372.
- [45] P.E.Gill, W.Murray, M.H.Wright: *Practical optimization*. Academic Press, London 1981.
- [46] D.Goldfarb: A family of variable metric algorithms derived by variational means. *Math Comput.* 24 (1970) 23-26.
- [47] D.Goldfarb, A.U.Idnani: A numerically stable dual method for solving strictly convex quadratic programmes. Report No. 81-102, Dept.of Computer Sci., The City College of New York, 1981.
- [48] A.A.Goldstein: On steepest descent. *SIAM J. Control* 3 (1965) 147-151.
- [49] G.H.Golub, C.F.Van Loan: *Matrix computations* (second edition). Johns Hopkins University Press, Baltimore 1989.
- [50] A.Griewank, P.L.Toint: Partitioned variable metric updates for large scale structured optimization problems. *Numer. Math.* 39 (1982) 119-137.
- [51] L.Grippio, F.Lampariello, S.Lucidi: A nonmonotone line search technique for Newton's method. *SIAM J. Numer. Anal.* 23 (1986) 707-716.
- [52] E.Hairer, S.P.Norsett, G.Wanner: *Solving ordinary differential equations I*. Springer Series in Computational Mathematics 8, Springer Verlag, Berlin 1987.
- [53] S.P.Han: Variable metric methods for minimizing a class of nondifferentiable functions. *Math. Programming* 20 (1981) 1-13.
- [54] M.R.Hestenes, C.M.Stiefel: Methods of conjugate gradient for solving linear systems. *J. Res. NBS* 49 (1964) 409-436.
- [55] W.Hock, K.Schittkowski: Test examples for nonlinear programming codes. *Lecture notes in economics and mathematical systems* 187. Springer Verlag, Berlin 1981.
- [56] R.Hooke, T.A.Jeeves: Direct search solution of numerical and statistical problems. *J. Assoc. Comp. Mach.* 8 (1961) 212-221.
- [57] S.Hoshino: A formulation of variable metric methods. *J. Inst. Math. Appl.* 10 (1972) 394-403.
- [58] 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.
- [59] 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.
- [60] C.M.Ip, M.J.Todd: Optimal conditioning and convergence in rank one quasi-Newton updates. *SIAM J. Numer. Anal.* 25 (1988) 206-221.
- [61] K.C.Kiwiel: An ellipsoid trust region bundle method for nonsmooth convex minimization. *SIAM J. on Control and Optimization* 27 (1989) 737-757.
- [62] C.L.Lawson, R.J.Hanson: *Solving least squares problems*. Prentice-Hall, Englewood Cliffs, New Jersey 1974.

- [63] A.V.Levy, A.Montalvo: The tunneling algorithm for the global minimization of functions. *SIAM Journal Sci. Stat. Comp.* 6 (1985) 15-19.
- [64] G.Li: Successive column correction algorithms for solving sparse nonlinear systems of equations. *Mathematical Programming* 43 (1989) 187-207.
- [65] P.Lindstrom, P.A.Wedin: A new linesearch algorithm for nonlinear least squares problems. *Math. Programming* 29 (1984) 268-296.
- [66] D.C.Liu, J.Nocedal: On the limited memory BFGS method for large-scale optimization. *Math. Programming* 45 (1989) 503-528.
- [67] 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.
- [68] L.Lukšan: An implementation of recursive quadratic programming variable metric methods for linearly constrained nonlinear minimax approximation. *Kybernetika* 21 (1985) 22-40.
- [69] L.Lukšan: Variable metric methods. Unconstrained minimization. Academia, Prague 1990 (in Czech).
- [70] L.Lukšan: Computational experience with improved variable metric methods for unconstrained minimization. *Kybernetika* 26 (1990) 415-431.
- [71] L.Lukšan: Computational experience with improved conjugate gradient methods for unconstrained minimization. *Kybernetika* 28 (1992) 249-262.
- [72] L.Lukšan: A note on comparison of statistical software for nonlinear regression. *Computational Statistics Quarterly* 6 (1991) 321-324.
- [73] L.Lukšan: Variationally derived scaling and variable metric updates from the preconvex part of the Broyden family. *JOTA* 73 (1992) 299-307.
- [74] L.Lukšan: Inexact trust region method for large sparse nonlinear least squares. *Kybernetika* 29 (1993) 305-324.
- [75] L.Lukšan: Efficient trust region method for nonlinear least squares. Report No. 503, Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.
- [76] L.Lukšan: Computational experience with known variable metric updates. *JOTA* 83 (1994) 27-47.
- [77] L.Lukšan: Inexact trust region method for large sparse systems of nonlinear equations. *JOTA* 81 (1994) 569-590.
- [78] L.Lukšan: Combined trust region methods for nonlinear least squares. Report No. 555, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1993.
- [79] L.Lukšan: Hybrid methods for large sparse nonlinear least squares. Report No. 561, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1993.
- [80] L.Lukšan, J.Vlček: Optimization of dynamical systems. Report No. 584, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1994
- [81] 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
- [82] L.Lukšan, M.Rozložník: Globally convergent methods for large sparse systems of nonlinear equations. Report No. 649, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1995

- [83] L.Lukšan, J.Vlček: Truncated Newton methods for large sparse equality constrained nonlinear programming problems. Report No. 652, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1995
- [84] L.Lukšan, J.Vlček: A bundle-newton method for nonsmooth unconstrained minimization. Report No. 654, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague 1995
- [85] M.M.Mäkelä, J.Neittaanmäki: Nonsmooth Optimization. World Scientific Publishing Co. Ltd. London 1992.
- [86] E.S.Marwill: Exploiting sparsity in Newton-like methods. Ph.D. Thesis, Cornell University, Ithaca 1978.
- [87] J.M.Martinez: A quasi-Newton method with modification of one column per iteration. Computing 33 (1984) 353-362.
- [88] 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.
- [89] 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.
- [90] J.J.Moré: The Levenberg-Maquardt algorithm. Implementation and theory. In: "Numerical Analysis" (G.A.Watson ed.) Springer Verlag, Berlin 1978.
- [91] J.J.Moré, B.S.Garbow, K.E.Hillström: Testing unconstrained optimization software. ACM Trans. Math. Software 7 (1981) 17-41.
- [92] J.J.Moré, D.C.Sorensen: Computing a trust region step. Report No. ANL-81-83, Argonne National Laboratory. 1981.
- [93] J.A.Nelder, R.Mead: A simplex method for function minimization. Computer J. 7 (1965) 308-313.
- [94] J.Nocedal: Updating quasi-Newton Matrices with limited storage. Math. Comput. 35 (1980) 773-782.
- [95] J.Nocedal, Y.Yuan: Combining trust region and line search techniques. To appear.
- [96] 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.
- [97] S.S.Oren, E. Spedicato: Optimal conditioning of self scaling variable metric algorithms. Math Programming 10 (1976) 70-90.
- [98] 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.
- [99] E.Polak, G.Ribière: Note sur la convergence des methodes de directions conjuguées. Revue Francaise Inform. Mech. Oper. 16-R1(1969) 35-43.
- [100] 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.
- [101] 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.
- [102] M.J.D.Powell: Restart procedures of the conjugate gradient method. Math. Programming 12 (1977) 241-254.

- [103] M.J.D.Powell: A fast algorithm for nonlinearly constrained optimization calculations. In: "Numerical analysis" (G.A.Watson ed.). Springer Verlag, Berlin 1977.
- [104] M.J.D.Powell: Convergence properties of algorithms for nonlinear optimization. Report No. DAMPT 1985/NA1, University of Cambridge, 1985.
- [105] H.Ramsin, P.A.Wedin: A Comparison of Some Algorithms for the Nonlinear Least Squares Problem. BIT 17 (1977) 72-90.
- [106] 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.
- [107] 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.
- [108] R.B.Schnabel, E.Eskow: A new Choleski factorization. SIAM J. Sci. Stat. Comput. 11 (1990), 1136-1158.
- [109] 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.
- [110] D.F.Shanno: Conditioning of quasi-Newton methods for function minimization. Math. Comput. 24 (1970) 647-656.
- [111] D.F.Shanno, K.J.Phua: Matrix conditioning and nonlinear optimization. Math. Programming 14 (1978) 144-160.
- [112] E.Spedicato: A class of rank-one positive definite quasi-Newton updates for unconstrained minimization. Math. Operationsforsch. Statist. Ser. Optimization 14 (1963) 61-70.
- [113] E.Spedicato, M.T.Vespucci: Numerical experiments with variations of the Gauss-Newton algorithm for nonlinear least squares. JOTA 57 (1988) 323-339.
- [114] 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.
- [115] T.Steihaug: Local and superlinear convergence for truncated iterated projections methods. Math. Programming 27 (1983) 176-190.
- [116] T.Steihaug: The conjugate gradient method and trust regions in large-scale optimization. SIAM J. Numer. Anal. 20 (1983) 626-637.
- [117] 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.
- [118] G.W.Stewart: A modification of Davidon's minimization method to accept difference approximations of derivatives. J. ACM 14 (1967) 72-83.
- [119] 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.
- [120] M.Šiška: Macroprocessor UFO (version 1990). Report No. 484 (in Czech), Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.
- [121] P.L.Toint: On sparse and symmetric matrix updating subject to a linear equation. Math of Comp. 31 (1977) 954-961.

- [122] P.L.Toint: On large scale nonlinear least squares calculations. *SIAM J. Sci. Stat. Comput.* 8 (1987) 416-435.
- [123] C.H.Tong: A comparative study of preconditioned Lanczos methods for nonsymmetric linear systems. Report No. SAND91-8240B, Sandia National Laboratories, Livermore 1992.
- [124] D.Touati-Ahmed, C.Storey: Efficient hybrid conjugate gradient techniques. *JOTA* 64 (1990), pp. 379-397.
- [125] M.Tůma: A quadratic programming algorithm for large and sparse problems. *Kybernetika* 27 (1991) 155-167.
- [126] M.Tůma: Sparse fractioned variable metric updates. Report No. 497, Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.
- [127] 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.
- [128] H.Yabe, T.Takahashi: Factorized quasi-Newton methods for nonlinear least squares problems. *Math. Programming* 51 (1991) 75-100.
- [129] 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.
- [130] 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.
- [131] A.Žilinskas, A.A.Thorn: Global optimization. Springer Verlag, Berlin 1990.

Appendix: Graphical screen output