Institute of Computer Science
Academy of Sciences of the Czech Republic

UFO 2017
Interactive System for Universal Functional Optimization

L.Lukšan, M.Tůma, C.Matonoha, J.Vlček, N.Ramešová, M.Šiška, J.Hartman

Technical report No. 1252

November 2017

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

Institute of Computer Science
Academy of Sciences of the Czech Republic

UFO 2017
Interactive System for Universal Functional Optimization

L.Lukšan, M.Tůma, C.Matonoha, J.Vlček, N.Ramešová, M.Šiška, J.Hartman [1]

Technical report No. 1252

November 2017

Abstract:

This report contains a description of the interactive system for universal functional optimization UFO, version 2017. This version contains interfaces to the MATLAB and SCILAB graphics environments.

Keywords:
Numerical optimization, nonlinear programming, nonlinear approximation, algorithms, software systems.

[1]This work was supported the Institute of Computer Science of the AS CR (RVO: 67985807). L.Lukšan is also from Technical University of Liberec, Hálkova 6, 461 17 Liberec.

Contents

1 Introduction to the UFO system

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

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

The special realization of the UFO system described in the subsequent sections makes this system portable and extensible. 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 three phases. In the first phase the optimization problem is specified and an optimization method is selected. This can be done in three different ways:

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

The first phase is realized by using the UFO control language preprocessor (UFOCLP). This preprocessor uses the BEL interpreter controlled by input template `UZDCLP.I` (see Appendix B.7). The BEL interpreter is written in Fortran 77 and its output is a Fortran 77 source program (`P.FOR` or `P.F`). This conception is very advantageous for the following reasons:

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

In the second phase, the UFO source program `P.FOR` (or `P.F`) is compiled by using a suitable Fortran 77 compiler and a final executable program is linked by using library modules. In the third phase, the final executable program is started and thus results which can be viewed by using extensive output means are obtained.

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

The UFO macroprocessor works in two passes. In the first pass, the file `P.TMP` is created. This file is a predecessor of the UFO source program. It contains macroinstructions and macrovariables which are processed in the second pass. The UFO source program `P.FOR` (or `P.F`) is the result of the second pass.

1.2 The UFO versions for PC computers

The UFO system is distributed in five versions depending on the installed operating system and the Fortran compiler:

UFOW6 - the 32 or 64 bit Windows system with Digital Visual Fortran compiler (version 6).
UFOW8 - the 32 or 64 bit Windows system with Intel Visual Fortran compiler (version 8).
UFOWN - the 32 or 64 bit Windows system with NAG Fortran compiler (version 5.2).
UFOWG - the 32 or 64 bit Windows system with GNU Fortran compiler (version 5.0).
UFOLG - the Red Hat Centos Linux system with GNU Fortran 95 compiler.

These versions can be found on page `http://www.cs.cas.cz/luksan/ufodis.html` as the files ufow6.zip, ufow8.zip, ufown.zip, ufowg.zip, ufolg.tar.gz together with the installation notes ufow6.txt, ufow8.txt, ufown.txt, ufowg.txt, ufolg.txt and the report V1252-17.pdf containing the UFO system description.

The UFO system can be implemented with different graphic dialogues (Section 4.4) and different graphic screen outputs (Section 5.3). The graphic system is chosen by using the macrovariable $GRAPHICS whose value is specified in the template `UZDCLP.I` (see Appendix B.7):

$GRAPHICS = 0 - The variant which does not use graphic tools. The graphic dialogue and the graphic screen output cannot be used, but an arbitrary Fortran compiler can be chosen in this case.

$GRAPHICS = –1 - The variant which generates output files for external graphic devices (see section 5.4). The graphic dialogue and the graphic screen output cannot be used, but an arbitrary Fortran compiler can be chosen in this case.

$GRAPHICS = –2 - The variant which either generates an output file `P.m` for the MATLAB graphics or uses the MATLAB graphics interactively to produce special screen output (see section 5.5). The graphic dialogue cannot be used, but an arbitrary Fortran compiler can be chosen in this case.

$GRAPHICS = –3 - The variant which either generates an output file `P.sci` for the SCILAB graphics or uses the SCILAB graphics interactively to produce special screen output (see section 5.6). The graphic dialogue cannot be used, but an arbitrary Fortran compiler can be chosen in this case.

$GRAPHICS = 4 - This variant requires the Microsoft Visual Fortran compiler (version 4) and uses the QuickWin graphic system with simplified control (the mouse cannot be used), so the graphic dialogue and the graphic screen output are possible.

$GRAPHICS = 6 - This variant requires the Digital Visual Fortran compiler (version 6) and uses the QuickWin graphic system with simplified control (the mouse cannot be used), so the graphic dialogue and the graphic screen output are possible.

$GRAPHICS = 8 - This variant requires the Intel Visual Fortran compiler (version 8) and uses the QuickWin graphic system with simplified control (the mouse cannot be used), so the graphic dialogue and the graphic screen output are possible.

The default values $GRAPHICS = 6, $GRAPHICS = 8, $GRAPHICS = 0, $GRAPHICS = 0, $GRAPHICS = 0 are used at versions UFOW6, UFOW8, UFOWN, UFOWG, UFOLG, respectively. In all cases, we can also set $GRAPHICS =–1 for external graphics, $GRAPHICS =–2 for the MATLAB graphics and $GRAPHICS =–3 for the SCILAB graphics.

1.3 Installation of the UFO system

The files ufow6.zip, ufow8.zip contain templates `*.I`, sample input files `*.UFO`, sample output files `*.OUT`, batch procedures `*.BAT`, programs `*.EXE` and other important files together with the subdirectory `LIB`, which contains libraries `*.LIB`. The files ufown.zip, ufowg.zip differ from the above files only in using library libufo.a contained in the main directory. The selected Windows version of the UFO system is installed by putting the file ufow6.zip (ufow8.zip, ufown.zip, ufowg.zip) into the directory `UFO` and unpacking it (e.g., using the routine `7z.exe`). Furthermore, paths and settings in the procedures (`UFO6.BAT`, `UFO8.BAT`, `UFON.BAT`, `UFOG.BAT`) and (`SET6.BAT`, `SET8.BAT`, `SETN.BAT`, `SETG.BAT`) have to be arranged to correspond to the actual directories in the PC used. For example, the procedure `UFO6.BAT` should have the form

```
CALL SET6
CALL C:\WINdows\system32\cmd.exe
```

and the procedure `SET6.BAT` should have the form

```
D:
CD D:\UFO6
COPY STANDARD.UFO P.UFO
SET LIB=C:\Program Files\Microsoft Visual Studio\DF98\LIB;
        C:\Program Files\Microsoft Visual Studio\VC98\LIB;D:\UFO6\LIB
SET INCLUDE=C:\Program Files\Microsoft Visual Studio\DF98\INCLUDE;D:\UFO6\LIB
PATH=C:\Program Files (x86)\scilab-5.4.0\bin\;C:\Program Files (x86)\java\jre6\bin;
     C:\Program Files (x86)\7-zip;C:\Program Files\Microsoft Visual Studio\Common\Tools;
     C:\Program Files\Microsoft Visual Studio\Common\Msdev98\BIN;
     C:\Program Files\Microsoft Visual Studio\DF98\BIN;
     C:\Program Files\Microsoft Visual Studio\VC98\BIN;D:\UFO6;%PATH%
:END
```

(with `D:\UFO6` and `C:\Program Files` replaced by the actual paths). More information concerning the installation is given in the text file ufow6.txt (ufow8.txt, ufown.txt, ufowg.txt).

If the connection to the CUTE test environment is required then the subdirectory `SIF` has to be created and the `*.SIF` files from the CUTE collection (page `http://www.cuter.rl.ac.uk/`) have to be copied into this subdirectory.

The WINDOWS versions UFOW6, UFOW8, UFOWN, UFOWG are provided with a simple environment based on the PSPad editor. The UFO environment is called by using the statement `UFO input_name` (procedure `UFO.BAT`). Here `input_name` is the first part of the batch file name `input_name.UFO`, that is used as a batch input for the UFO source program generation. This input file is opened in the PSPad editor and can be modified. If the parameter is omitted, the file STANDARD.UFO (a dialogue) is chosen. When the input file is prepared, we can use the scroll-bar menu (displayed by heart) to start procedures `UFOGO`, `UFOGO1`, `COMPIL`, `COMPIL1`, `GENER` (see Section 1.4), respectively. For editing files, the PSPad editor can be

called by using the statement `EDIT input_name.type` (procedure `EDIT.BAT`). Here `input_name.type` (e.g. `input_name.I` or `input_name.UFO`) is the full name of the edited file.

The LINUX version UFOLG is distributed by using the file `ufolg.tar.gz`, which contains templates `*.I`, sample input files `*.UFO`, sample output files `*.OUT`, library `ufolib.a`, executable files `gener`, `compil`, `ufogo`, program `ufobel` and other important files. The PC Linux version is installed by putting the file `ufolg.tar.gz` into the directory `ufo` and unpacking it (e.g. by using statements `gzip -d *.gz` and `tar xvf ufo.tar`). If the connection to the CUTE test environment is required, the subdirectory `ufo/sif` has to be created and the `*.SIF` files from the CUTE collection (page `http://www.cuter.rl.ac.uk/`) have to be copied into this subdirectory.

1.4 Execution of the UFO system

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

```
GENER input_name
```

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

```
COMPIL output_name
```

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

```
UFOGO problem_name
```

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

```
@ECHO OFF
IF EXIST PP.UFO COPY PP.UFO PPP.UFO
IF EXIST PP.FOR COPY PP.FOR PPP.FOR
IF EXIST PP.SIF COPY PP.SIF PPP.SIF
IF EXIST PP.I COPY PP.I PPP.I
IF EXIST Q.UFO COPY Q.UFO PP.UFO
COPY P.UFO Q.UFO
IF EXIST P.FOR COPY P.FOR PP.FOR
IF EXIST P.SIF COPY P.SIF PP.SIF
IF EXIST P.I COPY P.I PP.I
IF EXIST P.FOR DEL P.FOR
IF EXIST P.OBJ DEL P.OBJ
IF EXIST P.EXE DEL P.EXE
IF EXIST P.SIF DEL P.SIF
IF EXIST P.I DEL P.I
IF "%1"== GOTO S
COPY %1.UFO P.UFO
GOTO C
```

```
:S
COPY STANDARD.UFO P.UFO
:C
COPY UZDCLP.I BEL.TEM
DEL TFILE
GREP -i $BATCH P.UFO > GFILE
CALL TESTFILE GFILE
IF EXIST TFILE GOTO M
DEL TFILE
GREP -i $TDIALOGUE P.UFO > GFILE
CALL TESTFILE GFILE
IF EXIST TFILE GOTO M
CALL BELG
GOTO N
:M
CALL BEL
:N
COPY BEL.OUT P.TMP
IF NOT EXIST P.FOR GOTO END
IF EXIST ELFUNS.FOR TYPE ELFUNS.FOR >> P.FOR
IF EXIST GROUPS.FOR TYPE GROUPS.FOR >> P.FOR
IF EXIST RANGES.FOR TYPE RANGES.FOR >> P.FOR
IF EXIST SETTYP.FOR TYPE SETTYP.FOR >> P.FOR
IF EXIST EXTERN.FOR TYPE EXTERN.FOR >> P.FOR
IF EXIST ELFUNS.FOR DEL ELFUNS.FOR
IF EXIST GROUPS.FOR DEL GROUPS.FOR
IF EXIST RANGES.FOR DEL RANGES.FOR
IF EXIST SETTYP.FOR DEL SETTYP.FOR
IF EXIST EXTERN.FOR DEL EXTERN.FOR
:END
```

The procedure `COMPIL.BAT` has the form

```
@ECHO OFF
IF EXIST P.EXE DEL P.EXE
IF NOT EXIST P.F GOTO END
IF EXIST ELFUNS.FOR TYPE ELFUNS.FOR >> P.F
IF EXIST GROUPS.FOR TYPE GROUPS.FOR >> P.F
IF EXIST RANGES.FOR TYPE RANGES.FOR >> P.F
IF EXIST SETTYP.FOR TYPE SETTYP.FOR >> P.F
IF EXIST EXTERN.FOR TYPE EXTERN.FOR >> P.F
IF EXIST ELFUNS.FOR DEL ELFUNS.FOR
IF EXIST GROUPS.FOR DEL GROUPS.FOR
IF EXIST RANGES.FOR DEL RANGES.FOR
IF EXIST SETTYP.FOR DEL SETTYP.FOR
IF EXIST EXTERN.FOR DEL EXTERN.FOR
gfortran P.F -o P -L. libufo.a
IF NOT EXIST P.EXE GOTO END
IF EXIST PP.OUT COPY PP.OUT PPP.OUT
IF EXIST PP.DIM COPY PP.DIM PPP.DIM
IF EXIST P.OUT COPY P.OUT PP.OUT
IF EXIST P.DIM COPY P.DIM PP.DIM
IF EXIST P.OUT DEL P.OUT
```

```
IF EXIST P.DIM DEL P.DIM
P P.OUT
IF "%1"== GOTO S
IF "%1"=="P" GOTO S
COPY P.OUT %1.OUT
IF EXIST P.DAT COPY P.DAT %1.DAT
:S
IF NOT ERRORLEVEL 1 GOTO END
PAUSE
MODE CO80
:END
```

The procedure UFOGO.BAT has the form

```
@ECHO OFF
IF EXIST PP.UFO COPY PP.UFO PPP.UFO
IF EXIST PP.F COPY PP.F PPP.F
IF EXIST PP.SIF COPY PP.SIF PPP.SIF
IF EXIST PP.I COPY PP.I PPP.I
IF EXIST Q.UFO COPY Q.UFO PP.UFO
COPY P.UFO Q.UFO
IF EXIST P.F COPY P.F PP.F
IF EXIST P.SIF COPY P.SIF PP.SIF
IF EXIST P.I COPY P.I PP.I
IF EXIST P.F DEL P.F
IF EXIST P.O DEL P.O
IF EXIST P.EXE DEL P.EXE
IF EXIST P.SIF DEL P.SIF
IF EXIST P.I DEL P.I
IF "%1"== GOTO S
COPY %1.UFO P.UFO
GOTO C
:S
COPY STANDARD.UFO P.UFO
:C
COPY UZDCLP.I BEL.TEM
CALL UFOBEL
IF NOT EXIST P.F GOTO END
IF EXIST ELFUNS.FOR TYPE ELFUNS.FOR >> P.F
IF EXIST GROUPS.FOR TYPE GROUPS.FOR >> P.F
IF EXIST RANGES.FOR TYPE RANGES.FOR >> P.F
IF EXIST SETTYP.FOR TYPE SETTYP.FOR >> P.F
IF EXIST EXTERN.FOR TYPE EXTERN.FOR >> P.F
IF EXIST ELFUNS.FOR DEL ELFUNS.FOR
IF EXIST GROUPS.FOR DEL GROUPS.FOR
IF EXIST RANGES.FOR DEL RANGES.FOR
IF EXIST SETTYP.FOR DEL SETTYP.FOR
IF EXIST EXTERN.FOR DEL EXTERN.FOR
nagfor -o P P.F libufo.a
IF NOT EXIST P.EXE GOTO END
IF EXIST PP.OUT COPY PP.OUT PPP.OUT
IF EXIST PP.DIM COPY PP.DIM PPP.DIM
IF EXIST P.OUT COPY P.OUT PP.OUT
```

```
IF EXIST P.DIM COPY P.DIM PP.DIM
IF EXIST P.OUT DEL P.OUT
IF EXIST P.DIM DEL P.DIM
P P.OUT
IF "%1"== GOTO X
IF "%1"=="P" GOTO X
REM COPY P.OUT %1.OUT
REM IF EXIST P.DAT COPY P.DAT %1.DAT
:X
IF NOT ERRORLEVEL 1 GOTO END
PAUSE
MODE CO80
:END
```

In versions UFOW6, UFOW8, UFOWN, these procedures differ only by calling the Fortran compiler.

To show how the batch mode proceeds, we suppose that the model function has the form

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

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

```
$REM : model specifications
$SET(INPUT)
  X(1)=-1.2D0; X(2)= 1.0D0
$ENDSET
$SET(FMODELF)
  FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
$NF=2
$REM : the default method is used
$REM : print specifications
$MOUT=1
$NOUT=1
$REM : the batch mode is used
$BATCH
$REM : the standard form of the UFO source program is used
$STANDARD
```

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

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

Batch input files are written in the UFO control language. This language is described in Section 4.1 and in Appendix B. Here we note that a certain experience of the UFO control language can be obtained by using the demo-files `PROB01.UFO`,..., `PROB36.UFO`. These demo-files contain 36 test problems described in Chapter 7. We can solve them by using the statements `UFOGO PROB01`,..., `UFOGO PROB36`.

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

1.5 Graphic utilities of the UFO system

If the MATLAB system is installed, $GRAPHICS = –2 and $GRAPH='E' (Section 5.5), we can use the statement

```
MATGO file_name
```

to start the MATLAB graphics environment for drawing the final results. Here `file_name.m` is the input file for MATLAB. If `MATGO` statement does not contain a file name specification, then the standard input file is `P.m`. The `MATGO` statement is realized by the procedure `MATGO.BAT`, which has the form

```
@ECHO OFF
IF "%1"=="" GOTO P
IF "%1"=="@" GOTO I
IF NOT EXIST %1.m GOTO E1
CALL matlab -r "run('%1');exit;" -nodesktop -nosplash
PAUSE
IF EXIST P.EPS COPY P.eps %1.eps
GOTO END
:E1
ECHO FILE %1.m DOES NOT EXIST
GOTO END
:P
IF NOT EXIST P.m GOTO E2
IF EXIST PP.m COPY PP.m PPP.m
COPY P.m PP.m
CALL matlab -r "run('P');exit;" -nodesktop -nosplash
GOTO END
:E2
ECHO FILE P.m DOES NOT EXIST
GOTO END
:I
CALL matlab -r "run('P');display('Press ENTER to exit');pause;exit;" -nodesktop -nosplash
:END
```

If the SCILAB system is installed, $GRAPHICS = –3 and $GRAPH='E' (Section 5.6), we can use the statement

```
SCIGO file_name
```

to start the SCILAB graphics environment for drawing the final results. Here `file_name.sci` is the input file for SCILAB. If `SCIGO` statement does not contain a file name specification, then the standard input file is `P.sci`. The `SCIGO` statement is realized by the procedure `SCIGO.BAT`, which has the form

```
@ECHO OFF
IF "%1"=="" GOTO P
IF "%1"=="@" GOTO I
IF NOT EXIST %1.sci GOTO E1
CALL scilex -e "exec('%1.sci');exit;" -nw
PAUSE
IF EXIST P.EPS COPY P.eps %1.eps
GOTO END
:E1
ECHO FILE %1.sci DOES NOT EXIST
GOTO END
```

```
:P
IF NOT EXIST P.sci GOTO E2
IF EXIST PP.sci COPY PP.sci PPP.sci
COPY P.sci PP.sci
CALL scilex -e "exec('P.sci');exit;" -nw
GOTO END
:E2
ECHO FILE P.sci DOES NOT EXIST
GOTO END
:I
START scilex -e "exec('P.sci');halt('Press ENTER to exit');exit;" -nw
:END
```

1.6 Internal procedures and output files

In addition to the described procedures, the UFO system uses further internal procedures. The list of internal procedures follows (x is one of characters 6, 8, N, G for versions UFOW6, UFOW8, UFOWN, UFOWG):

`UFOx.BAT` - Starting the UFO system.
`SETx.BAT` - Setting the paths and parameters.
`UFO.BAT` - Starting the UFO environment.
`EDIT.BAT` - Starting the UFO editor.
`GENER.BAT` - Generation of the UFO source program.
`COMPIL.BAT` - Compilation of the UFO source program followed by the computation.
`COMPIL1.BAT` - Compilation of the UFO source program with debugging options followed by the computation.
`UFOGO.BAT` - Generation and compilation of the UFO source program followed by the computation.
`UFOGO1.BAT` - Generation and compilation of the UFO source program with debugging options followed by the computation.
`MATGO.BAT` - Starting the MATLAB graphics environment (Section 5.5).
`SCIGO.BAT` - Starting the SCILAB graphics environment (Section 5.6).
`PROFILE.BAT` - Computation of the performance profile (Section 6.4).
`PROFMAT.BAT` - Computation of the performance profile and its drawing using the MATLAB graphics (Section 6.4).
`PROFSCI.BAT` - Computation of the performance profile and its drawing using the SCILAB graphics (Section 6.4).
`PROFTEX.BAT` - Computation of the performance profile and its drawing using the LATEX graphics (Section 6.4).

In addition to the described files, the UFO system produces additional files which contain some useful information. A list of the most important UFO files follows:

`P.UFO` - Input template (every input file *.UFO is preliminary copied to P.UFO).
`P.TMP` - Temporary file containing a source program ancestor generated in the first pass of the UFO preprocessor (Section 1.1).
`P.FOR` - The main Fortran routine (UFO problem solver), for versions UFOW6, UFOW8, generated in the second pass of the UFO preprocessor (Section 1.1).
`P.F` - The main Fortran routine (UFO problem solver), for versions UFOWN, UFOWG, UFOLG, generated in the second pass of the UFO preprocessor (Section 1.1).
`P.OUT` - Text output file (Section 5.7).
`P.DAT` - Stored values of problem variables (Section 5.9).
`P.DUM` - Dummy screen output if the graphic output is required.
`P.DIM` - Dimensions of basic problem vectors and matrices (Section 5.10).

`P.SIF` - Messages of the SIF decoder (Section 6.6).
`P.I` - Template generated by the SIF decoder (Section 6.6).
`P.PAT` - Sparsity patterns of sparse Hessian and Jacobian matrices (Section 6.5).
`P.PER` - Input data for preparing performance profiles (Section 6.4).
`P.PRO` - Text file containing performance profiles (Section 6.4).
`P.m` - Input data for the MATLAB graphics environment (Sections 5.5 and 6.4).
`P.sci` - Input data for the SCILAB graphics environment (Sections 5.6 and 6.4).
`P.tex` - Latex picture for drawing performance profiles (Section 6.4).

If the specification $GRAPHICS = –1 is used, the UFO system generates additional input files:

`P.PAR` - Data containing problem and method parameters.
`P.GRF` - Data containing values of variables, approximating functions and constraints.
`P.DIF` - Data for drawing solutions of ordinary differential equations.
`P.SUR` - Data for drawing surfaces.
`P.PTH` - Data for drawing paths obtained by optimization methods.

1.7 Brief suggestions for the UFO users

If we want to solve a particular optimization problem, the best way to understand the UFO system is to find a similar problem in the list of sample problems (Chapter 7). This sample problem can be solved by typing `UFOGO PROBxx` (xx is the number of the sample problem). After solving the sample problem, we can modify the batch input file `PROBxx.UFO` to describe our problem. The following list refers to basic problems solved by the UFO system and to sections containing more detailed description.

Minimization of a general objective function: Sections 2.2 – 2.3.
Minimization of a linear objective function: Section 2.2.
Minimization of a quadratic objective function: Section 2.2.
Minimization of a partially separable function (the sum of approximating functions): Sections 2.4 – 2.6.
Minimization of the sum of squares (or powers) of approximating functions: Sections 2.4 – 2.6.
Minimization of the maximum of approximating functions: Sections 2.4 – 2.6.
Minimization of the sum of absolute values of approximating functions: Sections 2.4 – 2.6.
Optimization of systems described by ordinary differential equations: Sections 2.7 – 2.11.
Solution of systems of nonlinear functional equations: Section 2.16.
Solution of systems of ordinary differential equations: Section 2.17.

Moreover, all optimization problems can contain box constraints (Section 2.1) and general linear or nonlinear constraints (Sections 2.12 – 2.14).

The standard batch input file `*.UFO` usually contains substitutions serving for the problem description and directives influencing the control program generation and the method selection. The input data setting and output data processing are realized using the substitutions

```
$SET(INPUT)                    $SET(OUTPUT)
   input data setting             output data processing
$ENDSET                        $ENDSET
```

In macrovariable $INPUT, we specify statements or subroutines defining problem sizes, starting points, types of constraints, right hand sides of equalities and inequalities etc. In macrovariable $OUTPUT, we specify statements or subroutines for output data printing and exploiting. A similar way is used for defining problem functions. The model function can be specified using the substitutions

```
$SET(FMODELF)      $SET(GMODELF)      $SET(HMODELF)
   function value     gradient           Hessian matrix
$ENDSET            $ENDSET            $ENDSET
```

One can also specify the model function value and the model gradient simultaneously by the macrovariable $FGMODELF (or all the quantities by the macrovariable $FGHMODELF). Similarly, for defining the approximating function we use macrovariables $FMODELA, $GMODELA, $HMODELA, $FGMODELA, $FGHMODELA, and quantities concerning the constraint function are specified using macrovariables $FMODELC, $GMODELC, $HMODELC, $FGMODELC, $FGHMODELC. The external subroutines can be added to the UFO system using the substitution

```
$SET(SUBROUTINES)
   user supplied subroutines
$ENDSET
```

More information is given in Section 2 and Section 4.

Even if the problem sizes can be submitted to the UFO system using the macrovariable $INPUT, the sizes serving for declarations of arrays in the UFO control program have to be specified by the following individual substitutions

$NF =	number of variables,
$NA =	number of approximating functions,
$NAL =	number of linear approximating functions,
$NC =	number of constraints,
$NCL =	number of linear constraints.

Similarly, other important sizes, described in Section 2, can be specified in the same way.

Optimization methods need not be selected by the user, the system automatically chooses a suitable method. On the other hand, if the user is familiar with optimization methods, then the method can be selected by using suggestions given in Section 3. The most important data for the method selection can be given using the following macrovariables

$FORM	- Form of the method (sequential quadratic programming, interior point, etc).
$CLASS	- Class of the method (conjugate gradient, limited memory, variable metric, Newton, etc).
$TYPE	- Type of the method (line search, trust region, etc).
$DECOMP	- Matrix decomposition used (original matrix, Choleski decomposition, inversion, etc).
$NUMBER	- Number of the method (choice of the individual method for direction determination).
$SEARCH	- Choice of the line search method.
$UPDATE	- Choice of the variable metric update.

Further details concerning the optimization method can be specified by additional macrovariables as it is shown in Section 3.

Finally, the specifications concerning output possibilities can be selected using macrovariables $GRAPH, $MAP, $ISO, $HIL, $PATH, $DISPLAY (see Section 5). Standard form of the UFO control program (one method for one problem) is stated using statement $STANDARD and the batch mode is indicated by statement $BATCH.

1.8 Conventional names of basic variables and arrays

Basic variables of optimization problems (sizes, indices, function values) require the use of the following prescribed names.

NF	- Number of variables.
NX	- Number of box constraints (the index of the last box constrained variable).
FF	- Value of the objective function.
M	- Number of nonzero elements of sparse Hessian matrix HF (dimension of array $HF).
NA	- Number of approximating functions.
KA	- Index of the approximating function.

FA - Value of the KA-th approximating function.
MA - Number of nonzero elements of sparse Jacobian matrix AG (dimension of array $AG).
MHA - Number of nonzero elements of sparse Hessian matrix HA (dimension of array $HA).
MAH - Dimension of array $AH.
NC - Number of constraint functions.
KC - Index of the constraint function.
FC - Value of the KC-th constraint function.
MC - Number of nonzero elements of sparse Jacobian matrix CG (dimension of array $CG).
MHC - Number of nonzero elements of sparse Hessian matrix HC (dimension of array $HC).
MCH - Dimension of array $CH.
NE - Number of state variables and state functions (number of differential equations).
KE - Index of the state function.
FE - Value of the KE-th state function.

Names of arrays in optimization problems are determined by the following macrovariables.

$X - Vector of variables.
$XN - Scaling coefficients.
$XL - Lower bounds of variables.
$XU - Upper bounds of variables.
$IX - Types of box constraints.
$GF - Gradient of the objective function.
$HF - Hessian matrix of the objective function (only the upper half is stored).
$IH - Pointers of diagonal elements of sparse matrix HF.
$JH - Column indices of nonzero elements of sparse matrix HF.
$AF - Vector containing values of all approximating functions.
$GA - Gradient of the KA-th approximating function.
$AG - Jacobian matrix containing gradients of all approximating functions.
$IAG - Pointers of row-start elements of sparse matrix AG.
$JAG - Column indices of nonzero elements of sparse matrix AG.
$HA - Hessian matrix of the KA-th approximating function (only the upper half is stored).
$AH - Tensor containing Hessian matrices of all approximating functions.
$AM - Vector containing fitted values of all approximating functions.
$AW - Vector containing weights of all approximating functions.
$CF - Vector containing values of all constraint functions.
$GC - Gradient of the KC-th constraint function.
$CG - Jacobian matrix containing gradients of all constraint functions.
$ICG - Pointers of row-start elements of sparse matrix CG.
$JCG - Column indices of nonzero elements of sparse matrix CG.
$HC - Hessian matrix of the KC-th constraint function (only the upper half is stored).
$CH - Tensor containing Hessian matrices of all constraint functions.
$CL - Lower bounds of constraint functions.
$CU - Upper bounds of constraint functions.
$IC - Types of general constraints.
$CW - Vector containing weights of all constraint functions.
$EF - Vector containing values of all state functions.
$GE - Gradient of the KE-th state function.
$EG - Jacobian matrix containing gradients of all state functions.
$DE - Matrix containing derivatives of the KE-th state function with respect to state variables.
$ED - Tensor containing derivatives of all state functions with respect to state variables.

The name following the symbol $ is a default name. This default name can be changed by the user using the statement $NAME1='NAME2' in the macrovariable $INPUT. However, the same names cannot be

used for different objects, i.e., NAME2 cannot appear in the above list of default names. For instance, the example demonstrated in Section 1.4 can be rewritten in the form

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

1.9 Authors of the UFO system

The UFO system has been developed in the Institute of Computer Science, Czech Academy of Sciences, Prague. In addition, the UFO system contains two direct solvers for sparse linear systems based on UMFPACK code of T.A.Davis [85] and MA27 code of I.S.Duff [99]. Moreover, some modules for solving ordinary differential equations are based on subroutines proposed in the book of E.Hairer, S.P.Norsett and G.Wanner [137] (they were considerably rewritten).

The following table contains the names of individual authors, their letter codes introduced in source modules and their contribution in percent.

Author	code	contribution
L.Lukšan	LU	71.42
M.Tůma	TU	8.60
C.Matonoha	MA	5.64
J.Vlček	VL	4.05
N.Ramešová	RA	3.48
M.Šiška	SI	3.35
J.Hartman	HA	1.45
T.A.Davis	DA	1.49
I.S.Duff	DU	0.93

2 Problems solved using the UFO system

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$MODEL='AA' - Sum of absolute values minimization. In this case

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

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

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

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

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

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

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

and

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

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

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

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

and

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

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

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

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

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

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

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

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

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

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

The default option is $HESF='D'. The approximating functions $f_k^A : R^n \to R,\ 1 \le k \le n_A$, can have

several types of Jacobian matrices specified by the macrovariable $JACA:

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

If the approximating functions are used then we can choose several types of the Hessian matrix 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='N' - Hessian matrix is not used.

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

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

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

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

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

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

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

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

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

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

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

Besides the standard constraints described above, we can consider also the complementarity constraints of the form

$$f_k^C(x) \ge c_k^L, \qquad f_l^C(x) \ge c_l^L, \qquad f_k^C(x) f_l^C(x) = 0 \qquad \text{if} \quad (k,l) \in C,$$

where C is a set of corresponding pairs of indices. The constraints $f_k^C(x) \ge c_k^L$ or $f_l^C(x) \ge c_l^L$ can be replaced by box constraints $x_k \ge x_k^L$ or $x_l \ge x_l^L$, respectively. Thus, the box constraints can be included into the complementarity constraints.

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

1. Minimization of dynamical systems is not implemented in the sparse case.
2. Usually the UFO system serves for local optimization. Global optimization can be used only for small-size ($n \le 100$) dense problems that are unconstrained or contain box constraints.

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

In the rest of this report we will use the notation NF, NA, NC instead of n, n_A, n_C and X, F(X), FF(X), GF(X), HF(X), FA(KA;X), GA(KA;X), FC(KC;X), GC(KC;X) instead of x, $F(x)$, $f^F(x)$, $g^F(x)$, $h^F(x)$, $f_k^A(x)$, $g_k^A(x)$, $f_k^C(x)$, $g_k^C(x)$. This notation corresponds to names of variables and fields in the UFO system, which is listed in Section 1.8.

2.1 Specification of variables and the box constraints

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

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

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

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

$KBF=2; $NF=4
$SET(INPUT)
X(1)=x_1
X(2)=x_2; IX(2)=1; XL(2)=x_2^L

X(3)=x_3; IX(3)=3; XL(3)=x_3^L; XU(3)=x_3^U
X(4)=x_4; IX(4)=5
$ENDSET

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

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

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

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

2.2 Specification of the model function (dense problems)

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

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

or

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

Option $IEXT=0 is default.

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

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

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

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

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

$SET(HMODELF)
HF(1) = derivative ∂^2FF(X)/∂X(1)^2
HF(2) = derivative ∂^2FF(X)/∂X(1)∂X(2)
HF(3) = derivative ∂^2FF(X)/∂X(2)^2
HF(4) = derivative ∂^2FF(X)/∂X(1)∂X(3)
HF(5) = derivative ∂^2FF(X)/∂X(2)∂X(3)
HF(6) = derivative ∂^2FF(X)/∂X(3)^2
—
HF(NF∗(NF+1)/2) = derivative ∂^2FF(X)/∂X(NF)^2

(for given values of variables X(I), 1≤I≤NF)
$ENDSET

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

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

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

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

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

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

$KSF=1 - The model function FF(X) is smooth and well-conditioned.
$KSF=2 - The model function FF(X) is smooth but ill-conditioned.
$KSF=3 - The model function FF(X) is nonsmooth.

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

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

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

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

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

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

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

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

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

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

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

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

2.3 Specification of the model function (sparse problems)

The UFO system contains optimization methods which take into account the sparsity pattern of the Hessian matrix HF. This possibility decreases the computational time and storage requirements for large-scale optimization problems. In this case we use the option $HESF='S' which means that the sparsity pattern is specified. All other specifications remain the same as in the case of dense problems. The sparsity pattern of the Hessian matrix is specified by using the macrovariable $INPUT. Two integer vectors IH and JH are used where IH(I), 1≤I≤NF+1, are pointers and JH(K), 1≤K≤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 (let us say twice) since it is used for the declaration of working fields. For example, if we have the Hessian matrix

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

then we have to set:

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

JH(6)=4; JH(7)=3; JH(8)=5; JH(9)=4; JH(10)=5
$ENDADD

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

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

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

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

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

2.4 Objective functions for discrete approximation

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

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

or

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

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

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

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

or

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

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

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

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

or

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

where FA(KA;X), 1≤KA≤NA, are approximating functions.

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

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

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

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

for $KBA=0, or

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

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

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

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

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

$NORMA=0 - No scaling is performed. In this case AW(KA)=1, 1≤KA≤NA.

$NORMA=1 - Scaling parameters are determined automatically so that AW(KA)=|AM(KA)|, 1≤KA≤NA.

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

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

2.5 Specification of the approximating functions (dense problems)

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

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

or

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

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

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

or

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

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

$SET(HMODELA)
HA(1) = derivative ∂^2FA(KA;X)/∂X(1)^2
HA(2) = derivative ∂^2FA(KA;X)/∂X(1)∂X(2)
HA(3) = derivative ∂^2FA(KA;X)/∂X(2)^2
HA(4) = derivative ∂^2FA(KA;X)/∂X(1)∂X(3)
HA(5) = derivative ∂^2FA(KA;X)/∂X(2)∂X(3)
HA(6) = derivative ∂^2FA(KA;X)/∂X(3)^2

—

HA(NF∗(NF+1)/2) = derivative ∂^2FA(KA;X)/∂X(NF)^2
(for a given index KA and given values of variables X(I), 1≤I≤NF)

$ENDSET

or

$SET(HMODELAS)

AH(1) = derivative ∂^2FA(1;X)/∂X(1)^2
AH(2) = derivative ∂^2FA(1;X)/∂X(1)∂X(2)
AH(3) = derivative ∂^2FA(1;X)/∂X(2)^2
AH(4) = derivative ∂^2FA(1;X)/∂X(1)∂X(3)
AH(5) = derivative ∂^2FA(1;X)/∂X(2)∂X(3)
AH(6) = derivative ∂^2FA(1;X)/∂X(3)^2
—
AH(NF∗(NF+1)/2) = derivative ∂^2FA(1;X)/∂X(NF)^2
AH(NF∗(NF+1)/2+1) = derivative ∂^2FA(2;X)/∂X(1)^2
AH(NF∗(NF+1)/2+2) = derivative ∂^2FA(2;X)/∂X(1)∂X(2)
AH(NF∗(NF+1)/2+3) = derivative ∂^2FA(2;X)/∂X(2)^2
—
AH(NA∗NF∗(NF+1)/2) = derivative ∂^2FA(NA;X)/∂X(NF)^2

$ENDSET

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

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

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

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

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

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

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

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

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

The option $KSA=1 is default.

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

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

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

2.6 Specification of the approximating functions (sparse problems)

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

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

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

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

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

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

and the Jacobian matrix

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

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)=4
$ENDADD
```

As in the case of the dense problem, the first derivatives of the approximating functions can be specified by using the macrovariables $GMODELA or $GMODELAS. If $JACA='S', only nonzero elements of the gradients are specified. There are two possibilities distinguished by the macrovariable $ARED (the default value is $ARED='N'). If $ARED='N', indices of nonzero elements remains unchanged. If $ARED='Y', indices of nonzero elements are replaced by indices of elements of reduced gradients. If KA-th approximating function depends on LA variables, we use indices $1, 2, \ldots,$ LA.

For the above example the specifications have the form:

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

if $ARED='N',

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

if $ARED='Y' or

```
$SET(GMODELAS)
      AG(1) = ∂FA(1;X)/∂X(1)
      AG(2) = ∂FA(1;X)/∂X(4)
      AG(3) = ∂FA(2;X)/∂X(2)
```

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

in both cases.

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

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

if $ARED='Y' or

$SET(HMODELAS)
AH(1) = ∂^2FA(1;X)/∂X(1)^2
AH(2) = ∂^2FA(1;X)/∂X(1)∂X(4)
AH(3) = ∂^2FA(1;X)/∂X(4)^2
AH(4) = ∂^2FA(2;X)/∂X(2)^2
AH(5) = ∂^2FA(2;X)/∂X(2)∂X(4)
AH(6) = ∂^2FA(2;X)/∂X(4)^2
AH(7) = ∂^2FA(3;X)/∂X(3)^2
AH(8) = ∂^2FA(4;X)/∂X(1)^2
AH(9) = ∂^2FA(4;X)/∂X(1)∂X(2)
AH(10) = ∂^2FA(4;X)/∂X(2)^2

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.7 Objective functions for optimization of dynamical systems

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

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

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

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

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

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

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

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

At the same time

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

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

In all the above cases, the statement $NE=number_of_differential_equations must be used for the specification of number of differential equations NE. Moreover, values TAMIN and TAMAX have to be specified by using the macrovariable $INPUT.

$ADD(INPUT)
TA = initial (minimum) value of the independent variable (TA = TAMIN)
TAMAX = maximum value of the independent variable
$ENDADD

2.8 Specification of the state functions

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

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

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

or

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

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

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

or

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

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

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

or

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

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

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

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

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

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

2.9 Specification of the initial functions

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

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

or

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

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

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

or

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

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

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

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

2.10 Specification of the subintegral function

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

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

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

```
$SET(GMODELA)
        GA(1) = derivative ∂FA(X,YA(TA),TA)/∂X(1)
        GA(2) = derivative ∂FA(X,YA(TA),TA)/∂X(2)
```

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

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

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

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

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

2.11 Specification of the terminal function

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

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

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

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

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

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

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

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

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

2.12 Optimization with general constraints

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

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

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

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

$KBC=2; $NC=3
$ADD(INPUT)
IC(1)=1; CL(1)=c_1^L
IC(2)=1; CL(2)=c_2^L

IC(3)=3; CL(3)=c_3^L; CU(3)=c_3^L
$ENDADD

2.13 Specification of the constraint functions (dense problems)

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

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

or

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

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

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

or

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

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

$SET(HMODELC)
HC(1) = derivative ∂^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

or

$SET(HMODELCS)
CH(1) = derivative ∂^2FC(1;X)/∂X(1)^2
CH(2) = derivative ∂^2FC(1;X)/∂X(1)∂X(2)
CH(3) = derivative ∂^2FC(1;X)/∂X(2)^2
CH(4) = derivative ∂^2FC(1;X)/∂X(1)∂X(3)
CH(5) = derivative ∂^2FC(1;X)/∂X(2)∂X(3)
CH(6) = derivative ∂^2FC(1;X)/∂X(3)^2
—
CH(NF∗(NF+1)/2) = derivative ∂^2FC(1;X)/∂X(NF)^2
CH(NF∗(NF+1)/2+1) = derivative ∂^2FC(2;X)/∂X(1)^2
CH(NF∗(NF+1)/2+2) = derivative ∂^2FC(2;X)/∂X(1)∂X(2)
CH(NF∗(NF+1)/2+3) = derivative ∂^2FC(2;X)/∂X(2)^2
—
CH(NC∗NF∗(NF+1)/2) = derivative ∂^2FC(NC;X)/∂X(NF)^2
$ENDSET

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

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

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

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

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

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

$KSC=1 - The constraint functions FC(KC;X), 1≤KC≤NC, are smooth and well-conditioned.

$KSC=2 - The constraint functions FC(KC;X), 1≤KC≤NC, are smooth but ill-conditioned.
$KSC=3 - The constraint functions FC(KC;X), 1≤KC≤NC, are nonsmooth.

The option $KSC=1 is default.

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

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

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≤KC≤NCL, 1≤I≤NF, are specified by using the macrovariable $INPUT and the macrovariables $FMODELC or $FMODELCS, $GMODELC or $GMODELCS, $HMODELC or $HMODELCS are used only for the specification of the nonlinear constraint functions FC(KC;X), NCL<KC≤NC.

2.14 Specification of the constraint functions (sparse problems)

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

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

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

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

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

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

and the Jacobian matrix

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

then we have to set:

$NC=5
$MC=20 (the minimum required value is MC=10)
$ADD(INPUT)

```
        ICG(1)=1; ICG(2)=3; ICG(3)=5
        ICG(4)=6; ICG(5)=9; ICG(6)=11
        JCG(1)=1; JCG(2)=4; JCG(3)=2; JCG(4)=4; JCG(5)=3
        JCG(6)=1; JCG(7)=2; JCG(8)=3; JCG(9)=3; JCG(10)=4
    $ENDADD
```

As in the case of the dense problem, the first derivatives of the constraint functions can be specified by using the macrovariables $GMODELC or $GMODELCS. If $JACC='S', only nonzero elements of the gradients are specified. There are two possibilities distinguished by the macrovariable $CRED (the default value is $CRED='N'). If $CRED='N', indices of nonzero elements remains unchanged. If $CRED='Y', indices of nonzero elements are replaced by indices of elements of reduced gradients. If KC-th approximating function depends on LC variables, we use indices $1, 2, \ldots, LC$.

For the above example the specifications have the form:

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

if $CRED='N',

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

if $CRED='Y' or

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

in both cases.

As in the case of the dense problem, the second derivatives of the constraint functions can be specified by using the macrovariables $HMODELC or $HMODELCS. If $JACC='S', only nonzero elements of the Hessian matrices are specified. The use of macrovariable $HMODELC is allowed only if $CRED='Y'. For the above example the specifications have the form:

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

if $CRED='Y' or

```
$SET(HMODELCS)
        CH(1) = ∂²FC(1;X)/∂X(1)²
        CH(2) = ∂²FC(1;X)/∂X(1)∂X(4)
        CH(3) = ∂²FC(1;X)/∂X(4)²
        CH(4) = ∂²FC(2;X)/∂X(2)²
        CH(5) = ∂²FC(2;X)/∂X(2)∂X(4)
```

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

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

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

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

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

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

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

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

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

CALL $UKMCI2(K,I,GCKI),

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

2.15 Specification of complementarity constraints

Complementarity constraints can be written in the following form:

$$\begin{aligned} &\text{FC(ICC(I),X)} \geq \text{CL(ICC(I))}, \quad \text{FC(ICC(I+NCC),X)} \geq \text{CL(ICC(I+NCC))}, \\ &\text{FC(ICC(I),X) * FC(ICC(I+NCC),X)} = 0, \qquad \text{if ICC(I)} > 0 \text{ and ICC(I+NCC)} > 0, \\ \\ &\text{X}(-\text{ICC(I)}) \geq \text{XL}(-\text{ICC(I)}), \quad \text{FC(ICC(I+NCC),X)} \geq \text{CL(ICC(I+NCC))}, \\ &\text{X}(-\text{ICC(I)}) * \text{FC(ICC(I+NCC),X)} = 0, \qquad \text{if ICC(I)} < 0 \text{ and ICC(I+NCC)} > 0, \\ \\ &\text{FC(ICC(I),X)} \geq \text{CL(ICC(I))}, \quad \text{X}(-\text{ICC(I+NCC)}) \geq \text{XL}(-\text{ICC(I+NCC)}), \\ &\text{FC(ICC(I),X)} * \text{X}(-\text{ICC(I+NCC)}) = 0, \qquad \text{if ICC(I)} > 0 \text{ and ICC(I+NCC)} < 0, \\ \\ &\text{X}(-\text{ICC(I)}) \geq \text{XL}(-\text{ICC(I)}), \quad \text{X}(-\text{ICC(I+NCC)}) \geq \text{XL}(-\text{ICC(I+NCC)}), \\ &\text{X}(-\text{ICC(I)}) * \text{X}(-\text{ICC(I+NCC)}) = 0, \qquad \text{if ICC(I)} < 0 \text{ and ICC(I+NCC)} < 0, \end{aligned}$$

where $0 \leq$ I $\leq$ NCC (NCC is the number of complementarity constraints), and ICC is the array with $2 *$ NCC elements containing indices of inequality constraints (index lower than zero correspond to a box constraint and index greater than zero correspond to a general constraint. The constraint functions are specified in the same way as in subsections 2.12–2.14. Note that only one-sided constraints can be used in the complementarity case.

2.16 Solution of systems of nonlinear functional equations

If we set $MODEL='NE', then we suppose that the system of nonlinear functional equations

$$\text{FA(KA; X)} = 0, \qquad 1 \leq \text{KA} \leq \text{NF}$$

is solved. This possibility is equivalent to minimization of sum of squares when $MODEL='AQ', but now NA=NF so that special methods for solving systems of nonlinear functional equations can be used. Functions FA(KA;X), $1 \leq$ KA $\leq$ NF, are in fact approximating functions. Their specification is described in Sections 2.5 – 2.6.

2.17 Solution of systems of ordinary differential equations

If we set $MODEL='DE', then we suppose that the system of ordinary differential equations

$$\frac{d\text{YA(KE; TA)}}{d\text{TA}} = \text{FE(KE; YA(TA), TA)}, \qquad \text{YA(KE; TAMIN)} = \text{FY(KE)} \qquad 1 \leq \text{KE} \leq \text{NE}$$

is solved in the interval TAMIN $\leq$ TA $\leq$ TAMAX, where FE(KE;YA(TA),TA), 1$\leq$KE$\leq$NE, are smooth state functions and FY(KE), 1$\leq$KE$\leq$NE, are smooth initial functions. In this case, the statement $NE=number_of_differential_equations must be used for the specification of number of differential equations NE. Moreover, values TAMIN and TAMAX have to be specified by using the macrovariable $INPUT.

$ADD(INPUT)
  TA = initial (minimum) value of the independent variable (TA = TAMIN)
  TAMAX = maximum value of the independent variable
$ENDADD

Specification of state functions is described in Section 2.8. Specification of initial functions is described in Section 2.9. The resulting state variables (solution of system of ordinary differential equations) can be stored in all integration steps (if $MED=1) or in prescribed mesh points (if $MED=2). In the second case, the macrovariable $NA specifies the number of mesh points equidistantly distributed in the interval TAMIN $\leq$ TA $\leq$ TAMAX.

2.18 Additional specifications concerning optimization problems

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

If there are no general constraints and if the number of variables is not greater than 100, then we can use global optimization methods. A decision between local and global optimization is effected by means of macrovariable $EXTREM:

$EXTREM='L' - A local extremum is found, which usually contains the starting point in its basin of attraction.
$EXTREM='G' - All extremum points 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 macrovariables $FGMODELF and $FGHMODELF for a common specification of the value, the gradient and the Hessian matrix of the model function. Similarly we cannot use the macrovariables $FGMODELA or $FGMODELAS and $FGHMODELA or $FGHMODELAS for a common specification of the approximating functions.

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

$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

$MIT - maximum number of iterations
$MFV - maximum number of function evaluations
$MFG - maximum number of gradient 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 $MIT=500, $MFV=1000, $MFG=10000.

If a direct solver ($NUMBER=1) is used for direction determination in methods for sparse constrained problems, the statement $MMAX=space_for_sparse_factor should be given. The default value $MMAX=M can be insufficient in these cases.

3 Optimization methods in the UFO system

The UFO system has a modular structure. All optimization methods can be set up using the individual simple modules. For example, the sequential quadratic programming variable metric methods for nonlinearly constrained optimization problems are set up by using the modules for an objective function evaluation, penalty function definition, direction determination, quadratic programming solution, stepsize selection, and variable metric update. The optimization methods contained in the UFO system can be roughly divided into two groups. The first group contains methods for unconstrained and linearly constrained optimization problems, while the second group contains methods for general nonlinear programming problems (or problems which are transformed to general nonlinear programming problems, e.g. l_1, l_∞ and minimax 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='SL'	- Recursive linear or quadratic programming methods for dense unconstrained minimax problems.
$FORM='SQ'	- Recursive quadratic programming methods for dense nonlinear programming problems.
$FORM='SE'	- Recursive quadratic programming methods for sparse equality constrained problems.
$FORM='SI'	- Primal-dual interior point methods for sparse nonlinear programming problems.
$FORM='SF'	- Nonsmooth equation methods for sparse nonlinear programming problems.
$FORM='SP'	- Primal interior point methods for sparse nonlinear programming problems, sparse unconstrained minimax problems and sparse unconstrained l_∞ or l_1 approximation problems.
$FORM='SM'	- Smoothing methods for sparse unconstrained minimax problems and sparse unconstrained l_∞ or l_1 approximation problems.

Sections 3.1 – 3.24 concern methods for unconstrained and linearly constrained problems. These methods do not use the macrovariable $FORM for the classification. Sections 3.25 – 3.30 are devoted to minimization of unconstrained or linearly constrained structured functions, especially to l_1 and l_∞ approximation. These problems can be transformed to general nonlinear programming problems, but also special methods can be used. For dense l_1 approximation, only the choice $FORM='SQ' is possible. For sparse l_1 approximation, the choices $FORM='SP', $FORM='SM', $FORM='SI', $FORM='SF' can be used. For dense l_∞ approximation, the choices $FORM='SL' $FORM='SQ' are possible. For sparse l_∞ approximation, the choices $FORM='SP', $FORM='SM', $FORM='SI', $FORM='SF' can be used. Methods for general nonlinear programming problems are described in Sections 3.31 – 3.35. For dense nonlinear programming problems, only the choice $FORM='SQ' is possible. For sparse nonlinear programming problems, the choices $FORM='SE', $FORM='SI', $FORM='SF' can be used. The basic parts of optimization methods are described in Sections 3.36 – 3.40. Section 3.41 is devoted to global optimization methods.

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

$CLASS='HM'	- Heuristic methods for small-size problems. This class contains the pattern search method and the simplex method.
$CLASS='CD'	- Conjugate direction methods which use no matrices. This class contains conjugate gradient methods and variable metric methods with limited memory based on the Strang recursions.
$CLASS='VL'	- Variable metric methods with limited memory based on compact matrix updates.
$CLASS='VS'	- Variable metric methods with limited memory based on shifted product-form updates.
$CLASS='VP'	- Variable metric methods with limited memory based on projected product-form updates.
$CLASS='VR'	- Variable metric methods with limited memory based on reduced Hessians.

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

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

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

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

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

$CLASS='QL' - Quasi-Newton methods with limited memory for large-scale nonlinear least squares problems or large-scale nonlinear equations.

$CLASS='QB' - Simple quasi-Newton methods or Brent methods for nonlinear equations.

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

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

$CLASS='BV' - Bundle variable metric methods for nonsmooth optimization.

$CLASS='BL' - Bundle limited-memory variable metric methods for large-scale nonsmooth optimization.

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

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

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

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

Individual methods from the above classes can be chosen by using additional specifications. The most important ones, concerning direction determination and stepsize selection, are the type of the method, the kind of the matrix decomposition, the serial number of the method, and the way of the stepsize selection. For unconstrained optimization, these specifications are described in Sections 3.37–3.38. For constrained optimization, more information can be found in Sections 3.31–3.35.

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='R' - Cubic regularization methods.

$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 original 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 [123].

$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 [304].

$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 [34] or the Bunch-Kaufmann algorithms [35].

$DECOMP='I' - The inverse of a symmetric matrix is used as an input for the direction determination.

$DECOMP='P' - The product form SS^T of inverse is used as an input for the direction determination.

$DECOMP='R' - The $R^T R$ decomposition without permutations is used as an input for the direction determination. This decomposition is usually obtained by the recursive QR factorization [167].

$DECOMP='A' - The rectangular matrix is used as an input for the direction determination.

$DECOMP='Q' - The QR decomposition of a rectangular matrix without permutations is used as an input for the direction determination. This decomposition is usually obtained by using the Householder reflection with the explicitly stored orthogonal matrix Q.

$DECOMP='E' - The general square matrix is used as an input for the direction determination in the case NA=NF (system of nonlinear equations).

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

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

$DECOMP='Z' - The null space representation based on orthogonal projection is used as an input for the direction determination.

$DECOMP='G' - The range space representation based on the Schur complement is used as an input for the direction determination.

If $FORM='SI', then the following possibility for a representation of matrices in the direction determination is used implicitly:

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

If $FORM='SF', then the following possibility for a representation of matrices in the direction determination is used implicitly:

$DECOMP='F' - The nonsmooth equation Karush-Kuhn-Tucker matrix is used as an input for the direction determination.

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 allowed kinds of the matrix decomposition, owed to individual classes of optimization methods, are listed in the corresponding sections.

The serial number of the method is specified by the macrovariable $NUMBER. This option determines an individual realization of the direction determination method. In almost all cases, the selected value specifies the method for solving systems of linear equations in either line search or trust region frameworks. The values $NUMBER=1,2,3,4,5,6,7,8,9 can be used as is described in Section 3.37. If $CLASS='CD', $CLASS='VL', $CLASS='VR', the macrovariable $NUMBER has a different meaning, which is described in sections 3.2, 3.3, 3.4, 3.7. If $CLASS='VS' or $CLASS='VP', the implicit value $NUMBER=1 is used.

The way of the stepsize selection is specified by the macrovariable $SEARCH:

$SEARCH='B' - Basic line search methods based on various interpolation and extrapolation formulas (if $TYPE='L') or basic trust region methods with stepsize control based on the comparison of the actual and the predicted function decreases (if $TYPE='G').

$SEARCH='M' - Mixed line search methods which control the maximum stepsize like the trust region methods (if $TYPE='L') or mixed trust region methods which use interpolation formulas for stepsize reduction like the line search methods (if $TYPE='G').

$SEARCH='N' - Special nonmonotone line search methods.

$SEARCH='H' - Special line search methods described in [141], which are suitable for conjugate gradient methods.

Note that the choices $SEARCH='B' and $KTERS<0 specify standard nonmonotone line search methods introduced in [135]. The absolute value of $KTERS gives the number of nonmonotone steps (see Section

3.38).

If the variable metric or quasi-Newton methods are used, the individual update is specified by the macrovariables $UPTYPE and $UPDATE as is described in corresponding sections.

All options used for the method selection have default values which follow from the knowledge bases coded in the individual templates. Therefore, they need not be necessarily specified by the user. This fact especially concerns macrovariables $METx, $MOTx, $MESx, $MOSx, where x is the empty symbol or one of digits 1–5. The possibilities we describe in this section can be of service to users who are familiar with optimization methods.

Almost all optimization methods have different realizations for the three different representations of the objective function. If $HESF='D', dense variants can be used for either unconstrained problems or box constrained problems or linearly constrained problems (with dense linear constraints specified by $JACC='D'). If $HESF='S', sparse variants can be used for either unconstrained problems or box constrained problems or linearly constrained problems (with sparse linear constraints specified by $JACC='S'). Moreover, the option $HESF='B' can be used for partially separable problems.

If $MODEL='AF' or $MODEL='AQ' or $MODEL='AP', the objective function is partially separable and its Hessian matrix has a partitioned pattern. If $JACA='S' and $HESF='S', the original matrix with a partitioned pattern is transformed to the sparse matrix with a general pattern, which is then used for direction determination by using either direct or iterative solution methods. If $JACA='S' and $HESF='B', the original matrix with a partitioned pattern is immediately used for direction determination by using iterative solution methods (direct solution methods cannot be utilized in this case). Partitioned variants of optimization methods are usually less efficient due to the more expensive matrix operations. Therefore, we recommend preferring sparse variants ($HESF='S') to the partitioned ones ($HESF='B').

3.1 Heuristic methods

Heuristic (or comparative) methods are specified by the statement $CLASS='HM' and are generated from the driver template U0FDU1. These methods should be used only for small-size problems (with 10 variables at most). The main advantage of 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 [152].
$TYPE='S' - Simplex method of Nelder and Mead [270].

The default value is $TYPE='P'.

3.2 Nonlinear conjugate gradient methods

Nonlinear conjugate gradient methods are specified by the statements $CLASS='CD' and $DECOMP='C' and are generated from the driver template U1FLU1. These methods use only few vectors and are very efficient for large problems with computationally simple objective functions ($KCF=1 or $KCA=1). The main advantage of conjugate gradient methods is that no matrices are used (implicitly $HESF='N'). This fact highly decreases storage requirements.

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

$NUMBER=1 - Basic conjugate gradient methods described in [193].
$NUMBER=2 - Generalized conjugate gradient methods introduced in [176].

If $NUMBER=1, the individual methods and their modifications are specified by using the macrovariables $MET, $MET1, $MET2, $MET3, $MET4, $MET5 and $MOS1.

Macrovariable $MET determines the type of conjugate gradient method:

$MET=0 - The steepest descent method is used.

$MET=±1 - The Hestenes-Stiefel method [149] is used.
$MET=±2 - The Polak-Ribiere method [281] is used.
$MET=±3 - The Liu-Storey method [176] is used.
$MET=±4 - The Dai-Yuan method [71] is used.
$MET=±5 - The Fletcher-Reeves method [111] is used.
$MET=±6 - The conjugate descent method [107] is used.
$MET=±7 - The Perry version of Hestenes-Stiefel method [149] is used.
$MET=±8 - The Perry version of Polak-Ribiere method [281] is used.
$MET=±9 - The Perry version of Liu-Storey method [176] is used.
$MET=±10 - The Kou and Dai method [165] is used.

If $MET<0, the previous direction vector is used for the determination of conjugate direction parameters. If $MET>0, the vector of variables difference is used in this case. The default value is $MET=1.

Macrovariable $MET1 specifies a modification of the conjugate gradient method.

$MET1=±1 - The standard CG method [67] (basic or combined) is used.
$MET1=±2 - The two term descent CG method [378] (basic or combined) is used.
$MET1=±3 - The improved CG method with scaled gradients [379] is used.
$MET1=±4 - The three term descent CG method [380] is used.
$MET1=±5 - The modified descent CG method [369] is used.
$MET1=±6 - The modified descent CG method with scaled gradients [369] is used.
$MET1=±7 - The modified three term descent CG method [376] is used.
$MET1=±8 - The Dai-Liao CG method [67] is used.
$MET1=±9 - The convex combination [358] of CG methods is used.
$MET1=±10 - The modified CG method [370] is used.

If $MET1>0, the CG method with a general parameter is used. If $MET1<0, the CG method with a nonnegative parameter is used. The default value is $MET1=4 if |$MET|=1,2,3 and $MET1=3 otherwise.

Macrovariable $MET2 specifies some details concerning CG methods. If $MET1=1, $MET1=2, then the basic CG or the combined CG methods are used for $MET2=1 or $MET2=2, respectively. If $MET1=5, $MET1=6, $MET1=7, $MET1=8, then the general or the positive first terms are used for $MET2=1 or $MET2=-1, respectively. If $MET1=9, then the general or the convex combinations of CG methods are used for $MET2=1 or $MET2=-1, respectively.

Macrovariable $MET3 specifies a test on conjugacy or orthogonality described in [193].

$MET3=1 - The basic CG method is used.
$MET3=2 - A test on conjugacy is used.
$MET3=3 - A test on orthogonality is used.

The default value is $MET3=2 if |$MET|=4,5,6 and $MET3=1 otherwise.

Macrovariable $MET4 specifies the scaling parameter as described in [193].

$MET4=1 - No scaling is used.
$MET4=2 - The BFGS scaling in every iteration is used.
$MET4=3 - The DFP scaling in every iteration is used.
$MET4=4 - The Hoshino scaling in every iteration is used.

The default value is $MET4=1.

Macrovariable $MET5 specifies a nonquadratic correction.

$MET5=1 - No correction is used.
$MET5=2 - The backward Taylor expansion with 3 terms [317] is used.
$MET5=3 - The backward Taylor expansion with 4 terms [197] is used.
$MET5=4 - The value determined from the homogeneous model [197] is used.

$MET5=5 - The Wei nonquadratic model [171] is used.
$MET5=6 - The Deng nonquadratic model [374] is used.

Choices $MET5=2, $MET5=3, $MET5=4 are possible only if $MET1=±8 (Dai-Liao). The default value is $MET5=1.

Macrovariable $MOS1 specifies a periodic restart.

$MOS1=1 - Periodic restart after n iterations is used.
$MOS1=2 - Periodic restart after $2\,n$ iterations is used.
$MOS1=3 - Periodic restart after $12\,n$ iterations is used.

The default value is $MOS1=3.

If $NUMBER=2, then macrovariable $MET1 specifies the restart strategy as is described in [193].

$MET1=1 - The basic CG method is used.
$MET1=2 - The CG method with a positive parameter is used.
$MET1=3 - The CG method with a bounded positive parameter is used.
$MET1=4 - The CG method with a bounded positive parameter and the Storey restart is used.
$MET1=5 - The modified CG method with a bounded and restricted parameter is used.
$MET1=6 - The CG method with the Powell restart is used.

The default value is $MET1=4.

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

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

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

3.3 Variable metric methods with limited memory based on vector recurrences

Variable metric methods with limited memory based on vector recurrences are specified by the statements $CLASS='CD' and $DECOMP='V' and are generated from the driver template U1FLU1. These methods use several small-size matrices which are updated in every iteration in such a way that their product approximates the Hessian matrix as precisely as possible. Starting with the scaled unit matrix, every iteration contains a small number of variable metric updates utilizing previous differences of gradients. The number of these VM steps is specified by the macrovariable $MF. This fact highly decreases storage requirements.

he following variable metric methods with limited memory based on the Strang recurrences [249], corresponding to the choice $DECOMP='V', are implemented in the UFO system:

$NUMBER=1 - The BFGS method with limited memory described in [272]. The default number of VM steps is $MF=5.
$NUMBER=2 - The extended BFGS method with limited memory described in [157]. The default number of VM steps is $MF=3.
$NUMBER=3 - Modified methods with limited memory transformed to the BFGS form described in [349] – a general version. The default number of VM steps is $MF=5.
$NUMBER=4 - Modified methods with limited memory transformed to the BFGS form described in [349] – a simplified version. The default number of VM steps is $MF=5.
$NUMBER=5 - Modified methods with limited memory that utilize vectors from the previous iteration described in [350]. The default number of VM steps is $MF=5.

$NUMBER=6 - Modified methods with limited memory that construct conjugate directions described in [351]. The default number of VM steps is $MF=5.

$NUMBER=7 - Modified methods with limited memory that use corrections described in [352]. The default number of VM steps is $MF=5.

$NUMBER=8 - Block methods with limited memory described in [353]. The default number of VM steps is $MF=5.

$NUMBER=9 - The BNS version of the BFGS method with limited memory described in [39]. The default number of VM steps is $MF=5.

If $NUMBER=1, the limited memory method is realized by using various scaling techniques [175] specified by the macrovariable $MET1.

$MET1=1 - The scaling is suppressed.
$MET1=2 - The scalar scaling is used.
$MET1=3 - The diagonal scaling is used.
$MET1=4 - The scalar and diagonal scalings are used simultaneously.

The default value is $MET1=2. If $NUMBER=2, only the scalar scaling is possible, which is specified by the macrovariable $MET1.

$MET1=1 - The scaling is suppressed.
$MET1=2 - The scalar scaling is used.

The default value is $MET1=2. If $NUMBER>2, the scalar scaling is always used. If $NUMBER=7, the macrovariable $MET3 specifies the number of corrections. The possible values are $MET3=1, $MET3=2, $MET3=3, $MET3=4. The default value is $MET3=4.

Possible specifications (type-decomposition-number) for variable metric methods with limited memory based on vector recurrences are these:

L-V-1,
L-V-2,
L-V-3,
L-V-4,
L-V-5,
L-V-6,
L-V-7,
L-V-8,
L-V-9.

The default choice is L-V-1. Variable metric methods with limited memory based on vector recurrences can also be used for problems with sparse linear constraints when $JACC='S'.

3.4 Variable metric methods with limited memory based on compact matrix representations

Variable metric methods with limited memory based on matrix representations are specified by the statement $CLASS='VL' and are generated from the driver template U1FLU2. These methods use several small-size matrices which are updated in every iteration in such a way that their product approximates the Hessian matrix as precisely as possible. Starting with the scaled unit matrix, every iteration contains a small number of variable metric updates utilizing previous differences of gradients. The number of these VM steps is specified by the macrovariable $MF (the default value is $MF=5).

Individual variable metric methods are specified by using the macrovariables $MET, $MET1 and $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS update [29], [105], [127], [306] is used.

$MET=2 - The DFP update [83], [110] is used.
$MET=3 - The Hoshino update [153] is used.
$MET=4 - The safeguarded rank-one update [190] is used.
$MET=5 - The optimally conditioned update [84] is used.
$MET=6 - The rank-one based update [190] from the perfect Broyden subclass is used.
$MET=7 - The variable metric update with the constant VM parameter [194] is used.
$MET=8 - The heuristic well conditioned update [197] is used.

The default value is $MET=1. If $MET=7, then the constant VM parameter is specified by the macrovariable $ETA3. The default value is $ETA3=1. If $MET=1 or $MET=4, then the explicit matrix representation described in [39] is used. If $MET≠1 and $MET≠4, then the recursive matrix representation described in [239] is used.

Macrovariable $MET1 determines the scaling technique.

$MET1=1 - The scaling is suppressed.
$MET1=2 - The scalar scaling is used.

The default value is $MET1=2.

Macrovariable $MET5 determines a way of the method realization.

$MET5=1 - The realization that requires $6mn$ arithmetic operations is used.
$MET5=2 - The realization that requires $4mn$ arithmetic operations is used.

(these realizations differ by the sensitivity to round-off errors). The default value is $MET5=1.

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

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

The default choice is L-I-1.

3.5 Variable metric methods with limited memory based on shifted product-form updates

Variable metric methods with limited memory based on shifted product-form updates are specified by the statement $CLASS='VS' and are generated from the driver template U1FLU4. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=10). Variable metric methods with limited memory based on shifted product-form updates use a rectangular matrix containing $MF columns with $NF elements which is updated in every iteration in such a way that the shifted product of this matrix with its transposition approximates the Hessian matrix [344], [345], [346].

There are two types of shifted product-form updates which are distinguished by using the macrovariable $UPDATE:

$UPDATE='B' - Basic shifted product-form updates [344].
$UPDATE='V' - Variationally derived shifted product-form updates [345].

Individual variable metric methods with limited memory based on shifted product-form updates are specified by using the macrovariables $MET, $MET2, $MET3 and $MET5. If $UPDATE='B', macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

$MET=5 - The variationally derived method [194] from the preconvex part of the Broyden family is used.

$MET=6 - The hybrid globally convergent update described in [344] is used.

The default value is $MET=1.

Macrovariable $MET2 determines the value of the correction parameter.

$MET2=1 - The unit value is used.

$MET2=2 - The balancing value equal to the previous shift quotient is used.

$MET2=3 - The value derived from the gradient differences is used.

$MET2=4 - The geometric mean of the previous values is used.

$MET2=5 - The geometric mean of the previous shift quotient and the ratio $\zeta/(\zeta+\sigma)$ is used.

$MET2=6 - The ratio $\zeta/(\zeta+\sigma)$ is used, where ζ and σ are the old and the new shift parameters.

$MET2=7 - The ratio $\sqrt{\zeta}/(\sqrt{\zeta}+\sqrt{\sigma})$ is used.

Macrovariable $MET3 determines the value of the shift parameter.

$MET3=0 - The shift parameter from the previous iteration is used.

$MET3=1 - The simple choice with the constant shift quotient is used.

$MET3=2 - The choice defined by a quadratic equation is used.

$MET3=3 - The heuristic choice with the shift quotient not greater than half is used.

$MET3=4 - The heuristic choice with the optimally conditioned first iteration (the first form) is used.

$MET3=5 - The heuristic choice with the optimally conditioned first iteration (the second form) is used.

$MET3=6 - The heuristic choice derived from the lowest eigenvalue of the updated matrix is used.

The default value is $MET3=4.

If $UPDATE='B', macrovariable $MET5 determines the individual limited-memory method.

$MET5=1 - The rank-one limited memory method is used.

$MET5=2 - The simple rank-two limited memory method is used.

$MET5=3 - The rank-two limited memory method derived from the shifted Broyden class (method SSBC in [346]) is used.

$MET5=4 - The rank-two limited memory method derived from the shifted Broyden class (based on the minimization of the Frobenius norm) is used.

$MET5=5 - The rank-two limited memory method derived from the shifted Broyden class (method DSBC in [346]) is used.

The default value is $MET5=3.

If $UPDATE='V', macrovariable $MET5 determines the variationally derived limited-memory method.

$MET5=1 - The rank-one limited memory method (method VAR1 in [345]) is used.

$MET5=2 - The simple rank-two limited memory method (method VAR2 in [345]) is used.

$MET5=3 - The general rank-two limited memory method with the constant Broyden parameter η is used (for $\eta=1$ it is the previous choice).

$MET5=4 - The general rank-two limited memory method with the Broyden parameter η derived by the minimum principle is used.

$MET5=5 - The simple rank-two limited memory method with a special choice of matrices (the first variant) is used.

$MET5=6 - The simple rank-two limited memory method with a special choice of matrices (the second variant) is used.

$MET5=7 - The simple rank-two limited memory method with a special choice of matrices (the third variant) is used.

The default value is $MET5=3.

Possible specifications (type-decomposition-number) for variable metric methods with limited memory based on shifted product-form updates are these:

L-I-1.

3.6 Variable metric methods with limited memory based on projected product-form updates

Variable metric methods with limited memory based on projected product-form updates are specified by the statement $CLASS='VP' and are generated from the driver template U1FLU5. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited memory based on projected product-form updates use either one or two rectangular matrices containing $MF columns with $NF elements.

There are two types of projected product-form updates which are distinguished by using the macrovariable $UPDATE:

$UPDATE='C' - One rectangular matrix is used which is updated in every iteration in such a way that the product of this matrix with its transposition satisfies the quasi-Newton condition. This product is augmented by the scaled unit matrix and the resulting matrix is corrected by using a limited number of the Strang recurrences [347].

$UPDATE='P' - Two rectangular matrices are used which are updated in every iteration in such a way that the difference of products of these matrices with their transpositions augmented by the scaled unit matrix satisfies the quasi-Newton condition [348].

If $UPDATE='C', individual variable metric methods with limited memory based on projected product-form updates are specified by using the macrovariables $MET2, $MET3 and $MET5. Macrovariable $MET2 determines a nonquadratic correction.

$MET2=±1 - The unit value is used.
$MET2=±2 - The Spedicato value [317] is used.
$MET2=±3 - The modified Spedicato value [197] is used.
$MET2=±4 - The value determined from the homogeneous model [197] is used.
$MET2=±5 - The value determined from the Bigs cubic model [15] is used.

If $MET2<0, the basic variable metric update is used. If $MET2>0, a modified variable metric update is used. The default value is $MET2=1.

Macrovariable $MET3 determines a variable metric method used for a correction.

$MET3=1 - The variable metric correction with a constant Broyden parameter is used.
$MET3=2 - The variable metric correction with a variable Broyden parameter computed by a special formula is used.

The default value is $MET3=1.

Macrovariable $MET5 determines the number of the Strang recurrences used in the correction phase. This number has to be nonnegative. The default value is $MET5=2.

If $UPDATE='P', no additional specifications are used.

Possible specifications (type-decomposition-number) for variable metric methods with limited memory based on projected product-form updates are these:

L-I-1.

3.7 Variable metric methods with limited memory based on reduced Hessians

Variable metric methods with limited memory based on reduced Hessians are specified by the statement $CLASS='VR' and are generated from the driver template U1FLU3. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=10). Variable metric methods with limited memory based on reduced Hessians use a small-size matrix which is updated in every iteration in such a way that it approximates the reduced Hessian matrix as precisely as possible [121].

There are two families of variable metric methods with limited memory based on reduced Hessians implemented in the UFO system:

$NUMBER=1 - Basic variable metric methods with limited memory based on reduced Hessians described in [121].

$NUMBER=2 - Variable metric methods with limited memory based on reduced Hessians and orthogonal transformations described in [344].

Individual variable metric methods with limited memory based on reduced Hessians are specified by using the macrovariables $MET, $MET1, $MET2, $MET3, $MET4 and $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.
$MET=5 - The optimally conditioned method [84] is used.
$MET=6 - The rank-one based method [190] from the perfect Broyden subclass is used.
$MET=7 - The variationally derived method [194] from the perfect Broyden subclass is used.
$MET=8 - The heuristic method [197] is used.
$MET=9 - The method [381] derived from the matrix decomposition is used.
$MET=10 - The method [382] which minimizes the angle between the direction vector and the negative gradient is used.

The default value is $MET=1.

Macrovariable $MET1 determines the Oren (scaling) parameter [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3.

Macrovariable $MET2 determines the value of the Biggs (nonquadratic model) parameter [14].

$MET2=±1 - The unit value is used.
$MET2=±2 - The Spedicato value [317] is used.
$MET2=±3 - The modified Spedicato value [197] is used.
$MET2=±4 - The value determined from the homogeneous model [197] is used.
$MET2=±5 - The value determined from the Biggs cubic model [15] is used.

Moreover, if $MET2>0, the basic update is used and if $MET2<0, the modified update [291] is used. The default value is $MET2=2.

Macrovariable $MET3 determines the Powell correction [288].

$MET3=1 - The Powell correction is suppressed (the strong update elimination).
$MET3=2 - The Powell correction is suppressed (the weak update elimination).

$MET3=3 - The Powell correction is applied.

The default value is $MET3=1.

Macrovariable $MET4 specifies a rule for the determination of the scaling parameter [219].

$MET4=1 - The BFGS scaling is used.
$MET4=2 - The DFP scaling is used.
$MET4=3 - The geometric mean is used.
$MET4=4 - The harmonic mean is used.
$MET4=5 - The arithmetic mean is used.

The default value depends on the value of macrovariable $MET (values $MET4=1 or $MET4=3 are most frequently used).

Macrovariable $MET5 specifies a type of scaling.

$MET5=1 - The standard scaling is used.
$MET5=2 - The simplified scaling is used.

The default value is $MET5=1.

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

L-R-1 L-I-1,
L-R-2 L-I-2.

The default choice is L-R-1.

3.8 Variable metric methods

Variable metric methods are specified by the statement $CLASS='VM'. These methods use approximations of the Hessian matrix or its inverse, which are updated by rank-one or rank-two correction matrices. Variable metric methods are realized in three different forms for $HESF='D', $HESF='S' and $HESF='B' depending on the Hessian matrix specification. Although the variable metric methods can be realized as trust region methods ($TYPE='G') or cubic regularization methods ($TYPE='R') it is more advantageous to realize them as line search methods ($TYPE='L').

3.8.1 Variable metric methods for problems with dense Hessian matrices

Variable metric methods for problems with dense Hessian matrices are generated from the driver template U1FDU1 if $CLASS='VM' and $HESF='D'. These methods are the most efficient tools for solving small or medium size unconstrained or linearly constrained problems. Variable metric methods for dense problems use a symmetric positive definite matrix which is updated in every iteration in such a way that it approximates the Hessian matrix of the objective function or its inverse as precisely as possible. There are five families of variable metric methods for dense problems which are distinguished by using the macrovariable $UPDATE:

$UPDATE='B' - The Broyden family [29]. Variable metric methods from this family are the most commonly used ones since they are very robust and efficient.

$UPDATE='D' - The Broyden family with Davidon projections [84]. 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.

$UPDATE='L' - The Davidon family described in [178]. Here the Davidon projections are replaced by a special one-parameter update, where the parameter is chosen in such a way to guarantee the positive definiteness of the updated matrix.

$UPDATE='M' - The Broyden family of multistep updates described in [113]. Here the argument and gradient differences are combined with previous ones by using special interpolation techniques.

$UPDATE='S' - The shifted Broyden family [344]. Variable metric methods from this family are efficient without scaling strategies.

If $DECOMP='M' or $DECOMP='P' , only the choice $UPDATE='B' is possible. If $DECOMP='I', choices $UPDATE='B', $UPDATE='M', $UPDATE='S' are possible. In the remaining cases, choices $UPDATE='B', $UPDATE='D', $UPDATE='L' are possible. The default value is $UPDATE='B'.

Individual variable metric methods are specified by using macrovariables $MET, $MET1, $MET2 $MET3 and $MET4. Macrovariable $MET determines the following variable metric updates.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.
$MET=5 - The optimally conditioned method [84] is used.
$MET=6 - The rank-one based method [190] from the perfect Broyden subclass is used.
$MET=7 - The special variationally derived method [194] from the perfect Broyden subclass is used.
$MET=8 - The heuristic well conditioned method [197] is used.
$MET=9 - The method [381] derived from the matrix decomposition is used.
$MET=10 - The method [382] which minimizes the angle between the direction vector and the negative gradient is used.
$MET=11 - The method [197] which minimizes the norm of the direction vector is used.
$MET=12 - The least prior deviation method [254] is used.
$MET=13 - The general variationally derived method [194] from the perfect Broyden subclass is used.
$MET=14 - The determinant preserving method [178] is used.
$MET=15 - The Oren and Spedicato method [278] is used.

The default value is $MET=1. If $DECOMP='M', only values $MET=1,2,3,4 can be used. If $DECOMP='P', only value $MET=1 can be used. If $UPDATE='M', only values $MET=1,2,3,4,5,6,7,8 can be used. If $UPDATE='L' or $UPDATE='S', only values $MET=1,2,3,4,5,6 can be used (if $UPDATE='S', the choice $MET=6 corresponds to the hybrid globally convergent update described in [344]).

Macrovariable $MET1 determines the scaling parameter [277].

$MET1=±1 - No scaling is used.
$MET1=±2 - The initial scaling [310] is used.
$MET1=±3 - The controlled scaling [194] is used.
$MET1=±4 - The interval scaling [221] is used.
$MET1=±5 - The scaling in each iteration is used.

If $MET1>0, the basic initial scaling is used. If $MET1<0, the modified initial scaling is used. The default value is $MET1=4.

If $UPDATE='B' or $UPDATE='D' or $UPDATE='S', macrovariable $MET2 determines the parameter of the nonquadratic correction model [14].

$MET2=±1 - The unit value is used.
$MET2=±2 - The backward Taylor expansion with 3 terms [317] is used.
$MET2=±3 - The backward Taylor expansion with 4 terms [197] is used.
$MET2=±4 - The value determined from the homogeneous model [197] is used.
$MET2=±5 - The modified quasi-Newton condition with 3 terms [375] is used.
$MET2=±6 - The modified quasi-Newton condition with 4 terms [374] is used.

$MET2=±7 - The inverse spectral scaling [46] is used.

If $MET2>0, the basic update is used. If $MET2<0, the modified update [291] is used. The default value is $MET2=4. If $UPDATE='D' or $UPDATE='S', only values $MET2=1,2,3,4 can be used.

If $UPDATE='M', macrovariable $MET2 determines the multistep interpolation model [113].

$MET2=0 - The standard one-step quasi-Newton condition is used.
$MET2=1 - The double step equidistant model is used.
$MET2=2 - The double step accumulative model is used.
$MET2=3 - The first double step fixed-point model is used.
$MET2=4 - The second double step fixed-point model is used.

The default value is $MET2=4.

If $UPDATE='B', macrovariable $MET3 determines the Powell correction [288].

$MET3=1 - The Powell correction is suppressed (the strong update elimination).
$MET3=2 - The Powell correction is suppressed (the weak update elimination).
$MET3=3 - The Powell correction is applied.

The default value is $MET3=1.

If $UPDATE='L', macrovariable $MET3 determines the condition for the Davidon vector selection [178].

$MET3=1 - The conjugacy condition is used.
$MET3=2 - The quasi-Newton condition is used.

The default value is $MET3=1.

If $UPDATE='S', macrovariable $MET3 determines the value of the shift parameter [344].

$MET3=0 - The parameter from the previous iteration is used.
$MET3=1 - A simple choice with the constant relative parameter is used.
$MET3=2 - A choice defined by a quadratic equation is used.
$MET3=3 - A heuristic choice with the relative parameter not greater than half is used.
$MET3=4 - A heuristic choice with the optimally conditioned first iteration (the first formula) is used.
$MET3=5 - A heuristic choice with the optimally conditioned first iteration (the second formula) is used.
$MET3=6 - A heuristic choice derived from the lowest eigenvalue is used.

The default value is $MET3=4.

Macrovariable $MET4 specifies a rule for the determination of the scaling parameter [219].

$MET4=0 - The optimum scaling parameter is used.
$MET4=1 - The BFGS scaling parameter is used.
$MET4=2 - The DFP scaling parameter is used.
$MET4=3 - The geometric mean of the BFGS and DFP parameters is used.
$MET4=4 - The harmonic mean of the BFGS and DFP parameters is used.
$MET4=5 - The arithmetic mean of the BFGS and DFP parameters [84] is used.

The default value depends on the value of macrovariable $MET (values $MET4=1 or $MET4=3 are most frequently used).

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

L-G-1,	L-S-1,	L-B-1,	L-I-1,	L-P-1,	L-M-1,
					L-M-3,
G-G-1,	G-S-1,	G-B-1,			G-M-1,
G-G-2,	G-S-2,	G-B-2,			G-M-2,
					G-M-3,
					G-M-4,
					G-M-5,
					G-M-6,
					G-M-7.

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

3.8.2 Variable metric methods for problems with sparse or partitioned Hessian matrices

Variable metric methods for problems with sparse Hessian matrices are generated from the driver template U1FSU1 if $CLASS='VM' and $HESF='S'. In this case, the sparse Hessian matrix has a general pattern specified by arrays IH, JH (see Section 2.3), which is preserved by sparse variable metric updates. If $DECOMP='M', the types of sparse variable metric updates are specified by using the macrovariable $UPTYPE:

$UPTYPE='B' - The basic variable metric updates derived from variational principles [322].

$UPTYPE='F' - The fractioned variable metric updates introduced in [337], which combine the variable metric and the Newton approaches. Fractioned updates can only be used in the unconstrained case.

The default value is $UPTYPE='B'. The individual sparse variable metric updates (or families) are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The Marwil projection update [248] is considered.

$UPDATE='T' - The Toint projection update (the best method given in [330]) is considered.

$UPDATE='B' - The partitioned variable metric updates from the Broyden family [134]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default value is $UPDATE='B' in case $MODEL='AF' or $MODEL='AQ' or $MODEL='AP' and $UPDATE='M' otherwise.

If $UPTYPE='B' and either $UPDATE='M' or $UPDATE='T', the sparse updates are applied on various sparsity patterns. These patterns are specified by using the macrovariable $CHORDAL:

$CHORDAL='N' - The sparse updates are applied on the input Hessian pattern.

$CHORDAL='Y' - The sparse updates are applied on an extended Hessian pattern which corresponds to a chordal graph.

The default value is $CHORDAL='N'. The choice $CHORDAL='Y' is ignored (replaced by $CHORDAL='N') if $UPTYPE='F' or $UPDATE='B'.

If $UPDATE='B', the particular update is specified by using the macrovariable $MET.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

If $DECOMP='G', less efficient sparse product form updates from the Broyden family are used. In this case, the particular update is specified by using the macrovariable $MET as in the previous case. The default value is $MET=1.

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

L-G-1,	L-M-1,
	L-M-3,
G-G-1,	G-M-1,
	G-M-2,
	G-M-3,
	G-M-4,
	G-M-5,
	G-M-6,
	G-M-7,
	G-M-8,
	G-M-9.

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

Variable metric methods for problems with partitioned Hessian matrices are generated from the driver template U1FBU1 if $CLASS='VM' and $HESF='B'. This choice is permitted only if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'. In this case, the Hessian matrix has a partitioned pattern specified by arrays IAG, JAG (see Section 2.6). Only the partitioned variable metric updates, specified by the choice $UPDATE='B', can be used. These updates are the same as in case the Hessian matrix is sparse with a general pattern, but the use of a partitioned pattern for the direction determination is usually less efficient due to more expensive matrix operations.

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

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

The default choice is L-M-3.

3.9 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 from given explicit formulas or automatically using automatic differentiation technique or numerically using gradient differences. The UFO system performs automatic differentiation (if $IADS=2) or numerical differentiation (if $IADS<2) automatically whenever the macrovariable $HMODELF (or $FGHMODELF) is not defined. Modified Newton methods are realized in three different forms for $HESF='D', $HESF='S' and $HESF='B' depending on the Hessian matrix specification. Although the modified Newton methods can be realized as line search methods ($TYPE='L'), it is more advantageous to realize them as trust region methods ($TYPE='G') or cubic regularization methods ($TYPE='R').

3.9.1 Modified Newton methods for problems with dense Hessian matrices

Modified Newton methods for problems with dense Hessian matrices are generated from the driver template U2FDU1 if $CLASS='MN' and $HESF='D'. In this case, we have to compute $n(n+1)/2$ second derivatives. Symmetric Hessian matrix is stored in the packed form containing $n(n+1)/2$ elements. Possible specifications (type-decomposition-number) for dense modified Newton methods in the unconstrained case are these:

L-G-1,	L-S-1,	L-B-1,	L-M-1,
L-G-2,	L-S-2,	L-B-2,	L-M-2,
			L-M-3,
G-G-1,	G-S-1,	G-B-1,	G-M-1,
G-G-2,	G-S-2,	G-B-2,	G-M-2,
			G-M-3,
			G-M-4,
			G-M-5,
			G-M-6,
			G-M-7,
			R-M-7.

The default choice is G-M-7. In both the box constrained and the linearly constrained cases we cannot use specifications with $DECOMP='S' and $DECOMP='B'.

3.9.2 Modified Newton methods for problems with sparse or partitioned Hessian matrices

Modified Newton methods for problems with sparse Hessian matrices are generated from the driver template U2FSU1 if $CLASS='MN' and $HESF='S'. In this case, the sparse Hessian matrix has a general pattern specified by arrays IH, JH (see Section 2.3). If $MODEL='FF', only the structurally nonzero second derivatives are given analytically by using the prescribed pattern. The numerical computation of the second derivatives is based on the fact that a substantially lower number of differences is used in comparison with the dense case. The determination of suitable differences is a combinatorial problem equivalent to a graph coloring problem [50], [49]. If $MODEL='AF' or $MODEL='AQ' or $MODEL='AP', arrays IH, JH, HF, are determined from arrays IAG, JAG. If the second derivatives are not given analytically, there are two possibilities distinguished by using the macrovariable $NUMDER:

$NUMDER=1 - Second derivatives of individual approximating functions are computed.
$NUMDER=2 - The graph coloring problem [50], [49] (as in case $MODEL='FF') is solved.

The default value is $NUMDER=1. If $NUMDER=1. Only the nonzero second derivatives of the approximating functions are given analytically by using the prescribed pattern. The numerical computation of the second derivatives is based on the fact that the approximating functions depend on a small number of variables so that the number of differences is substantially lower in comparison to the dense case.

If $MODEL='AQ' (sum of squares), the combination [200] of both the modified Newton and the modified Gauss-Newton methods can be used. This choice is possible by using the macrovariable $MET.

$MET=1 - The modified Newton method is used.
$MET=2 - The combined method is used.

The default value is $MET=2.

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

L-G-1, L-M-1,
L-M-3,
G-G-1, G-M-1,
G-M-2,
G-M-3,
G-M-4,
G-M-5,
G-M-6,
G-M-7,
G-M-8,
G-M-9,
R-M-7.

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

Modified Newton methods for problems with partitioned Hessian matrices are generated from the driver template U2FBU1 if $CLASS='MN' and $HESF='B'. This choice is permitted only if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'. In this case, the Hessian matrix has a partitioned pattern specified by arrays IAG, JAG (see Section 2.6). A computation of the second derivatives is the same as in case the Hessian matrix is sparse with a general pattern and $NUMDER=1, but the use of a partitioned pattern for the direction determination is usually less efficient due to more expensive matrix operations.

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

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

The default choice is G-M-3.

3.10 Truncated Newton methods

Truncated Newton methods are specified by the statement $CLASS='TN' and are generated from the driver template U1FSU2. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. Truncated Newton methods are very efficient for large problems with computationally simple objective functions ($KCF=1 or $KCA=1). The main advantage of truncated Newton methods is that no matrices are used (implicitly $HESF='N'). This fact highly decreases storage requirements.

The precision control strategy for inexact solution is specified by the macrovariable $MOS.

$MOS=1 - The simple strategy is used for precision control.
$MOS=2 - The geometric decreasing strategy is used for precision control.
$MOS=3 - The harmonic decreasing strategy is used for precision control.

The default value is $MOS=3. The preconditioning technique is specified by the macrovariable $MOS2.

$MOS2=0 - Preconditioning is not used.
$MOS2=1 - Preconditioning by the limited-memory variable metric methods based on vector recurrences is used.
$MOS2=2 - Preconditioning by the limited-memory variable metric methods based on compact matrix representations is used.
$MOS2=3 - Preconditioning by diagonal matrices is used.
$MOS2=4 - Preconditioning by tridiagonal matrices is used.
$MOS2=5 - Preconditioning by pentadiagonal matrices is used.

$MOS2=6 - Preconditioning by the Lanczos method is used.

The default value is $MOS2=0.

If $MOS2=1, then macrovariable $MOS3 specifies the limited-memory variable metric update.

$MOS3=1 - The BFGS update [272] is used.
$MOS3=2 - The update described in [157] is used.
$MOS3=3 - The modified BFGS update that utilizes vectors from the previous iteration [350] is used.
$MOS3=4 - The modified BFGS update that constructs conjugate directions is used.

The default value is $MOS3=1.

If $MOS2=2, then macrovariable $MOS3 specifies the matrix representation of the BFGS update.

$MOS3=1 - The explicit matrix representation [39] is used.
$MOS3=2 - The recursive matrix representation [239] is used.

The default value is $MOS3=1.

If $MOS2=3 or $MOS2=4 or $MOS2=5, then the preconditioning technique can be determined using macrovariables $MOS1, $MOS3, $MOS4 and $MOS5. The macrovariable $MOS1 specifies the number of unpreconditioned iterations.

$MOS1=0 - The preconditioning is used in all iterations.
$MOS1>1 - The preconditioning is not used in the first $MOS1 iterations, see [240].

The default value is $MOS1=0. The macrovariable $MOS3 specifies the method for obtaining the band preconditioner.

$MOS3=1 - The band preconditioner is obtained by the numerical differentiation, see [217].
$MOS3=2 - The band preconditioner is obtained by the BFGS method equivalent to the preconditioned conjugate gradient method, see [217] and [262].

The default value is $MOS3=1. If $MOS3=2, then the macrovariable $MET3 specifies the maximum number of the BFGS updates (if $MET3=0, then this number is infinity). The default value is $MET3=0. If $MOS2=6, then the macrovariable $MOS3 gives the maximum number of the Lanczos steps (the order of the Lanczos tridiagonal matrix). The default value is $MOS3=5. The macrovariable $MOS4 specifies the rejecting the band preconditioner after the Gill-Murray [123] decomposition.

$MOS4=0 - Preconditioning in both the ill-conditioned and the indefinite cases is suppressed.
$MOS4=1 - Preconditioning in the ill-conditioned case is suppressed.
$MOS4=2 - Preconditioning is always used.

The default value is $MOS4=0.

The macrovariable $MOS5 specifies the correction of the band preconditioner. If $MOS2=3, then the following choices are possible.

$MOS5=0 - Correction is not used.
$MOS5=1 - The diagonal elements are replaced by their absolute values.
$MOS5=2 - The diagonal elements are replaced by their absolute values. These new values are increased if necessary.

The default value is $MOS5=2. If $MOS2=4 and $MOS3=1 or $MOS2=5 and $MOS3=1, then the following choices are possible.

$MOS5=0 - Correction is not used.
$MOS5=1 - The diagonal elements are replaced by their absolute values.

$MOS5=2 - The diagonal elements are replaced by their absolute values. These new values are increased if necessary.

$MOS5=3 - The diagonal elements are replaced by their absolute values and the off-diagonal elements are corrected to ensure positive definiteness.

The default value is $MOS5=2. If $MOS2=4 or $MOS2=5 and $MOS3=2, then the following choices are possible.

$MOS5=0 - Correction is not used.
$MOS5=1 - The off-diagonal elements are decreased to ensure positive definiteness.
$MOS5=2 - The off-diagonal elements are adaptively decreased to ensure positive definiteness.
$MOS5=3 - The off-diagonal elements are corrected to ensure positive definiteness.

The default value is $MOS5=3. If $MOS2=6, then the following choices are possible.

$MOS5=1 - Preconditionong in the current iteration is used.
$MOS5=2 - Preconditionong in the next iteration is used.

The default value is $MOS5=1.

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

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

The default choice is G-M-3. Note that choices G-M-4 and G-M-5 are also permitted, but no preconditioning can be used in these cases.

3.11 Modified Gauss-Newton methods for nonlinear least squares problems

Modified Gauss-Newton methods, specified by the statement $CLASS='GN', are special optimization methods for solving nonlinear least squares ($MODEL='AQ') or nonlinear least powers ($MODEL='AP') problems and can be used also for solving systems of nonlinear equations ($MODEL='NE'). These 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 variable metric updates. Modified Gauss-Newton methods are realized in five different forms (for $HESF='D', $HESF='S', $HESF='B', $HESF='N' with $JACA='D', $HESF='N' with $JACA='S') depending on the Hessian matrix specification or the Jacobian matrix specification. Although the modified Gauss-Newton methods can be realized as the line search methods ($TYPE='L'), it is more advantageous to realize them as the trust region methods ($TYPE='G'). If the Hessian matrix is not specified ($HESF='N'), the normal equation matrix is not used. The Jacobian matrix, defining a linear least squares problem, is utilized in each iteration instead. Such so-called normal equation free Gauss-Newton methods are realized in two different forms (for $JACA='D' and $JACA='S') depending on the Jacobian matrix specification.

3.11.1 Modified Gauss-Newton methods for problems with dense Hessian matrices

Modified Gauss-Newton methods for problems with dense Hessian matrices are generated from the driver template U2SDU1 if $CLASS='GN' and $HESF='D'. In this case, the normal equation matrix is also dense and we can use hybrid methods with dense updates:

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

$UPDATE='B' - The 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 [200].

$UPDATE='F' - The Fletcher hybrid approach [1], [112] is used. The Hessian matrix is approximated either by the normal equation matrix or by the matrix obtained by using the variable metric updates. The decision between the two cases is based on the rate of the function value decrease and on the normal equation matrix conditioning.

$UPDATE='S' - The Dennis structured approach [90] 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.

The default value is $UPDATE='N'.

Individual variable metric updates from the above families are specified by using the macrovariable $MET.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The original (unsafeguarded) rank-one method is used.

The value $MET=4 is allowed only if $UPDATE='S' and is the default in this case. The value $MET=1 is the default in the other cases.

If $UPDATE='B' or $UPDATE='S', macrovariable $MET1 determines the scaling parameter [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=5 (values $MET1=3 and $MET1=4 cannot be used if $UPDATE='S').

If $UPDATE='B', macrovariable $MET2 determines the parameter of the nonquadratic correction model [14].

$MET2=1 - The unit value is used.
$MET2=2 - The backward Taylor expansion with 3 terms [317] is used.

The default value is $MET2=1.

If $UPDATE='S', macrovariable $MET2 determines the type of scaling.

$MET2=1 - The standard scaling is used.
$MET2=2 - The Biggs scaling [15] is used.

The default value is $MET2=1.

If $UPDATE='S', macrovariable $MET3 determines the realization of the hybrid method.

$MET3=1 - The standard realization is used.
$MET3=2 - The Huschens realization [155] is used.

The default value is $MET3=1.

If $UPDATE='S', macrovariable $MET4 determines the quasi-Newton condition.

$MET4=1 - The standard quasi-Newton condition is used.
$MET4=2 - The quasi-Newton condition with intermediate gradients [91] is used.

The default value is $MET4=1.

If $UPDATE='B' or $UPDATE='F', the variable metric updates 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 described in [200]. A decision between these possibilities is mediated by the macrovariable $MET5.

$MET5=1 - The simple update is used.
$MET5=2 - The cumulative update is used.

The default values are $MET5=2 if $UPDATE='B' and $MET5=1 if $UPDATE='F'.

In the dense case, the modified Gauss-Newton methods can be realized with additional special matrix decomposition which cannot be used in other cases. If $DECOMP='R', the recursive QR decomposition [281] is used with an additional correction of the upper triangular matrix R. Possible specifications (type-decomposition-number) for dense modified Gauss-Newton methods in the unconstrained case are these:

L-G-1,	L-S-1,	L-B-1,	L-R-1,	L-M-1,
				L-M-3,
G-G-1,	G-S-1,	G-B-1,	G-R-1,	G-M-1,
G-G-2,	G-S-2,	G-B-2,	G-R-2,	G-M-2,
				G-M-3,
				G-M-4,
				G-M-5,
				G-M-6,
				G-M-7,
T-G-1,	T-S-1,		T-R-1,	T-M-1,
T-G-2,				
	T-S-7,			T-M-7,
				R-M-7,
				M-M-1.

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

3.11.2 Modified Gauss-Newton methods for problems with sparse and partitioned Hessian matrices

Modified Gauss-Newton methods for problems with sparse Hessian matrices are generated from the driver template U2SSU1 if $CLASS='GN' and $HESF='S'. In this case, the normal equation matrix is sparse with a general pattern specified by arrays IH, JH (see Section 2.3). We can use hybrid methods with the following sparse or partitioned updates:

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

$UPDATE='M' - The sparse update based on the Marwil projection is applied either to the normal equation matrix or to the previous approximation of the Hessian matrix if conditions for leaving the modified Gauss-Newton method are satisfied [200].

$UPDATE='B' - The 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 [200].

$UPDATE='D' - The Brown-Dennis structured approach [30] 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='S' - The Dennis structured approach [90] 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.

The default value is $UPDATE='M'.

Individual variable metric updates from the above families are specified by using the macrovariable $MET as in the dense case. The default value is $MET=4. The macrovariable $MET is not utilized if $UPDATE='M'.

Variable metric updates ($UPDATE='M' or $UPDATE='B') can be realized either as simple updates (normal equation matrix is updated) or as cumulative updates (previous approximation of the Hessian matrix is updated). A decision between these possibilities is mediated by the macrovariable $MET5 similarly as in the dense case. The default value is $MET5=2 if $UPDATE='M' and $MET5=1 if $UPDATE='B'

If $UPDATE='D', we can use several switches for utilizing variable metric updates specified by the macrovariable $MET3.

$MET3=0 - The Fletcher and Xu switch [112] is used.
$MET3=1 - The modification of the Fletcher and Xu switch is used.
$MET3=2 - The Dennis and Welsch switch [95] is used.
$MET3=3 - The Ramsin and Wedin switch [296] is used.

The default value is $MET3=0.

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

L-G-1,	L-M-1,
	L-M-3,
G-G-1,	G-M-1,
G-G-2,	G-M-2,
	G-M-3,
	G-M-4,
	G-M-5,
	G-M-6,
	G-M-7,
	G-M-8,
T-G-1,	T-M-1,
	T-M-7,
	R-M-7,
	M-M-1.

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

Modified Gauss-Newton methods for problems with partitioned Hessian matrices are generated from the driver template U2SBU1 if $CLASS='GN' and $HESF='B'. In this case, the normal equation matrix has a partitioned pattern specified by arrays IAG, JAG (see Section 2.6). We can use hybrid methods with partitioned updates $UPDATE='N', $UPDATE='B', $UPDATE='D', $UPDATE='S' whose details are explained in Section 3.11.2. Note that the use of a partitioned pattern for the direction determination is usually less efficient due to more expensive matrix operations.

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

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

The default choice is G-M-3.

3.11.3 Modified Gauss-Newton methods for problems with dense Jacobian matrices

Modified Gauss-Newton methods for problems with dense Jacobian matrices are generated from the driver template U1SDU1 if $CLASS='GN', $HESF='N' and $JACA='D'. In this case, we can use hybrid methods with dense rectangular updates:

$UPDATE='N' - No update is used. The method utilizes the rectangular Jacobian matrix.

$UPDATE='L' - The special variable metric update [242] is applied either to the Jacobian matrix or to the rectangular factor of the previous approximation of the Hessian matrix if conditions for leaving the Gauss-Newton method are satisfied [200].

$UPDATE='S' - The Shen and Zhou structured partitioned update [312] is used if conditions for leaving the Gauss-Newton method are satisfied.

The default value is $UPDATE='S'.

If $UPDATE='L', the variable metric update can be realized either as a simple update (Jacobian matrix is updated) or as a cumulative update (the rectangular factor of the previous approximation of the Hessian matrix is updated). A decision between these possibilities is mediated by the macrovariable $MET5.

$MET5=1 - The simple update is used.

$MET5=2 - The cumulative update is used.

The default value is $MET5=2.

If $UPDATE='S', the particular realization of the variable metric update can be specified by macrovariables $MET2, $MET3, $MET4 as in Section 3.11.1 (when $UPDATE='S').

The Jacobian matrix can be stored either rowwise ($JASA='R') or columnwise ($JASA='C'). The default values are $JASA='C' if $DECOMP='A' or $DECOMP='Q' and $JASA='R' if $DECOMP='E'. The specification $JASA='C' cannot be used for problems with linear approximating functions if $NAL>0.

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

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

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

3.11.4 Modified Gauss-Newton methods for problems with sparse Jacobian matrices

Modified Gauss-Newton methods for problems with sparse Jacobian matrices are generated from the driver template U1SSU1 if $CLASS='GN', $HESF='N' and $JACA='S'. In this case, we can use hybrid methods with simple variable metric updates:

$UPDATE='N' - No update is used. The method utilizes the original Jacobian matrix.

$UPDATE='V' - The simple factorized BFGS update [200] 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 [200] is used. The second order information is approximated by adding a special dense row to the Jacobian matrix.

The default value is $UPDATE='N'.

If $UPDATE='R', we can use several switches for utilizing variable metric updates, specified by the macrovariable $MET3 as in Section 3.11.2 (when $UPDATE='D'). The default value is $MET3=0.

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

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

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

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

3.12 Quasi-Newton methods for systems of nonlinear equations

Quasi-Newton methods, specified by the statement $CLASS='QN', are special methods for solving systems of nonlinear equations ($MODEL='NE') which can be also used for minimizing the sum of squares ($MODEL='AQ'). These 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.

3.12.1 Quasi-Newton methods for systems with dense Jacobian matrices

Quasi-Newton methods for systems with dense Jacobian matrices are generated from the driver template U0SDU1 if $CLASS='QN' and $JACA='D'. In this case, we can use quasi-Newton methods with dense updates:

$UPDATE='N' - No update is used. If the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined), an approximation of the Jacobian matrix is computed numerically by using differences.

$UPDATE='A' - The adjoint quasi-Newton updates [302], [303] are used in almost all iterations. These updates are efficient only if the first derivatives are specified analytically.

$UPDATE='B' - The Broyden [28] rank-one quasi-Newton updates are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.

$UPDATE='X' - The Fletcher and Xu [364] rank-one quasi-Newton updates are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.

The default value is $UPDATE='B'.

If $UPDATE='A', the individual adjoint quasi-Newton methods are specified by using the macrovariable $MET.

$MET=1 - The basic residual adjoint quasi-Newton update [303] is used.
$MET=2 - The new quasi-Newton update [241] is used.
$MET=3 - The secant residual adjoint quasi-Newton update [302] is used.
$MET=4 - The tangent (two-sided) residual adjoint quasi-Newton update [302] is used.

The default value is $MET=2.

If $UPDATE='B' or $UPDATE='X' , the individual quasi-Newton methods are specified by using the macrovariable $MET.

$MET=1 - The good Broyden update [28] is used.
$MET=2 - The column update [246] is used.
$MET=3 - The Greenstadt update [133] is used.
$MET=4 - The first Todd optimal update [159] is used.
$MET=5 - The second Todd optimal update [159] is used.

If $DECOMP='A', only values $MET=1 and $MET=2 can be used. The default value is $MET=1.

If $DECOMP='E', the rank-one updates are realized by one of the following methods distinghuised by using the macrovariable $MOS3:

$MOS3=1 - The Bennet method without permutations [26] is used.
$MOS3=2 - The Schwetlick method with permutations [320] is used.
$MOS3=3 - The combination of the previous methods is used.

The default value is $MOS3=1.

The Jacobian matrix can be stored either rowwise ($JASA='R') or columnwise ($JASA='C'). The default values are $JASA='C', if $DECOMP='A' or $DECOMP='Q', and $JASA='R' if $DECOMP='E' (the specification $JASA='C' cannot be used for problems with linear approximating functions with $NAL>0).

Dense quasi-Newton methods can only be used in the unconstrained case. Possible specifications (type-decomposition-number) for dense quasi-Newton methods are these:

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

The default choice is G-Q-1. The specification $DECOMP='E' can only be used if NA=NF (system of nonlinear equations).

3.12.2 Quasi-Newton methods for systems with sparse Jacobian matrices

Quasi-Newton methods for systems with sparse Jacobian matrices are generated from the driver template U0SSU1 if $CLASS='QN' and $JACA='S'. In this case, 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 [51] graph coloring algorithm is used.

The default value is $NUMDER=1.
Moreover, various sparse quasi-Newton updates which preserve the pattern of the Jacobian matrix can be used.

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

$UPDATE='N' - No update is used. If the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined), an approximation of the Jacobian matrix is computed numerically by using differences.
$UPDATE='A' - Sparse Schubert-like adjoint quasi-Newton update is used in almost all iterations. This update is efficient only if the first derivatives are specified analytically.
$UPDATE='B' - Sparse quasi-Newton updates are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.
$UPDATE='S' - Modified Newton methods such as the row scaling update are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.
$UPDATE='T' - Sparse tensor correction is used in almost all iterations. This correction is advantageous only if the first derivatives are specified analytically.

The default value is $UPDATE='N'.

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

$UPDATE='N' - No update is used. If the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined), an approximation of the Jacobian matrix is computed numerically by using differences.
$UPDATE='A' - Sparse Schubert-like adjoint quasi-Newton update is used in almost all iterations. This update is efficient only if the first derivatives are specified analytically.
$UPDATE='B' - Sparse quasi-Newton updates are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.
$UPDATE='S' - Modified Newton methods that use the row scaling update in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.
$UPDATE='C' - Cyclic column determination methods are used in almost all iterations. If the first derivatives are not specified analytically, the Jacobian matrix is approximated numerically by using differences after restarts.
$UPDATE='T' - Sparse tensor correction is used in almost all iterations. This correction is advantageous only if the first derivatives are specified analytically.

The default value is $UPDATE='N'.
Individual quasi-Newton methods are specified by using the macrovariable $MET. If $UPDATE='A' the macrovariable $MET is not used. If $UPDATE='B', the following specifications are possible.

$MET=1 - The Schubert update [305] is used.
$MET=2 - The Bogle-Perkins update [24] is used.
$MET=3 - The column update [246] is used.

The default value is $MET=1.
If $UPDATE='S', the following specifications are possible.

$MET=0 - The modified Newton method is used.
$MET=1 - The row scaling update [246] is used.

The default value is $MET=0.
If $UPDATE='C', the following specifications are possible.

$MET=0 - The cyclic column determination method [170] is used.
$MET=1 - The cyclic column determination method [170] is used followed by the Schubert update [305].
$MET=2 - The cyclic column determination method [170] is used followed by the Bogle-Perkins update [24].
$MET=3 - The cyclic column determination method [170] is used followed by the column update [246].

The default value is $MET=1.

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

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

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

3.13 Quasi-Newton methods with limited memory for systems of nonlinear equations

Quasi-Newton methods with limited memory are specified by the statement $CLASS='QL' and are generated from the driver template U0SSU2. The number of QN steps is specified by the macrovariable $MF (the default value is $MF=5). These methods are special methods for solving sparse systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore, only the case NA=NF is permitted. Quasi-Newton methods with limited memory use an initial approximation of the sparse Jacobian matrix together with several small-size matrices which are updated in every iteration in such a way that their product approximates the Jacobian matrix as precisely as possible [39]. There are two possibilities which are distinguished by using the macrovariable $UPDATE:

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

$UPDATE='B' - The Broyden good update with limited memory [39] is used in almost all iterations. After the restart, the Jacobian matrix is approximated numerically by using differences.

The default value is $UPDATE='N'.

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

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

The default choice is G-E-3.

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

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

$UPDATE='B' - The inverse column scaling update [247] is used in almost all iterations. After the restart, the Jacobian matrix is approximated numerically by using differences.

The default value is $UPDATE='B'.

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

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

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

3.14 Truncated Newton methods for systems of nonlinear equations

Truncated Newton methods are specified by the statement $CLASS='TN' and are generated from the driver template U0SSU3. These methods are special methods for solving systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore, only the case NA=NF is permitted. Truncated Newton methods differ from quasi-Newton methods in that the sparse Jacobian matrix multiplication is replaced by the numerical differentiation. These methods are very efficient for large problems with computationally simple functions in nonlinear equations ($KCA=1). The main advantage of truncated Newton methods is that no matrices are used (implicitly $JACA='N'). This fact highly decreases storage requirements.

Truncated Newton methods are implemented either as the line search methods or as the trust region methods and are based on the smoothed CGS subalgorithm. This subalgorithm can be preconditioned by using the tridiagonal decomposition. This possibility is determined by the macrovariable $MOS2.

$MOS2=0 - The tridiagonal decomposition is not used.
$MOS2=1 - The tridiagonal decomposition is used before the iterative process is started.
$MOS2=2 - The tridiagonal decomposition is used as a preconditioner.
$MOS2=3 - Both previous cases are assumed.

The default value is $MOS2=0.

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

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

The default choice is G-E-3.

3.15 Simple quasi-Newton and Brent methods for systems of nonlinear equations

Simple quasi-Newton and Brent methods are specified by the statement $CLASS='QB' and are generated from the driver template U0SDU2. These methods are special simple methods for solving dense systems of nonlinear equations ($MODEL='NE') when the first derivatives are not specified analytically (the macrovariable $GMODELA is not defined). Therefore, only the case NA=NF is permitted. Individual methods are selected using the macrovariable $NUMBER:

$NUMBER=1 - The Brent method described in [27].
$NUMBER=3 - The simple Newton method (this method can also be used if the macrovariable $GMODELA is defined).

The default value is $NUMBER=3.

3.16 Simplex type methods for linear programming problems

Simplex type methods for linear programming problems are specified by the statement $CLASS='LP'. These methods are realized in two different forms (for $JACC='D' and $JACC='S') depending on the constraint Jacobian matrix specification and are generated from the driver templates U1LDL1 and U1LSL1.

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

$NUMBER=1 - Primal reduced gradient (null-space) method (like the method proposed in [122]), which is a special implementation of the steepest descent reduced gradient method.
$NUMBER=2 - Primal projected gradient (range-space) method, which is a special implementation of the steepest descent projected gradient method.

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

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

$NUMBER=1 - Primal reduced gradient (null-space) simplex type method described in [335].
$NUMBER=2 - Primal projected steepest descent (range-space) method.

A possible specification (type-number) for sparse simplex type linear programming methods are L-1 and L-2. The default choice is L-1, but this choice is not suitable for problems with equality constraints.

3.17 Interior point methods for linear programming problems

Interior point methods for linear programming problems are specified by the statement $CLASS='LI' and are generated from the driver template U1LSL2. These methods, based on an infeasible primal-dual predictor-corrector strategy, can be used only in the sparse case when $JACC='S'. Moreover, only the standard LP constraints $Ax = b$, $x \geq 0$ can be considered at present. Individual methods are chosen by using the macrovariable $MLP:

$MLP=1 - The first algorithm of Miao [252].
$MLP=2 - The second algorithm of Miao [252].
$MLP=3 - The Mizuno algorithm [255].

The default value is $MLP=1.

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

$NUMBER=1 - Direct solution based on the Gill-Murray decomposition applied to the Schur complement.
$NUMBER=2 - Direct solution based on the Bunch-Parlett decomposition applied to the original Karush-Kuhn-Tucker system.
$NUMBER=3 - Iterative solution based on the conjugate gradient method applied to the Schur complement.

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

3.18 Simplex type methods for quadratic programming problems

Simplex type methods for quadratic programming problems 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 and are generated from the driver templates U1QDL1 and U1QSL1.

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

$NUMBER=1 - Primal reduced gradient (null-space) method (like the method proposed in [124]), which is a special implementation of the Newton reduced gradient method.
$NUMBER=2 - Primal projected gradient (range-space) method (like the method proposed in [102]), which is a special implementation of the Newton projected gradient method.
$NUMBER=3 - Dual projected gradient (range-space) method (like the method proposed in [128]).

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

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

$NUMBER=1 - Primal reduced gradient (null-space) simplex type method described in [335].

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

Possible specification (type-number) for sparse simplex type quadratic programming methods are L-1 and L-2. The default choice is L-2. The choice L-1 is not suitable for problems with equality constraints.

3.19 Interior point methods for quadratic programming problems

Interior point methods for quadratic programming problems are specified by the statement $CLASS='QI' and are generated from the driver template U1QSL2. These primal-dual methods, based on the logarithmic barrier function and iterative solution of the indefinite Karush-Kuhn-Tucker system, can be used only in the sparse case when $JACC='S'. Interior point methods for quadratic programming problems are, in fact, the same as methods with the choices $TYPE='L' and $DECOMP='I' described in Section 3.32.

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

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [99] of the indefinite Karush-Kuhn-Tucker system is used.

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system is applied which uses a special determination of the required precision. The particular realization of the inexact preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

The default value is $NUMBER=3.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.
$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.
$MOS1=2 - The precision guaranteeing descent direction is used together with an extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - Preconditioning is suppressed.
$MOS2=±1 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the normal equation form.
$MOS2=±2 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the augmented system form.

If $MOS2>0, a complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.
$MOS3=1 - A simple one-dimensional residual smoothing is used.

The default value is $MOS3=0.

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

3.20 Proximal bundle methods for nonsmooth optimization

Proximal bundle methods for nonsmooth optimization problems are specified by the statement $CLASS ='BM' and are generated from the driver template U1FDU3. These methods use a bundle of gradients computed in trial points and updated in every iteration. The size of this bundle is specified by the macrovariable $MB (the default value is $MB='NF+3'). Proximal bundle methods solve a special quadratic programming subproblem derived from the cutting plane approach [342]. This subproblem is, in fact, the same as in the recursive quadratic programming methods for minimax problems. Proximal bundle methods are realized only for unconstrained or linearly constrained dense problems ($JACA='D'). The special quadratic programming subproblem can be solved by using the following methods:

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [182].
$NUMBER=2 - Primal projected gradient (range-space) method, which is a special implementation of the Newton projected gradient method.

The special quadratic programming subproblem is defined in such a way that it has a diagonal Hessian matrix. There are several methods for computing the diagonal weight coefficients which are selected by using the macrovariables $MOS and $MES2.

$MOS=1 - If $MES2=1, the weights are updated by using the curvature of the one-dimensional quadratic function.

$MOS=1 - If $MES2=2, the weights are updated by using the minimum position estimate (suitable for polyhedral and nearly polyhedral functions).

$MOS=2 - The weights are updated by using the quasi-Newton condition.

The default values are $MOS=1 and $MES2=1.

Proximal bundle methods are only realized as line search methods in two modifications which are specified by the macrovariable $MEX.

$MEX=0 - The convex version is assumed.

$MEX=1 - The nonconvex version is assumed and we can define a measure of nonconvexity by using the macrovariable $ETA5. The default value is $ETA5=0.25.

The default values is $MEX=1.

Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Proximal bundle methods are sensitive to the values of parameters $ETA5 and $XMAX. Therefore, these values should be carefully chosen.

Possible specifications (type-number) for proximal bundle methods are L-1 and L-2. The default choice is L-1. Proximal bundle methods can be used when $KSF=3 or $KSA=3. They can be also used for minimizing the sum of absolute values ($MODEL='AA') and for solving minimax problems ($MODEL='AM').

3.21 Bundle Newton methods for nonsmooth optimization

Bundle Newton methods for nonsmooth optimization problems are specified by the statement $CLASS ='BN' and are generated from the driver template U2FDU3. These methods use a bundle of gradients and Hessian matrices computed in trial points and updated in every iteration. The size of this bundle is specified by the macrovariable $MB (the default value is $MB='NF+3'). Bundle Newton methods solve a special quadratic programming subproblem derived from the cutting plane approach which contains second order information [226]. This subproblem is, in fact, the same as in recursive quadratic programming methods for minimax problems. Bundle Newton methods are only realized for unconstrained or linearly constrained dense problems ($JACA='D'). The special quadratic programming subproblem can be solved by using the following methods:

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [182].

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

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

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

Possible specifications (type-number) for bundle Newton methods are L-1 and L-2. The default choice is L-1. Bundle Newton methods can be used when $KSF=3 or $KSA=3. They can be also used for minimizing the sum of absolute values ($MODEL='AA') and for solving minimax problems ($MODEL='AM').

3.22 Bundle variable metric methods for nonsmooth optimization

Bundle variable metric methods for nonsmooth optimization problems are specified by the statement $CLASS='BV' and are generated from the driver template U1FDU5. These methods are based on a special realization of variable metric method updates. This realization uses special null steps and restarts. The stepsize selection is based on the polyhedral approximation obtained by using bundles of points and subgradients. Bundle variable metric methods are realized only for unconstrained or linearly constrained dense problems ($JACA='D'). They need not solve any quadratic programming subproblem.

Bundle variable metric methods are realized as line search methods in two modifications which are specified by the macrovariable $MEX.

$MEX=0	- The convex version [229] is assumed.
$MEX=1	- The nonconvex version [343] is assumed and we can define a measure of nonconvexity by using the macrovariable $ETA5. The default value is $ETA5=0.25.

The default values is $MEX=1.

Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Bundle variable metric methods are sensitive to the values of parameters $ETA5 and $XMAX. Therefore, these values should be carefully chosen.

Bundle variable metric methods use an auxiliary bundle of gradients which serves for the initial stepsize determination in the line search subalgorithm. The size of this bundle is specified by the macrovariable $MB (the default value is $MB='NF+3').

Possible specifications (type-number) for bundle variable metric methods are L-1 and L-2. The default choice is L-1. Bundle variable metric methods can be used when $KSF=3 or $KSA=3. They can be also used for minimizing the sum of absolute values ($MODEL='AA') and for solving minimax problems ($MODEL='AM').

3.23 Bundle variable metric methods with limited memory for nonsmooth optimization

Bundle variable metric methods with limited memory for nonsmooth optimization problems are specified by the statement $CLASS ='BL' and are generated from the driver template U1FLU7. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). These methods are based on a special realization of limited memory variable metric updates. This realization uses special null steps and restarts. The stepsize selection is based on the polyhedral approximation obtained by using bundles of points and subgradients. Bundle variable metric methods with limited memory are realized only for unconstrained or box constrained sparse problems ($JACA='S' and $HESF='S'). They need not solve any quadratic programming subproblem.

Bundle variable metric methods with limited memory are realized as line search methods in several modifications which are specified by the macrovariables $MEX, $MEX1, $MEX2, $MEX3.

$MEX=0	- The convex version [229] is assumed.
$MEX=1	- The nonconvex version [343] is assumed and we can define a measure of nonconvexity by using the macrovariable $ETA5. The default value is $ETA5=0.25.
$MEX1=0	- The basic algorithm is used.
$MEX1=1	- An additional restart is used if the direction vector is short.
$MEX1=2	- The controlled correction of the direction vector is used. The correction parameter is specified by the macrovariable $ETA3. The default value is $ETA3=$10^{-12}$.
$MEX1=3	- The permanent correction of the direction vector is used. The correction parameter is specified by the macrovariable $ETA3. The default value is $ETA3=$10^{-12}$.
$MEX2=0	- The basic termination criterion is used.
$MEX2=1	- The extended termination criterion for large-scale problems is used.

$MEX3=0 - The termination is used after a zero difference of gradients.
$MEX3=1 - The extrapolation with a constant stepsize is used after a zero difference of gradients.
$MEX3=2 - The extrapolation with an increasing stepsize is used after a zero difference of gradients.

The default values are $MEX=1, $MEX1=0, $MEX2=0, $MEX3=0.

Variable metric updates are controlled by the macrovariables $MET, $MET2, $MET3, $MET4.

$MET=0 - The limited memory BFGS matrix is updated only in descent steps.
$MET=1 - The limited memory BFGS matrix is updated in all steps.

$MET2=0 - No updates are used in null steps.
$MET2=1 - Rank-one updates are used in $MET3 consecutive null steps.
$MET2=2 - Rank-one updates are used in all null steps.
$MET2=3 - BFGS updates are used in $MET4 consecutive null steps.

The default values are $MET=1, $MET2=1, $MET3=5, $MET4=5.

Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Bundle variable metric methods with limited memory are sensitive to the values of parameters $ETA5 and $XMAX. Therefore, these values should be carefully chosen.

Bundle variable metric methods with limited memory use an auxiliary bundle of gradients which serves for the initial stepsize determination in the line search subalgorithm. The size of this bundle is specified by the macrovariable $MB (the default value is $MB=20).

A possible specification (type-number) for bundle variable metric methods with limited memory is L-1.

3.24 Bundle variable metric methods for sparse sum of absolute values

If $MODEL='AA', then the objective function is the sum of absolute values. In this case, bundle variable metric methods [238] are specified by using the statement $CLASS ='BV' and are generated from the driver template U1FSU5. These methods are based on a special realization of partitioned variable metric updates. This realization uses special null steps and restarts. The stepsize selection is based on the polyhedral approximation obtained by using bundles of points and subgradients. Bundle variable metric methods for sparse sum of absolute values are realized only for unconstrained or box constrained sparse problems ($JACA='S' and $HESF='S'). They need not solve any quadratic programming subproblem.

Bundle variable metric methods for sparse sum of absolute values are realized as line search methods in several modifications which are specified by the macrovariables $MEX, $MEX1, $MEX2, $MEX3.

$MEX=0 - The convex version [229] is assumed.
$MEX=1 - The nonconvex version [343] is assumed and we can define a measure of nonconvexity by using the macrovariable $ETA5. The default value is $ETA5=0.25.

$MEX1=0 - The basic algorithm is used.
$MEX1=1 - An additional restart is used if the direction vector is short.
$MEX1=2 - The controlled correction of the direction vector is used. The correction parameter is specified by the macrovariable $ETA3. The default value is $ETA3=$10^{-12}$.
$MEX1=3 - The permanent correction of the direction vector is used. The correction parameter is specified by the macrovariable $ETA3. The default value is $ETA3=$10^{-12}$.

$MEX2=0 - The basic termination criterion is used.
$MEX2=1 - The extended termination criterion for large-scale problems is used.

$MEX3=0 - The termination is used after a zero difference of gradients.
$MEX3=1 - The extrapolation with a constant stepsize is used after a zero difference of gradients.

$MEX3=2 - The extrapolation with an increasing stepsize is used after a zero difference of gradients.

The default values are $MEX=1, $MEX1=0, $MEX2=0, $MEX3=0.

The variable metric updates can be scaled. This possibility is specified by the macrovariable $MET1.

$MET1=0 - Scaling is suppressed.
$MET1=1 - Scaling is performed.

The default value is $MET1=0.

Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard, which guarantees that the new point lies in the region where the bundle model is valid. The default value is $XMAX=1000. Bundle variable metric methods for sparse sum of absolute values are sensitive to the values of parameters $ETA5 and $XMAX. Therefore, these values should be carefully chosen.

Bundle variable metric methods for sparse sum of absolute values use an auxiliary bundle of gradients, which serves for the initial stepsize determination in the line search subalgorithm. The size of this bundle is specified by the macrovariable $MB (the default value is $MB=20).

A possible specification (type-number) for bundle variable metric methods for sparse sum of absolute values is L-1.

3.25 Primal interior point methods for sparse sum of absolute values

If $MODEL='AA', then the objective function is the sum of absolute values. In this case, primal interior point methods [211] can be chosen by using the statement $FORM='SP'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Primal interior point variable metric methods generated from the driver template U1ASU1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates. The partitioned variable metric updates from the Broyden family [134] are used.

$CLASS='MN' - Primal interior point modified Newton methods generated from the driver template U2ASU1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='MN'.

If $CLASS='VM', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.

$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern terms of the Lagrangian function.

The default value is $MET5=1.

The macrovariable $MEP2 specifies restarts in the indefinite case.

$MEP2=0 - The Gill-Murray decomposition of the indefinite matrix is used.
$MEP2=1 - The unit matrix is used.

The default value is $MEP2=1.

The macrovariable $MEP3 determines a strategy for computation of the barrier parameter.

$MEP3=1 - A geometric sequence is used.
$MEP3=2 - A monotone sequence is used.
$MEP3=3 - A retarded monotone sequence is used.
$MEP3=4 - A retarded monotone sequence with stagnations is used.

The default value is $MEP3=4.

Primal interior point methods for sparse sum of absolute values use parameters specified by macrovariables $ETA4, $ETA5. The macrovariable $ETA4 determines the reduction of the barrier parameter. The default value is $ETA4=0.85. The macrovariable $ETA5 determines the minimum value of the barrier parameter. The default value is $ETA5=$10^{-8}$. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the sum of absolute values is well defined. The default value is $XMAX=1000.

Primal interior point methods for sparse sum of absolute values use either line search (if $TYPE='L') or trust region (if $TYPE='G') strategies. Possible specifications (type-decomposition-number) are these:

L-G-1,		L-M-1,
G-G-1,	G-B-1,	G-M-1,
		G-M-7.

The default choice is L-G-1.

3.26 Smoothing methods for sparse sum of absolute values

If $MODEL='AA', then the objective function is the sum of absolute values. In this case, smoothing methods can be chosen by using the statement $FORM='SM'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Smoothing variable metric methods generated from the driver template U1ASU2. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates. The partitioned variable metric updates from the Broyden family [134] are used.
$CLASS='MN' - Smoothing modified Newton methods generated from the driver template U2ASU2. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='MN'.

If $CLASS='VM', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.

$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern terms of the Lagrangian function.

The default value is $MET5=1.

The macrovariable $MEP2 specifies whether the values of exponentials are saved.

$MEP2=0 - The values of exponentials are not saved.
$MEP2=1 - The values of exponentials are saved.

The default value is $MEP2=0.

The macrovariable $MEP3 determines a strategy for computation of the smoothing parameter.

$MEP3=1 - A geometric sequence is used.
$MEP3=2 - A monotone sequence is used.
$MEP3=3 - A retarded monotone sequence is used.
$MEP3=4 - A simple retarded monotone sequence with stagnations is used.
$MEP3=5 - A special monotone sequence is used.

The default value is $MEP3=3.

Smoothing methods for sparse sum of absolute values use parameters specified by macrovariables $ETA4, $ETA5, $ETA6. The macrovariable $ETA4 determines the reduction of the smoothing parameter. The default value is $ETA4=0.95. The macrovariable $ETA5 determines the minimum value of the smoothing parameter. The default value is $ETA5=$10^{-8}$. The macrovariable $ETA6 determines the threshold value of the gradient norm for the switching in the computation of the smoothing parameter. The default value is $ETA6=$10^{-4}$, if $MEP3=5, or $ETA6=1, otherwise. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the sum of absolute values is well defined. The default value is $XMAX=1000.

Smoothing methods for sparse sum of absolute values use either line search (if $TYPE='L') or trust region (if $TYPE='G') strategies. Possible specifications (type-decomposition-number) are these:

L-G-1,		L-M-1,
G-G-1,	G-B-1,	G-M-1,
		G-M-7.

The default choice is L-G-1.

3.27 Recursive linear programming methods for dense minimax problems

If $MODEL='AM', then the objective function is the maximum of approximating functions or their absolute values. In this case, recursive linear programming methods [243] can be chosen by using the statements $FORM='SL' and $CLASS='LP'. Recursive linear programming methods, generated from the driver template U1MDU1, 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. A possible specification (type-number) for recursive linear programming methods is G-1.

3.28 Recursive quadratic programming methods for dense minimax problems

If $MODEL='AM', then the objective function is the maximum of approximating functions or their absolute values. In this case, recursive quadratic programming methods [144], [183] can be chosen by using the statement $FORM='SL' and $CLASS='VM' or $CLASS='MN'. These methods belong to the following classes:

$CLASS='VM' - Recursive quadratic programming variable metric methods generated from the driver template U1MDU1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates belonging to the Broyden family.

$CLASS='MN' - Recursive quadratic programming modified Newton methods generated from the driver template U2MDU1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='VM'.

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 [108].

If $TYPE='L', the special line search method ($MES=5), described in [183], can be used.

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

$NUMBER=1 - Dual projected gradient (range-space) method proposed in [182].
$NUMBER=2 - Primal projected gradient (range-space) method, which is a special implementation of the Newton projected gradient method.

Recursive quadratic programming variable metric methods use the same variable metric updates as methods with the choices $DECOMP='G' and $UPDATE='B' described in Section 3.8.1 (values from $MET=1,2,...,12 can be used). Similarly, recursive quadratic programming modified Newton methods correspond to the methods with the choice $DECOMP='G' described in Section 3.7 (the Gill-Murray decomposition is used).

Although the minimax problems can be solved by using bundle methods described in Sections 3.19 – 3.21, it is more efficient to use the recursive quadratic programming methods that utilize a special structure of the minimax problem.

All of the above methods can be used only for dense unconstrained or linearly constrained problems. Possible specifications (type-number) for recursive quadratic programming methods are these:

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

The default choice is L-1.

3.29 Primal interior point methods for sparse minimax problems

If $MODEL='AM', then the objective function is the maximum of approximating functions or their absolute values. In this case, primal interior point methods [209] can be chosen by using the statement $FORM='SP'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Primal interior point variable metric methods generated from the driver template U1MSU1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates. The partitioned variable metric updates from the Broyden family [134] are used.

$CLASS='MN' - Primal interior point modified Newton methods generated from the driver template U2MSU1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='MN'.

If $CLASS='VM', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern the terms of the Lagrangian function.

The default value is $MET5=1.

The macrovariable $MEP determines the particular barrier function.

$MEP=0 - The minimax function is used.
$MEP=1 - The logarithmic barrier function is used.
$MEP=2 - The Ben-Tal barrier function is used.
$MEP=3 - The composite barrier function is used.
$MEP=4 - The Carrol barrier function is used.

The default value is $MEP=1.

The macrovariable $MEP3 determines a strategy for computation of the barrier parameter.

$MEP3=1 - A geometric sequence is used.
$MEP3=2 - A monotone sequence is used.
$MEP3=3 - A retarded monotone sequence is used.
$MEP3=4 - A retarded monotone sequence with stagnations is used.

The default value is $MEP3=2.

Primal interior point methods for sparse minimax problems use parameters specified by macrovariables $ETA3, $ETA4, $ETA5. The macrovariable $ETA3 determines the precision of the solution to inner nonlinear equation. The default value is $ETA3=$10^{-6}$. The macrovariable $ETA4 determines the reduction of the barrier parameter. The default value is $ETA4=0.85. The macrovariable $ETA5 determines the minimum value of the barrier parameter. The default value is $ETA5=$10^{-10}$. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard which guarantees that the new point lies in the region where the minimax function is well defined. The default value is $XMAX=1000.

Primal interior point methods for sparse minimax problems use either line search (if $TYPE='L') or trust region (if $TYPE='G') strategies. Possible specifications (type-decomposition-number) are these:

L-G-1,		L-M-1,
G-G-1,	G-B-1,	G-M-1,
		G-M-7.

The default choice is L-G-1.

3.30 Smoothing methods for sparse minimax problems

If $MODEL='AM', then the objective function is the maximum of approximating functions or their absolute values. In this case, smoothing methods [282], [365] can be chosen by using the statement $FORM='SM'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Smoothing variable metric methods generated from the driver template U1MSU2. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates. The partitioned variable metric updates from the Broyden family [134] are used.

$CLASS='MN' - Smoothing modified Newton methods generated from the driver template U2MSU2. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='MN'.

If $CLASS='VM', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern the terms of the Lagrangian function.

The default value is $MET5=1.

The macrovariable $MEP2 specifies whether the values of exponentials are saved.

$MEP2=0 - The values of exponentials are not saved.
$MEP2=1 - The values of exponentials are saved.

The default value is $MEP2=0.

The macrovariable $MEP3 determines a strategy for the computation of the smoothing parameter.

$MEP3=1 - A geometric sequence is used.
$MEP3=2 - A monotone sequence is used.
$MEP3=3 - A retarded monotone sequence is used.
$MEP3=4 - A retarded monotone sequence with stagnations is used.
$MEP3=5 - A special monotone sequence is used.

The default value is $MEP3=2.

Smoothing methods for sparse minimax problems use parameters specified by macrovariables $ETA4, $ETA5, $ETA6. The macrovariable $ETA4 determines the reduction of the smoothing parameter. The default value is $ETA4=0.95. The macrovariable $ETA5 specifies the minimum value of the smoothing parameter. The default value is $ETA5=$10^{-8}$. The macrovariable $ETA6 determines the threshold value of the gradient norm for the switching in the computation of the smoothing parameter. The default value is $ETA6=$10^{-4}$, if $MEP3=5, or $ETA6=1, otherwise. Another important parameter is the maximum stepsize defined by the macrovariable $XMAX. The maximum stepsize is a safeguard, which guarantees that the new point lies in the region where the minimax function is well defined. The default value is $XMAX=1000.

Smoothing methods for sparse minimax problems use a line search strategy. A possible specification (type-decomposition-number) is L-G-2.

3.31 Recursive quadratic programming methods for dense nonlinear programming problems

Recursive quadratic programming methods for dense general 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 generated from the driver template U1FDN1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates.
$CLASS='MN' - Recursive quadratic programming modified Newton methods generated from the driver template U2FDN1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically.

The default value is $CLASS='VM'.

Recursive quadratic programming variable metric methods use the same variable metric updates as methods with the choices $DECOMP='G' and $UPDATE='B' described in Section 3.8.1 (values from $MET=1,2,...,12 can be used). Similarly, recursive quadratic programming modified Newton methods correspond to the methods with the choice $DECOMP='G' described in Section 3.7 (the Gill-Murray decomposition is used).

Recursive quadratic programming methods for dense general nonlinear programming problems are realized as line search methods ($TYPE='L') with the l_1-exact penalty function. They are like the methods proposed in [288]. The special line search method ($MES=5) for l_1-exact penalty function can be used successfully. The quadratic programming subproblem can be solved by using the two different methods:

$NUMBER=1 - Dual projected gradient (range-space) method (like the method proposed in [128]).
$NUMBER=2 - Primal projected gradient (range-space) method (like the method proposed in [102]), which is a special implementation of the Newton projected gradient method.

Possible specifications (type-number) for recursive quadratic programming methods for dense general nonlinear programming problems are L-1 and L-2. The default choice is L-1.

Recursive quadratic programming methods can be used for dense nonlinear programming problems with various objective functions. If we set $MODEL='AA' (sum of absolute values) or $MODEL='AM' (minimax), then an extended nonlinear programming problem containing extra variables is defined and solved.

3.32 Recursive quadratic programming methods for sparse equality constrained problems

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

$CLASS='VM' - Recursive quadratic programming variable metric methods generated from the driver template U1FSE1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates.

$CLASS='VL' - Recursive quadratic programming variable metric methods with limited memory based on compact representations of variable metric updates generated from the driver template U1FSE2. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited memory use several small-size matrices which are updated in each iteration in such a way that their product approximates the Hessian matrix of the Lagrangian function as precisely as possible [39].

$CLASS='MN' - Recursive quadratic programming modified Newton methods generated from the driver template U2FSE1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically. The sparsity pattern is required.

$CLASS='TN' - Recursive quadratic programming truncated Newton methods generated from the driver template U1FSE3. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. The sparsity pattern is not required.

The default value is $CLASS='MN'.

If $CLASS='VM' and $HESF='D', dense variable metric updates are used. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

If $CLASS='VM' and $HESF='S', the individual variable metric updates are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [248].

$UPDATE='B' - The partitioned variable metric updates from the Broyden family [134]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default values are $UPDATE='B' if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP' and $UPDATE='M' in the remaining cases. If $UPDATE='B', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.

$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern the terms of the Lagrangian function.

The default value is $MET5=1.

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

$TYPE='L' - Line search methods based on various merit functions or filter structures.
$TYPE='G' - Trust region methods. These methods use two direction determination subproblems [94], [166], [225]. The vertical subproblem, solved by using the dog-leg method, serves for a sufficient decrease of constraint violations. The horizontal subproblem, solved by a special realization of the conjugate gradient method, serves for minimization of a quadratic approximation of a particular merit function.

The default value is $TYPE='L'.

If $TYPE='L', various penalty functions or filter structures can be used for stepsize selection. The corresponding choice is determined by the macrovariable $MERIT:

$MERIT='P' - The penalty function is used.
$MERIT='M' - The Markov filter [340], [208] is used.
$MERIT='F' - The Fletcher-Leyffer filter [109], [340] is used.

The default value is $MERIT='P'. If $MERIT='P', additional choices specified by macrovariables $MEP, $MEP1, $MEP2 are possible.

Macrovariable $MEP determines the particular merit function:

$MEP=0 - No merit function is used. In this case, the line search option $KTERS=6 is implicitly assumed.
$MEP=1 - The Powell l_1 exact penalty function is used.
$MEP=2 - The l_2 augmented Lagrangian function is used.
$MEP=3 - The l_1 augmented Lagrangian function is used.
$MEP=4 - The Han l_1 exact penalty function is used.
$MEP=5 - The Schittkowski augmented Lagrangian function is used.

The default value is $MEP=2.

Macrovariable $MEP1 specifies the second order correction for overcoming the Maratos effect.

$MEP1=1 - The second order correction is suppressed.
$MEP1=2 - The second order correction is determined as a least squares solution of the shifted constraint system.

The default value is $MEP1=1.

Macrovariable $MEP2 specifies estimates of Lagrange multipliers at the beginning of each iteration.

$MEP2=1 - The initial estimate is taken from the previous iteration.

$MEP2=2 - The initial estimate is determined as a least squares solution of the first part of the Karush-Kuhn-Tucker system.

The default value is $MEP2=1.

If $TYPE='G', only the default specification $MERIT='P' is possible (with penalty functions determined by macrovariable $MEP as above). However, default values $MEP1=1 and $MEP2=1 are used in this case.

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

$DECOMP='K' - The direction vector is determined as a solution of the indefinite Karush-Kuhn-Tucker system [228].

$DECOMP='Z' - The direction vector is decomposed into two parts. The vertical part is computed directly from the constraint violation. The horizontal part, lying in the null-space, is computed iteratively by using a special realization of the conjugate gradient method. Instead of projecting into the null-space, either the augmented system or an orthogonal projection matrix, both determined from a range-space basis, are used [154].

$DECOMP='G' - The direction vector is determined directly from the Lagrangian multipliers which are determined iteratively by using the conjugate gradient method in the range space using the Schur complement. This choice can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

The default value is $DECOMP='K'.

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

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [99] of the indefinite Karush-Kuhn-Tucker system is used. This choice can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2, $MOS3 and $MOS4.

$NUMBER=4 - An inexact preconditioned conjugate residual (PCR) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1 and $MOS2.

$NUMBER=5 - An inexact symmetric preconditioned quasi-minimum residual (PQMR) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1 and $MOS2.

$NUMBER=6 - An inexact nonsymmetric preconditioned conjugate gradient squared (PCGS) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

The default value is $NUMBER=3.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.

$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.

$MOS1=2 - The precision guaranteeing descent direction is used together with the extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - Preconditioning is suppressed.

$MOS2=±1 - The indefinite preconditioner [228] based on a diagonal approximation of the Hessian matrix is used in the normal equation form.

$MOS2=2 - The indefinite preconditioner [228] based on a diagonal approximation of the Hessian matrix is used in the augmented system form.

$MOS2=±3 - The indefinite preconditioner [228] based on a diagonal perturbation of the Schur complement is used.

If $MOS2>0, a complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.

$MOS3=1 - The simple one-dimensional residual smoothing is used.

The default value is $MOS3=0.

Macrovariable $MOS4 specifies the choice of the initial direction.

$MOS4=0 - The zero initial direction is used.

$MOS4=1 - The vertical initial direction is used.

The default value is $MOS4=0.

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

$NUMBER=3 - An inexact null-space preconditioned conjugate gradient (NPCG) method for the determination of the horizontal direction, which uses a special determination of the required precision, is applied. A particular realization of this method depends on the specifications given by the macrovariables $MOS1 and $MOS2.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.

$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.

$MOS1=2 - The precision guaranteeing descent direction is used together with the extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a way for computing the preconditioner.

$MOS2=±1 - The preconditioner is computed by using the orthogonal projection matrix determined from a range-space basis.

$MOS2=±2 - The preconditioner is computed by using the augmented system determined from a range-space basis.

If $MOS2>0, the diagonal approximation of the Hessian matrix is used. If $MOS2<0, the unit approximation of the Hessian matrix is used. The default value is $MOS2=1.

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

$NUMBER=3 - The sparse Gill-Murray decomposition of the Hessian matrix of the Lagrangian function followed by a range-space smoothed conjugate gradient (RSCG) method for a positive definite range space system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

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

The default value is $NUMBER=3.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.

$MOS1=1 - The precision guaranteeing descent direction is used.

The default value is $MOS1=1.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - The preconditioning is suppressed.

$MOS2=±1 - The positive definite preconditioner [228] based on a diagonal approximation of the Hessian matrix is used.

$MOS2=±2 - The polynomial preconditioner [263] based on a decomposition of the normal equation is used.

If $MOS2>0, The complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.

$MOS3=1 - The simple one-dimensional residual smoothing is used.

The default value is $MOS3=1.

If $TYPE='G', only the default specifications $DECOMP='Z' and $NUMBER='3' are possible which correspond to the trust region conjugate gradient (TRCG) method. Macrovariable $MOS2 specifies a way for computing the projection step.

$MOS2=1 - The projection step is computed by using the orthogonal projection matrix determined from a range-space basis.

$MOS2=2 - The projection step is computed by using the augmented system determined from a range-space basis.

The default value is $MOS2=1.

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

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

The default choice is L-K-3. The choices L-K-1, L-G-3, and L-G-4 can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

3.33 Primal-dual interior point methods for sparse nonlinear programming problems

Primal-dual interior point methods for sparse nonlinear programming problems [205] are specified by the statement $FORM='SI'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Primal-dual interior point variable metric methods generated from the driver template U1FSI1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates.

$CLASS='VL' - Primal-dual interior point variable metric methods with limited memory based on compact representations of variable metric updates generated from the driver template U1FSI2. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited memory use several small-size matrices which are updated in each iteration in such a way that their product approximates the Hessian matrix of the Lagrangian function as precisely as possible [39].

$CLASS='MN' - Primal-dual interior point modified Newton methods generated from the driver template U2FSI1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically. The sparsity pattern is required.

$CLASS='TN' - Primal-dual interior point truncated Newton methods generated from the driver template U1FSI3. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. The sparsity pattern is not required.

The default value is $CLASS='MN'.

If $CLASS='VM' and $HESF='D', dense variable metric updates are used. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

If $CLASS='VM' and $HESF='S', the individual variable metric updates are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [248].

$UPDATE='B' - The partitioned variable metric updates from the Broyden family [134]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default values are $UPDATE='B' if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP' and $UPDATE='M' in the remaining cases. If $UPDATE='B', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern active terms of the Lagrangian function.
$MET5=3 - The updates concern all terms of the Lagrangian function.

The default value is $MET5=1.

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

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

Primal-dual interior point methods for sparse nonlinear programming problems are realized in the way which is specified by using the macrovariable $TYPE:

$TYPE='L' - Line search methods based on various merit functions or filter structures.
$TYPE='G' - Trust region methods. These methods use two direction determination subproblems [94], [166]. The vertical subproblem, solved by using the dog-leg method, serves for a sufficient decrease of constraint violations. The horizontal subproblem, solved by a special realization of the conjugate gradient method, serves for minimization of a quadratic approximation of a particular merit function.

The default value is $TYPE='L'.

If $TYPE='L', various penalty functions or filter structures can be used for stepsize selection. The corresponding choice is determined by the macrovariable $MERIT:

$MERIT='P' - The penalty function is used.
$MERIT='M' - The Markov filter [340], [208] is used.
$MERIT='F' - The Fletcher-Leyffer filter [109], [340] is used.
$MERIT='B' - The barrier filter [340] is used.

The default value is $MERIT='P'.

If $MERIT='P', then macrovariable $MEP determines the particular merit function:

$MEP=0 - No merit function is used. In this case, the line search option $KTERS=6 is implicitly assumed.
$MEP=1 - The augmented logarithmic barrier function is used.
$MEP=2 - The augmented Lagrangian function is used.

The default value is $MEP=1.

If $TYPE='G', only the default specification $MERIT='P' is possible. In this case, macrovariable $MEP determines the particular merit function:

$MEP=1 - The augmented logarithmic barrier function is used.
$MEP=2 - The simplified augmented logarithmic barrier function is used.

The default value is $MEP=1.

Other important specifications can be determined by using macrovariables $MEP1, $MEP2, $MEP3, $MEP4, $MEP5 and $MOP1, $MOP2.

Macrovariable $MEP1 determines an approach to the KKT-equations formulation.

$MEP1=1 - The primal formulation is used
$MEP1=2 - The primal-dual formulation is used.

The default value is $MEP1=2.

Macrovariable $MEP2 determines a strategy for computation of primal and dual stepsizes.

$MEP2=0 - Individual components of primal and dual stepsizes are handled separately.
$MEP2=±1 - Primal and dual stepsizes are not greater than their maximum values.
$MEP2=±2 - Primal and dual stepsizes are not greater than their maximum values, but the dual stepsize is not greater than the primal stepsize.

If $MEP2>0, the basic stepsize does not depend on the primal stepsize. If $MEP2<0, the basic stepsize is not greater than the primal stepsize (this is implicitly assumed when $MEP=1). The default value is $MEP2=0.

Macrovariable $MEP3 determines a strategy for computation of the barrier parameter.

$MEP3=±1 - The Shanno-Vanderbei formula [339] is used.
$MEP3=±2 - The El Bakry formula [101] is used.

If $MEP3>0, the basic strategy is used. If $MEP3<0, the monotone decreasing strategy is used. The default value is $MEP3=1.

Macrovariable $MEP4 determines a regularization strategy if gradients of constraint functions are nearly linearly dependent.

$MEP4=0 - Regularization is not used.
$MEP4=1 - Regularization described in [205] is used.

The default value is $MEP4=0. Macrovariable $MEP4 is used only if $TYPE='L'.

Macrovariable $MEP5 determines a regularization strategy for the ill-conditioned Hessian matrix.

$MEP5=0 - Regularization is not used.
$MEP5=1 - Regularization by the scaled unit matrix is used.
$MEP5=2 - Regularization by an inexact differentiation is used.

The default value is $MEP5=0. Macrovariable $MEP5 is used only if $TYPE='L'.

Macrovariable $MOP1 determines a strategy for the slack variable updates.

$MOP1=1 - Slack variables are determined by the line search.
$MOP1=2 - Slack variables are determined to satisfy constraints.

The default value is $MOP1=1.

Macrovariable $MOP2 determines a strategy for box constraint updates.

$MOP2=1 - Box constraints are determined by the line search.
$MOP2=2 - Initial box constraints are feasible.
$MOP2=3 - Box constraints are always feasible.

The default value is $MOP2=1.

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

$DECOMP='I' - The direction vector is determined as a solution of the indefinite Karush-Kuhn-Tucker system [228].

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

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [99] of the indefinite Karush-Kuhn-Tucker system is used. This choice can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

The default value is $NUMBER=3.

If $TYPE='L', other important specifications can be determined by using macrovariables $MES4, $MES5 and $MOS1, $MOS2, $MOS3.

Macrovariable $MES4 determines the KKT norm increasing strategy.

$MES4=0 - The KKT norm increasing is suppressed.

$MES4=1 - The KKT norm increases in the MES4 iterations.

The default value is $MES4=0.

Macrovariable $MES5 determines a restart strategy.

$MES5=0 - The restart is suppressed.

$MES5=1 - The uniform descent is assured.

The default value is $MES5=0. The value $MES5=1 is used only if $MERIT='P' and $MEP=0.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.

$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.

$MOS1=2 - The precision guaranteeing descent direction is used together with the extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - Preconditioning is suppressed.

$MOS2=±1 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the normal equation form.

$MOS2=2 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the augmented system form.

$MOS2=±3 - The indefinite preconditioner [205] based on the Gill-Murray decomposition of the approximation of the Hessian matrix is used.

If $MOS2>0, a complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.
$MOS3=1 - The simple one-dimensional residual smoothing is used.

The default value is $MOS3=0.

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

$NUMBER=3 - The trust region method that uses two direction determination subproblems [36], [37]. The vertical subproblem, solved by using the dog-leg method, serves for a sufficient decrease of constraint violations. The horizontal subproblem, solved by a special realization of the conjugate gradient method, serves for minimization of a quadratic approximation of a particular merit function. A realization of the inexact preconditioned conjugate gradient method depends on specifications given by the macrovariables $MOS1, $MOS2, $MOS3 and $MOS4.

Macrovariable $MOS1 specifies the choice of the penalty parameter.

$MOS1=1 - The constant penalty parameter is used.
$MOS1=2 - The penalty parameter is determined from the quadratic model.

The default value is $MOS1=1.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=1 - The preconditioner is computed by using the orthogonal projection matrix determined from a range-space basis.

Macrovariable $MOS3 determines a decision in case a negative curvature is detected.

$MOS3=1 - Additional conjugate gradient iterations are performed.
$MOS3=2 - A boundary step is determined.

The default value is $MOS3=1.

Macrovariable $MOS4 determines a norm for the trust region definition.

$MOS4=0 - The trust region is defined by slack variables.
$MOS4=1 - The trust region is defined by the Chebyshev l_∞ norm.
$MOS4=2 - The trust region is defined by the Euclidean l_2 norm.

The default value is $MOS4=2.

Possible specifications (type-decomposition-number) for primal-dual interior point methods for sparse equality and inequality constrained nonlinear programming problems are these:

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

The default choice is L-I-3. The choice L-I-1 can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

Primal-dual interior point methods can be used for sparse nonlinear programming problems with various objective functions. If we set $MODEL='AA' (sum of absolute values) or $MODEL='AM' (minimax), then an extended nonlinear programming problem containing extra variables is defined and solved.

3.34 Nonsmooth equation methods for sparse nonlinear programming problems

Nonsmooth equation methods for sparse nonlinear programming problems [207] are specified by the statement $FORM='SF'. These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Nonsmooth equation variable metric methods generated from the driver template U1FSF1. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates.

$CLASS='VL' - Nonsmooth equation variable metric methods with limited memory based on compact representations of variable metric updates generated from the driver template U1FSF2. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited memory use several small-size matrices which are updated in each iteration in such a way that their product approximates the Hessian matrix of the Lagrangian function as precisely as possible [39].

$CLASS='MN' - Nonsmooth equation modified Newton methods generated from the driver template U2FSF1. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically. The sparsity pattern is required.

$CLASS='TN' - Nonsmooth equation truncated Newton methods generated from the driver template U1FSF3. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. The sparsity pattern is not required.

The default value is $CLASS='MN'.

If $CLASS='VM' and $HESF='D', dense variable metric updates are used. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

If $CLASS='VM' and $HESF='S', the individual variable metric updates are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [248].

$UPDATE='B' - The partitioned variable metric updates from the Broyden family [134]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default values are $UPDATE='B' if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP' and $UPDATE='M' in the remaining cases. If $UPDATE='B', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - Updates concerns approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern active terms of the Lagrangian function.
$MET5=3 - The updates concern all terms of the Lagrangian function.

The default value is $MET5=1.

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

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

Nonsmooth equation methods for sparse nonlinear programming problems are realized only in the line search framework ($TYPE='L'). Various penalty functions or filter structures can be used for stepsize selection. The corresponding choice is determined by the macrovariable $MERIT:

$MERIT='P' - The penalty function is used.
$MERIT='M' - The Markov filter [340], [208] is used.
$MERIT='F' - The Fletcher-Leyffer filter [109], [340] is used.

The default value is $MERIT='P'.

If $MERIT='P', then macrovariable $MEP determines the particular merit function:

$MEP=0 - No merit function is used. In this case, the line search option $KTERS=6 is implicitly assumed.
$MEP=2 - The augmented Lagrangian function is used together with the residual function.

The default value is $MEP=2.

Other important specifications can be determined by using macrovariables $MEP2, $MEP3, $MEP4, $MEP5 and $MOP1, $MOP2.

Macrovariable $MEP2 determines the restriction on the Lagrange multipliers.

$MEP2=0 - Lagrange multipliers are not restricted.
$MEP2=1 - Lagrange multipliers are positive in all iterations.

The default value is $MEP2=0.

Macrovariable $MEP3, used only if $MERIT='P', determines a strategy for the use of the residual function.

$MEP3=0 - The augmented Lagrangian function is used as the merit function (the residual function is not used).
$MEP3=1 - The combination of the augmented Lagrangian function and the residual function is used as the merit function.
$MEP3=2 - The residual function is used as the merit function (with the exception of restarted iterations).

The default value is $MEP3=0.

Macrovariable $MEP4 determines a regularization strategy if gradients of constraint functions are nearly linearly dependent.

$MEP4=0 - Regularization is not used.
$MEP4=1 - Regularization described in [205] is used.

The default value is $MEP4=0.

Macrovariable $MEP5 determines a regularization strategy for the ill-conditioned Hessian matrix.

$MEP5=0 - Regularization is not used.
$MEP5=1 - Regularization by the scaled unit matrix is used.
$MEP5=2 - Regularization by an inexact differentiation is used.

The default value is $MEP5=0.

Macrovariable $MOP1 determines a strategy for the slack variable updates.

$MOP1=1 - Slack variables are determined by the line search.
$MOP1=2 - Slack variables are determined to satisfy constraints.

The default value is $MOP1=1.

Macrovariable $MOP2 determines a strategy for box constraint updates.

$MOP2=1 - Box constraints are determined by the line search.
$MOP2=2 - Initial box constraints are feasible.
$MOP2=3 - Box constraints are always feasible.

The default value is $MOP2=1.

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

$DECOMP='F' - The direction vector is determined as a solution of the indefinite Karush-Kuhn-Tucker system [228].

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

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [99] of the indefinite Karush-Kuhn-Tucker system is used. This choice can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

The default value is $NUMBER=3.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.
$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.
$MOS1=2 - Yhe precision guaranteeing descent direction is used together with the extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - Preconditioning is suppressed.
$MOS2=±1 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the normal equation form.
$MOS2=2 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the augmented system form.
$MOS2=±3 - The indefinite preconditioner [205] based on the Gill-Murray decomposition of the approximation of the Hessian matrix is used.

If $MOS2>0, a complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.
$MOS3=1 - The simple one-dimensional residual smoothing is used.

The default value is $MOS3=0.

Possible specifications (type-decomposition-number) for nonsmooth equation methods for sparse equality and inequality constrained nonlinear programming problems are these:

L-F-1,
L-F-3.

The default choice is L-F-3. The choice L-F-1 can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

Nonsmooth equation methods can be used for sparse nonlinear programming problems with various objective functions. If we set $MODEL='AA' (sum of absolute values) or $MODEL='AM' (minimax), then an extended nonlinear programming problem containing extra variables is defined and solved.

3.35 Primal-dual interior point methods for sparse problems with complementarity constraints

Primal-dual interior point methods for sparse problems with complementarity constraints [215] are specified by the statement $FORM='SI' (when $NCC>0). These methods, which are intended for large problems, belong to the following classes:

$CLASS='VM' - Primal-dual interior point variable metric methods. An approximation of the Hessian matrix of the Lagrangian function is updated in each iteration by using variable metric updates.

$CLASS='VL' - Primal-dual interior point variable metric methods with limited memory based on compact representations of variable metric updates. The number of VM steps is specified by the macrovariable $MF (the default value is $MF=5). Variable metric methods with limited memory use several small-size matrices which are updated in each iteration in such a way that their product approximates the Hessian matrix of the Lagrangian function as precisely as possible [39].

$CLASS='MN' - Primal-dual interior point modified Newton methods. The Hessian matrix of the Lagrangian function is computed in each iteration either analytically or numerically. The sparsity pattern is required.

$CLASS='TN' - Primal-dual interior point truncated Newton methods. These methods differ from modified Newton methods in that the directional derivatives are determined by the numerical differentiation instead of the sparse Hessian matrix multiplication. The sparsity pattern is not required.

The default value is $CLASS='MN'.

If $CLASS='VM' and $HESF='D', dense variable metric updates are used. Macrovariable $MET determines the variable metric update.

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=2 - The DFP method [83], [110] is used.
$MET=3 - The Hoshino method [153] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

If $CLASS='VM' and $HESF='S', the individual variable metric updates are specified by using the macrovariable $UPDATE:

$UPDATE='M' - The simple Marwil projection update [248].
$UPDATE='B' - The partitioned variable metric updates from the Broyden family [134]. These updates can only be used if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP'.

The default values are $UPDATE='B' if $MODEL='AF' or $MODEL='AQ' or $MODEL='AP' and $UPDATE='M' in the remaining cases. If $UPDATE='B', the particular variable metric method is specified by using macrovariables $MET, $MET1, $MET5. Macrovariable $MET determines the variable metric update.

$MET=1 - The partitioned BFGS method [29], [105], [127], [306] is used.
$MET=2 - The partitioned DFP method [83], [110] is used.
$MET=3 - The partitioned Hoshino method [153] is used.
$MET=4 - The partitioned safeguarded rank-one method [190] is used.

The default value is $MET=1. Macrovariable $MET1 determines scaling of variable metric updates [277].

$MET1=1 - No scaling is used.
$MET1=2 - The initial scaling [310] is used.
$MET1=3 - The controlled scaling [194] is used.
$MET1=4 - The interval scaling [221] is used.
$MET1=5 - The scaling in each iteration is used.

The default value is $MET1=3. Macrovariable $MET5 determines subjects of variable metric updates.

$MET5=0 - The updates concern approximating functions.
$MET5=1 - The updates concern approximating functions multiplied by the signs of the Lagrange multipliers.
$MET5=2 - The updates concern active terms of the Lagrangian function.
$MET5=3 - The updates concern all terms of the Lagrangian function.

The default value is $MET5=1.

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

$MET=1 - The BFGS method [29], [105], [127], [306] is used.
$MET=4 - The safeguarded rank-one method [190] is used.

The default value is $MET=1.

Primal-dual interior point methods for sparse problems with complementarity constraints are realized only in the line search framework ($TYPE='L'). Various penalty functions or filter structures can be used for stepsize selection. The corresponding choice is determined by the macrovariable $MERIT:

$MERIT='P' - The penalty function is used.
$MERIT='M' - The Markov filter [340], [208] is used.
$MERIT='F' - The Fletcher-Leyffer filter [109], [340] is used.
$MERIT='B' - The barrier filter [340] is used.

If $MERIT='P', then macrovariable $MEP determines the particular merit function:

$MEP=0 - No merit function is used. In this case, the line search option $KTERS=6 is implicitly assumed.
$MEP=1 - The augmented logarithmic barrier function is used.

The default value is $MEP=0.

Other important specifications can be determined by using macrovariables $MEP3, $MEP4 and $MEP5.

Macrovariable $MEP3 determines a strategy for computation of the barrier parameter.

$MEP3<0 - The standard Shanno-Vanderbei formula [339] is used.
$MEP3>0 - The special Shanno-Vanderbei formula [339] is used.

The default value is $MEP3=1.

Macrovariable $MEP4 determines a regularization strategy if gradients of constraint functions are approximately linearly dependent.

$MEP4=0 - Regularization is not used.
$MEP4=1 - Regularization described in [205] is used.

The default value is $MEP4=0.

Macrovariable $MEP5 determines a restart strategy.

$MEP5=0 - The restart is suppressed.
$MEP5=1 - The uniform descent is assured.

The default value is $MEP5=0. The value $MEP5=1 is used only if $MERIT='P' and $MEP=0.

There are two linear algebra approaches for the direction determination which are specified by the macrovariable $NUMBER:

$NUMBER=1 - An exact sparse Bunch-Parlett (BP) decomposition [99] of the indefinite Karush-Kuhn-Tucker system is used. This choice can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

$NUMBER=3 - An inexact preconditioned conjugate gradient (PCG) method for the indefinite Karush-Kuhn-Tucker system, which uses a special determination of the required precision, is applied. The particular realization of this method depends on specifications given by the macrovariables $MOS1, $MOS2 and $MOS3.

The default value is $NUMBER=3.

Macrovariable $MOS1 specifies the precision control and the choice of the penalty parameter.

$MOS1=0 - The precision control is suppressed.
$MOS1=1 - The precision guaranteeing descent direction is used together with the basic choice of the penalty parameter.
$MOS1=2 - The precision guaranteeing descent direction is used together with the extended choice of the penalty parameter.

The default value is $MOS1=0.

Macrovariable $MOS2 specifies a preconditioning technique.

$MOS2=0 - Preconditioning is suppressed.
$MOS2=±1 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the normal equation form.
$MOS2=2 - The indefinite preconditioner [215] based on a diagonal approximation of the Hessian matrix is used in the augmented system form.

If $MOS2>0, a complete Gill-Murray decomposition is used. If $MOS2<0, an incomplete Gill-Murray decomposition is used. The default value is $MOS2=1.

Macrovariable $MOS3 specifies residual smoothing of the conjugate gradient method.

$MOS3=0 - The residual smoothing is suppressed.
$MOS3=1 - The simple one-dimensional residual smoothing is used.

The default value is $MOS3=0.

Possible specifications (type-decomposition-number) for primal-dual interior point methods for sparse problems with complementarity constraints are these:

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

The default choice is L-I-3. The choice L-I-1 can be used only if $CLASS='MN' or $CLASS='VM' and $HESF='S' (if a sparsity pattern is available).

3.36 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 [137], use a stepsize control based on a local truncation error.

A solution to the initial value problem for ordinary differential equations can be stored for subsequent processing. The extent of the data stored is determined by using the macrovariable $MED.

$MED=0 - No data are stored.
$MED=1 - The data in all solution steps are stored.
$MED=2 - The data in equidistant mesh points are stored. The number of mesh points is specified by using the statement $NA=number_of_mesh_points in this case.

3.37 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 macrovariable $TYPE specifies the following types of methods (which have been already mentioned at the begining of Section 3):

$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='R' - Cubic regularization methods.

Now we will explain the specification $NUMBER. This explanation concerns only methods described in Sections 3.8 – 3.14.

3.37.1 Line search methods for direction determination

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

$NUMBER=1 - Direct methods for solving linear systems utilizing various matrix decompositions. In the sparse case, symbolic decomposition is determined before the optimization process is started. Thus numerical computations with known factors are carried out in the subsequent iterations.

$NUMBER=2 - Direct methods for solving linear systems are combined with the conjugate gradient method if $HESF='D'. The sparse Bunch-Parlett decomposition of the augmented system describing the linear least squares problem if $JACA='S'.

$NUMBER=3 - Inexact iterative methods for solving linear systems (specified below). The precision control is specified by the macrovariable $MOS. The preconditioning technique is specified by the macrovariable $MOS2.

$NUMBER=4 - Inexact iterative methods for solving linear systems (specified below). The precision control is specified by the macrovariable $MOS. The preconditioning technique is specified by the macrovariable $MOS2.

$NUMBER=5 - Inexact iterative methods for solving linear systems (specified below). The precision control is specified by the macrovariable $MOS. The preconditioning technique is specified by the macrovariable $MOS2.

If $NUMBER=1 and $HESF='D', the following matrix decompositions are possible:

$DECOMP='M' - The Gill-Murray decomposition [123] of a symmetric matrix if $MOS2=0. The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2.

$DECOMP='G' - The Gill-Murray decomposition [123] of a symmetric matrix.

$DECOMP='S' - The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2.

$DECOMP='R' - The Choleski decomposition of a symmetric positive definite matrix obtained from the sequential QR decomposition [147].

$DECOMP='B' - The Bunch-Parlett decomposition [34] or the Bunch-Kaufman decomposition [35] of a symmetric indefinite matrix.

$DECOMP='I' - The inverse of a symmetric matrix.

If $DECOMP='B', the macrovariable $MOS2 determines the method for indefinite symmetric matrix decomposition:

$MOS2=1 - The Bunch-Parlett decomposition with numerical permutations is used.

$MOS2=2 - The Bunch-Kaufman decomposition with numerical permutations is used.

$MOS2=3 - The Bunch-Parlett decomposition with double indexing is used.

The default value is $MOS2=1.

If $NUMBER=1 and $HESF='S', the following matrix decompositions are possible:

$DECOMP='M' - The sparse Gill-Murray decomposition of a symmetric matrix.

$DECOMP='G' - The sparse Gill-Murray decomposition of a symmetric matrix.

$DECOMP='B' - The sparse Bunch-Parlett decomposition [99], [100].

If $NUMBER=1 and $JACA='D', the following matrix decompositions are possible:

$DECOMP='A' - The orthogonal QR decomposition with permutations of a rectangular matrix.
$DECOMP='Q' - The orthogonal QR decomposition without permutations of a rectangular matrix.
$DECOMP='E' - The complete LU decomposition of a general square matrix.

The Jacobian matrix can be stored either rowwise (if $JASA='R') or columnwise (if $JASA='C'). The default values are $JASA='C' if $DECOMP='A' or $DECOMP='Q' and $JASA='R' if $DECOMP='E' (the specification $JASA='C' cannot be used for problems with linear approximating functions if $NAL>0). The macrovariable $MOS2 determines the technique of column permutations:

$MOS2=1 - The numerical permutations are used.
$MOS2=2 - The double indexing is used.

The default value is $MOS2=1. If $DECOMP='A', the macrovariable $MOS3 determines the type of QR decomposition:

$MOS3=1 - The incomplete QR decomposition is used.
$MOS3=2 - The complete QR decomposition [19] is used.

The default value is $MOS3=1. If $DECOMP='A', the macrovariable $MOS4 determines the strategy of pivoting:

$MOS4=1 - The basic pivoting is used.
$MOS4=2 - The basic pivoting with the incremental rank determinantion [17] is used.
$MOS4=3 - The rank revealing pivoting is used.

The default value is $MOS4=1. If $DECOMP='Q', the macrovariable $MOS3 determines the technique of the quasi-Newton update:

$MOS3=1 - The orthogonal matrix is not explicitly constructed. The unit matrix is updated.
$MOS3=2 - The orthogonal matrix is explicitly constructed and updated [82].

The default value is $MOS3=1.

If $NUMBER=1 and $JACA='S', the following matrix decompositions are possible:

$DECOMP='A' - The sparse orthogonal QR decomposition [336] of a rectangular matrix.
$DECOMP='E' - The sparse LU decomposition [85] of a general square matrix.

If $NUMBER=2 and $HESF='D', the following matrix decompositions are possible:

$DECOMP='M' - The Gill-Murray decomposition [123] of a symmetric matrix if $MOS2=0. The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2.
$DECOMP='G' - The Gill-Murray decomposition [123] of a symmetric matrix.
$DECOMP='S' - The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2.
$DECOMP='B' - The Bunch-Parlett decomposition [34] or the Bunch-Kaufman decomposition [35] of a symmetric indefinite matrix specified by the macrovariable $MOS2 as in case $NUMBER=1.

If $NUMBER=2 and $JACA='S', the following matrix decomposition is possible:

$DECOMP='A' - The sparse Bunch-Parlett decomposition of the augmented system describing the linear least squares problem [19], [99].

If $NUMBER=3 and $HESF='S', the following iterative method is used:

$DECOMP='M' - The conjugate gradient method [88] for solving symmetric linear systems.

If $NUMBER=3 and $JACA='S', the following iterative methods are used:

$DECOMP='A' - The CGLS method [279] for solving linear least squares problems.
$DECOMP='E' - The smoothed CGS method [333] for solving nonsymmetric linear systems.

If $NUMBER=4 and $JACA='S', the following iterative methods are used:

$DECOMP='A' - The LSQR method [279] for solving linear least squares problems.
$DECOMP='E' - The GMRES method [300] for solving nonsymmetric linear systems.

If $NUMBER=5 and $JACA='S', the following iterative method is used:

$DECOMP='E' - The smoothed BICGSTAB method [338] for solving nonsymmetric linear systems.

The iterative methods with $NUMBER=3,4,5 use macrovariables $MOS and $MOS2. The macrovariable $MOS determines the precision control strategy and has the following meaning:

$MOS=1 - The simple strategy is used for precision control.
$MOS=2 - The geometric decreasing strategy is used for precision control.
$MOS=3 - The harmonic decreasing strategy is used for precision control.

The default value is $MOS=3. The macrovariable $MOS2 determines the choice of a suitable preconditioner in the sparse case. If $MOS2<0, only the preconditioning is applied. If $MOS2>0, the preliminary solution obtained by the preconditioner is tested. If the preliminary solution is not sufficiently accurate, the preconditioned iterations follow. If $DECOMP='M' and $HESF='S', the following preconditioning techniques for the conjugate gradient method are possible:

$MOS2=0 - Preconditioning is suppressed.
$MOS2=±1 - The incomplete Gill-Murray decomposition (full implementation) is used.
$MOS2=±2 - The ILUT type decomposition (the first type) is used.
$MOS2=±3 - The ILUT type decomposition (the second type) is used.
$MOS2=±4 - The ILUT type decomposition (the third type) is used.
$MOS2=±5 - The incomplete Gill-Murray decomposition (simplified implementation) is used.
$MOS2=±6 - The SSOR preconditioner is used.

The default value is $MOS2=-1. If $DECOMP='E' and $JACA='S', the following preconditioning techniques for the CGS, GMRES and BICGSTAB methods are possible:

$MOS2=0 - Preconditioning is suppressed.
$MOS2=±1 - The incomplete LU decomposition is used.
$MOS2=±2 - The SSOR preconditioner is used.

The default value is $MOS2=-1.

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

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

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

3.37.2 Trust region methods for direction determination

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 (with $XDEL=0) is recommended. The trust region methods can be internally scaled. This way is very advantageous for nonlinear regression problems containing exponentials. The trust region scaling is specified by the macrovariable $MOS1.

$MOS1=1 - No scaling is performed.
$MOS1=2 - The scaling coefficients are derived from diagonal elements of the normal equation matrix [199].

The default value is $MOS1=1.

There are nine trust region methods:

$NUMBER=1 - The dog-leg methods utilizing various matrix decompositions. The individual dog-leg methods are specified by the macrovariable $MOS. If $MOS=1, the single dog-leg method [284] is used. If $MOS=2, the double dog-leg method [92] is used. If $MOS=3, the optimum dog-leg method [41] is used. The default value is $MOS=2. In the sparse case, the symbolic decomposition is determined before the iterative process is started. Thus numerical computations with known factors are carried out in the subsequent iterations.

$NUMBER=2 - The multiple dog-leg methods (combinations of dog-leg methods and conjugate gradient methods) [198] are supposed. The number of dog-leg steps is specified by the statement $MOS=number_of_steps. The default value is $MOS=5.

$NUMBER=3 - The inexact iterative trust region methods (specified below). The precision control is specified by the macrovariable $MOS in the same way as in line search methods. The preconditioning technique is specified by the macrovariable $MOS2 in the same way as in line search methods.

$NUMBER=4 - The inexact iterative trust region methods (specified below). The precision control is specified by the macrovariable $MOS in the same way as in line search methods. The preconditioning technique is specified by the macrovariable $MOS2 in the same way as in line search methods.

$NUMBER=5 - The inexact iterative trust region methods (specified below). The precision control is specified by the macrovariable $MOS in the same way as in line search methods. The preconditioning technique is specified by the macrovariable $MOS2 in the same way as in line search methods.

$NUMBER=6 - The Gould-Lucidi-Roma-Toint iterative trust region method [132] that uses the Lanczos process to obtain an approximation to the optimum locally constrained step (permitted only for $DECOMP='M'). The number of Lanczos steps is specified by the macrovariable $MOS3. The default value is $MOS3=100. If $HESF='S', the method can be preconditioned by using the incomplete Gill-Murray decomposition. This possibility is specified by the macrovariable $MOS2 in the same way as in line search methods.

$NUMBER=7 - The More-Sorensen optimum locally constrained trust region method [258].

$NUMBER=8 - The sequential subspace method [139] (permitted only for $DECOMP='M' and $HESF='S'). The quadratic function is minimized over a subspace which is adjusted in successive iterations to ensure convergence to an optimum. The subspace contains one step of SQP method [23]. This iteration is obtained by solving a linear system whose preconditioner is specified by the macrovariable $MOS2. If $MOS2=1, then the diagonal preconditioning is used, if $MOS2=2, then the incomplete Gill-Murray decompostition is used, and if $MOS2=3, then the SSOR preconditioning is used. The default velue is $MOS2=2.

$NUMBER=9 - The Sorensen trust region method [299] (permitted only for $DECOMP='M' and $HESF='S'). The trust region subproblem is recast as a parametrized eigenvalue problem. An optimal value of the parameter is computed and the optimal solution of the trust-region subproblem is found from the eigenvectors associated with the two smallest eigenvalues of the parameterized eigenvalue problem corresponding to the optimal parameter. The eigenpairs are computed using ARPACK Implicitly Restarted Lanczos Method [168]. The number of Lanczos steps is determined by the macrovariable $MOS3. The default value is $MOS3=100.

If $NUMBER=1 or $NUMBER=2 and $HESF='D', the following matrix decompositions are possible:

$DECOMP='M' - The Gill-Murray decomposition [123] of a symmetric matrix if $MOS2=0 or $HESF='S'. The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1 and $HESF='D'. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2 and $HESF='D'.

$DECOMP='G' - The Gill-Murray decomposition [123] of a symmetric matrix.

$DECOMP='S' - The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2.

$DECOMP='B' - The Bunch-Parlett decomposition [34] or the Bunch-Kaufman decomposition [35] of a symmetric indefinite matrix.

If $DECOMP='B', the macrovariable $MOS2 determines the method for indefinite symmetric matrix decomposition in the same way as in line search methods.

If $NUMBER=1 or $NUMBER=2 and $HESF='S', the following matrix decompositions are possible:

$DECOMP='M' - The sparse Gill-Murray decomposition of a symmetric matrix.

$DECOMP='G' - The sparse Gill-Murray decomposition of a symmetric matrix (this possibility cannot be used if $NUMBER=2).

$DECOMP='B' - The sparse Bunch-Parlett decomposition [99], [100] (this possibility cannot be used if $NUMBER=2).

If $NUMBER=1 or $NUMBER=2 and $JACA='D', the following matrix decompositions are possible:

$DECOMP='A' - The orthogonal QR decomposition with permutations of a rectangular matrix (this possibility cannot be used if $NUMBER=2).

$DECOMP='Q' - The orthogonal QR decomposition without permutations of a rectangular matrix.

$DECOMP='E' - The complete LU decomposition of a general square matrix.

The Jacobian matrix can be stored either rowwise (if $JASA='R') or columnwise (if $JASA='C') in the same way as in line search methods. The macrovariables $MOS2, $MOS3, $MOS4 have the same meaning as in line search methods.

If $NUMBER=1 or $NUMBER=2 and $JACA='S', the following matrix decompositions are possible:

$DECOMP='A' - The sparse orthogonal QR decomposition [336] of a rectangular matrix.

$DECOMP='E' - The sparse LU decomposition [85] of a general square matrix.

There is an exception. If $CLASS='QN', then the sparse Bunch-Parlett decomposition of the augmented system describing the linear least squares problem [19], [99] is used if $DECOMP='A'.

If $NUMBER=3 and $HESF='D' or $HESF='S', the following iterative method is used:

$DECOMP='M' - The Steihaug [323] and Toint [331] iterative trust region method.

If $NUMBER=3 and $JACA='S', the following iterative methods are used:

$DECOMP='A' - The CGLS trust region method [195].
$DECOMP='E' - The smoothed CGS trust region method [224].

If $NUMBER=4 and $HESF='D' or $HESF='S', the following iterative method is used:

$DECOMP='M' - The shifted Steihaug-Toint iterative trust region method [198], [204]. The number of Lanczos steps is specified by the macrovariable $MOS3. The default value is $MOS3=5.

If $NUMBER=4 and $JACA='S', the following iterative methods are used:

$DECOMP='A' - The LSQR trust region method [195].
$DECOMP='E' - The GMRES trust region method [224].

If $NUMBER=5 and $HESF='D' or $HESF='S', the following iterative method is used:

$DECOMP='M' - The combined conjugate gradient and Lanczos iterative trust region method [198]. The number of Lanczos steps is specified by the macrovariable $MOS3. The default value is $MOS3=5.

If $NUMBER=5 and $JACA='S', the following iterative method is used:

$DECOMP='E' - The smoothed BICGSTAB trust region method [224].

The method specified by the choice $NUMBER=6 (with $DECOMP='M') is described above.

The method specified by the choice $NUMBER=6 with $DECOMP='M' is described above. The iterative methods with $NUMBER=3,4,5,6 use macrovariables $MOS, $MOS2, $MOS4 and $MOS5. The macrovariable $MOS specifies the precision control as in line search methods. The macrovariable $MOS2 specifies the preconditioning technique as in line search methods. If $HESF='S', the macrovariable $MOS4 specifies the rejecting the preconditioner after the incomplete Gill-Murray decomposition.

$MOS4=0 - Preconditioning in both the ill-conditioned and the indefinite cases is suppressed.
$MOS4=1 - Preconditioning in the ill-conditioned case is suppressed.
$MOS4=2 - Preconditioning is always used.

The default value is $MOS4=0. The macrovariable $MOS5 specifies the realization of the incomplete Gill-Murray decomposition.

$MOS5=1 - The basic incomplete Gill-Murray decomposition is used.
$MOS5=2 - The modified incomplete Gill-Murray decomposition that uses an artificial vector is used.

The default value is $MOS5=1.

If $NUMBER=7, the following matrix decompositions are possible:

$DECOMP='M' - The Gill-Murray decomposition [123] of a symmetric matrix if $MOS2=0 and $HESF='D'. The Schnabel-Eskow decomposition [304] of a symmetric matrix with numerical permutations if $MOS2=1 and $HESF='D'. The Schnabel-Eskow decomposition [304] of a symmetric matrix with double indexing if $MOS2=2 and $HESF='D'. The sparse Gill-Murray decomposition of a symmetric matrix if $HESF='S'.

$DECOMP='A' - The special augmented rectangular matrix is used.

The methods specified by the choices $NUMBER=8 and $NUMBER=9 (with $DECOMP='M') are described above.

3.37.3 Other methods for direction determination

Except for the line search and the trust region methods, the UFO system contains three special methods suitable especially for nonlinear least-squares problems:

$TYPE='T' - Special trust region methods for nonlinear least squares problems.
$TYPE='M' - Modified Marquardt methods for nonlinear least squares problems.
$TYPE='R' - Cubic regularization methods.

If $TYPE='T', only specifications $NUMBER=1, $NUMBER=2 and $NUMBER=7 can be used. These specifications have the same meaning as in case $TYPE='G', but the implementation is simpler (if $NUMBER=7, the simplified optimum locally constrained trust region method [199] is used).

If $TYPE='M', only specification $NUMBER=1 can be used. In this case, the modified Marquardt method proposed by Fletcher [103] is applied.

If $TYPE='R', only specification $NUMBER=7 can be used. In this case, the optimum variant of the cubic regularization method described in [42], [43] is applied.

3.38 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 four 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.
$SEARCH='N' - Special nonmonotone line search methods.
$SEARCH='H' - Special line search methods described in [141], which are suitable for conjugate gradient methods.

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

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

$MES=1 - The uniformly increasing extrapolation or bisection interpolation is used.
$MES=2 - Two point quadratic extrapolation or interpolation is used.
$MES=3 - Three point quadratic extrapolation or interpolation is used.
$MES=4 - Three point cubic extrapolation or interpolation is used.
$MES=5 - Special extrapolation or interpolation is used based on the special form of the objective function.

The default value is $MES=4.

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

$MES=1 - The uniformly increasing extrapolation or bisection interpolation is used.
$MES=2 - Quadratic extrapolation or interpolation (with one directional derivative) is used.
$MES=3 - Quadratic extrapolation or interpolation (with two directional derivatives) is used.

$MES=4 - Cubic extrapolation or interpolation [83] is used.
$MES=5 - Homogeneous extrapolation or interpolation [19] is used.
$MES=6 - Conic extrapolation or interpolation [19] is used.

The default value is $MES=4.

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

$MES1=1 - Constant extrapolation is used.
$MES1=2 - Extrapolation specified by the macrovariable $MES is used.
$MES1=3 - Extrapolation is suppressed.

$MES2=1 - Standard line search termination criterion is used.
$MES2=2 - Special termination criterion for nonconvex functions is used.
$MES2=3 - Line search is terminated after at least two function evaluations.

$MES3=1 - Safeguard against rounding errors is suppressed.
$MES3=2 - The first level of safeguard is used.
$MES3=3 - The second level of safeguard is used.

The default values are $MES1=2, $MES2=2, $MES3=2.

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

$KTERS<0 - The nonmonotone line search procedure proposed in [135] is used. The absolute value of the macrovariable $KTERS, which cannot be greater than 10, gives the number of nonmonotone steps.
$KTERS=1 - The perfect stepsize. The relative precision of the stepsize parameter is given by the value $EPS3 .
$KTERS=2 - The Goldstein stepsize [129]. The termination precision is given by the value $EPS1.
$KTERS=3 - The weak Wolfe stepsize [55]. The termination precision is given by the values $EPS1 and $EPS2.
$KTERS=4 - The strong Wolfe stepsize [107]. The termination precision is given by the values $EPS1 and $EPS2.
$KTERS=5 - The Armijo stepsize [10]. The termination precision is given by the value $EPS1.
$KTERS=6 - The first stepsize. The stepsize selection is terminated after the first function evaluation.

The default values are $KTERS=5 if $KDS=0 or $KTERS=3 if $KDS=1.

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

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

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

$INITS=±1 - Values $c_1 = 1$ and $c_2 = 0$ are used.
$INITS=±2 - Values $c_1 = 1$ and $c_2 = 4$ are used.
$INITS=±3 - Values $c_1 = 1$ and $c_2 = 2$ are used.
$INITS=±4 - Values $c_1 = 0$ and $c_2 = 2$ are used.

The default value depends on the optimization method used. Usually $INITS=-3 for the nonlinear conjugate gradient methods and $INITS=2 for the variable metric or the modified Newton methods.

Trust region methods are 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 [276].

Trust region methods are also influenced by using the macrovariable $KTERS. If $KTERS<0, then the nonmonotone trust region procedure proposed in [89] is used. The absolute value of the macrovariable $KTERS, which cannot be greater than 10, gives the number of nonmonotone steps.

3.39 Methods for numerical differentiation

The UFO system computes derivatives of the model function (approximating functions, constraint functions) numerically whenever they are not given analytically. In this case, the UFO preprocessor generates a corresponding part of the UFO source program. The main problem of a numerical differentiation is a difference determination which has to be chosen in such a way that the total influence of both the cancellation and the roundoff error is as small as possible. There are three possibilities in the UFO system which are distinguished by using the macrovariable $MCG:

$MCG=0 - The simple difference determination described in [93] is used.
$MCG=1 - The optimum difference determination proposed in [126] is used.
$MCG=2 - The optimum difference determination proposed in [324] is used.

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

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

If either $MODEL='DF' or $MODEL='DQ', the objective function is computed from the solution of an initial value problem for ordinary differential equations. The initial value problem is solved and the integral criterion is evaluated by using integration methods specified by the macrovariable $SOLVER as described in Section 3.36. If the partial derivatives of all the functions used are given analytically, the gradient of the objective function is computed by integration methods. There are two possibilities specified by the macrovariable $SYSTEM:

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

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

3.41 Global optimization methods

Global optimization methods are used if $EXTREM='G' is specified. The global optimization methods use local optimization methods for finding local minima. Therefore, the particular local optimization method has to be chosen by using the macrovariables $CLASS, $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 by using clustering algorithms to determine basins of attraction (clusters) of the individual minima. The basins of attraction (clusters) obtained are not searched repeatedly.

$GCLASS=4 - Multi-level methods. Modern stochastic methods which involve a combination of sampling and local search techniques. These methods combine strong theoretical properties with an attractive computational behavior. These methods are simpler but more efficient than the clustering methods.

The default value is $GCLASS=4.

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

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

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

$GTYPE=3 - Modified multi-start methods. Random points, distributed uniformly in a given region, are generated and local minimization is started whenever a point with a lower function value than that just obtained is found.

$GTYPE=4 - Bayesian reduced multi-start methods [20]. Random samples of points are repeatedly generated. Every random sample is reduced and local minimization is started from all points belonging to the reduced sample. The local minima obtained are compared and selected. This process is repeated while the Bayesian termination criterion is not satisfied.

The default value is $GTYPE=4.

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

$GTYPE=1 - Tunneling function methods [169]. 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 with a function value equal to or lower than the starting point is found.

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

$GTYPE=3 - Filled function methods [117], [118]. The idea of these 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 nor saddle points in any basin of a higher minimizer of the objective function, but it does have a minimizer or a saddle point in a basin of a lower minimizer of the objective function.

The default value is $GTYPE=2.

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

$GTYPE=1 - Density clustering method [21]. 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 seen point, are added to the cluster.

$GTYPE=2 - Single linkage clustering method [21]. In this case, the next two clusters to be merged are those for which the distance between the nearest points is the smallest. When this distance becomes larger than the threshold distance, the procedure is stopped. Starting with each point in a separate cluster, the points at distances smaller than the threshold distance are linked. A cluster is recognized as a set of points linked together.

The default value is $GTYPE=2.

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

$GTYPE=1 - Multi-level single linkage method [298]. In this case, the function values of the sample points are used in a very simple manner to obtain a very powerful method. The local search procedure is applied to every sample point, except if there is another sample point within the critical distance which has a smaller function value. Clusters can be constructed by associating a point with a local minimum if there exists a chain of points linking it to that minimum. This is done so that the distance between each successive pair is, at most, equal to the critical distance and the function value is decreasing along the chain. A point in this way could be assigned to more than one minimum.

$GTYPE=2 - Multi-level mode analysis method [298]. This method is a generalization of the mode analysis method. The region is partitioned into cells. After the sample reduction, it is determined which cells contain enough points to be "full". For each full cell the function value of the cell is defined to be equal to the smallest function value of any of the sample points in the cell. Finally, for every full cell, local minimization is applied except if a cell has a neighboring cell which is full and has a smaller function value.

$GTYPE=3 - Modified multi-level single linkage method. This is a multi level single linkage method with some modifications which are described in [298].

The default value is $GTYPE=3.

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

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

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

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

4 Input possibilities in the UFO system

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

4.1 The UFO control language

The form of the UFO source program can be determined by using the statements of the UFO control language. The UFO control language is based on the batch editing language (BEL) [327] described in Appendix B. The UFO control language contains four types of instructions:

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

Standard Fortran 77 instructions used in the UFO control language have some extensions and limitations. The main extension is the free format. The instructions need not have a limited length, they can be written everywhere in the input file and if they are written in the same line, the character ';' is used to separate the instructions. The continuation of an instruction is specified by the character '& '. The main limitation concerns the application of instructions in the UFO source program. Therefore, statement numbers greater than 9999 cannot be used, comments can be introduced by the character $'*'$ only and the only continuation character can be '& '. At the same time, 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 '$' and are supposed to be of the type character. Their values are always in the form of a string of characters which can be sometimes interpreted as an integer or a real or a logical constant. The chief significance of the macrovariables is their use in substituting their values for their names in the Fortran 77 statements. In this case, we place the macrovariable (beginning with '$') in the text, but if it is followed by a letter or a digit we have to use brackets. For example if we write

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

or

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

or

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

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

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

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

2. Text insertion: If we write

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

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

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

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

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

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

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

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

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

   or

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

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

```
$DO(I=1,NF,1)
  $DO(J=1,NC,1)
    CALL SUB($I,$J,$I.0D2+$J.0D0)
  $ENDDO
$ENDDO
```

then the UFO preprocessor generates the sequence

```
CALL SUB(1,1,1.0D2+1.0D0)
CALL SUB(1,2,1.0D2+2.0D0)
CALL SUB(1,3,1.0D2+3.0D0)
CALL SUB(2,1,2.0D2+1.0D0)
CALL SUB(2,2,2.0D2+2.0D0)
CALL SUB(2,3,2.0D2+3.0D0)
```

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

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

then the UFO preprocessor generates the sequence

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

7. File substitutions: Suppose we have a file with a name file_name.extension. Then, we can include it into the UFO source program by using the macroinstructions

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

or

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

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

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

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

9. Comments: Every line beginning with $REM is a comment line, which is ignored by the macroprocessor.

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

We have described basic possibilities of the UFO control language that are sufficient for preparing the batch input file. More details are given in subsequent chapters and especially in Appendix B. Usually, the macrovariable $STANDARD is used if one optimization problem is solved by one optimization method. The following three examples show possibilities how to use various methods for solving various optimization problems. The first example demonstrates the use of the UFO control language for solving one optimization problem by using two different optimization methods: the robust but less efficient pattern-search method ($CLASS='HM', $TYPE='P') and the superlinearly convergent variable metric method with numerical computation of gradients ($CLASS='VM'). The input template has the form:

```
$REM +-------------------------------------------------------------+
$REM +             MODEL DESCRIPTION                               +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  X(1)=-1.2D0; X(2)= 1.0D0
$ENDSET
$SET(FMODELF)
  FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
```

```
$NF=2
$REM : output specifications
$LOUT=2
$MOUT=1
$NOUT=1

$REM +-------------------------------------------------------------+
$REM +             START OF THE PROGRAM GENERATION                 +
$REM +-------------------------------------------------------------+
$BATCH;  $REM : the batch mode
$GLOBAL; $REM : global declarations
$TSTART; $REM : start of time measurement

$REM +-------------------------------------------------------------+
$REM +             THE FIRST METHOD                                +
$REM +-------------------------------------------------------------+
$REM : parameters of the first method
$XDEL='1.0$P-2'
$TOLB='1.0$P-2'
$MIT=800
$MFV=800
$CLASS='HM'
$TYPE='P'
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation
$METERASE; $REM : clearing parameters of the first method

$REM +-------------------------------------------------------------+
$REM +             THE SECOND METHOD                               +
$REM +-------------------------------------------------------------+
$REM : parameters of the second method
$HESF='D'
$TOLX='1.0$P-16'
$TOLF='1.0$P-16'
$TOLG='1.0$P-8'
$TOLB='1.0$P-16'
$MIT=800
$MFV=800
$CLASS='VM'
$TYPE='L'
$DECOMP='I'
$NUMBER=1
$METHOD; $REM : method specification and generation

$TSTOP;  $REM : print of the CPU time
$END; $REM : end of the program generation
```

The text output file follows:

```
 CLASS = HM - PN1   UPDATE = N   MODEL = FF   HESF = N   NF =       2
   0 NIT=   53 NFV=  222 NDC=    0  FV BOUND  F= 0.5266202783E-02
 FF =  0.5266202783D-02
```

```
 X  =  0.9322729460D+00  0.8717390863D+00
 CLASS = VM - LI1   UPDATE = B   MODEL = FF   HESF = D   NF =       2
   0 NIT=   12 NFV=   42 NFG=    0  FV BOUND  F= 0.1936407754E-20 G=0.607D-09
 FF =  0.1936407754D-20
 X  =  0.1000000000D+01  0.1000000000D+01
 TIME= 0:00:00.00
```

The second example demonstrates the use of the UFO control language for solving three similar optimization problems by the same optimization method. All problems concern nonlinear approximation (sum of values and sum of squares) with the same number of variables. The input template has the form:

```
$REM +-------------------------------------------------------------+
$REM +             GLOBAL PARAMETERS                               +
$REM +-------------------------------------------------------------+
$NF=100
$NA=500
$MA=2000
$M=9000
$REM : output specifications
$LOUT=2
$MOUT=1
$NOUT=0

$REM +-------------------------------------------------------------+
$REM +             METHOD SELECTION                                +
$REM +-------------------------------------------------------------+
$REM : parameters of the method
$TOLX='1.0$P-10'
$TOLF='1.0$P-15'
$TOLG='1.0$P-5'
$JACA='S'
$HESF='S'
$MIT=800
$MFV=1200
$CLASS='VM'
$TYPE='L'
$DECOMP='M'
$NUMBER=3
$UPDATE='B'
$MOS2=0
$NEXT=1

$REM +-------------------------------------------------------------+
$REM +             START OF THE PROGRAM GENERATION                 +
$REM +-------------------------------------------------------------+
$BATCH;  $REM : the batch mode $GLOBAL; $REM : global declarations




$REM +-------------------------------------------------------------+
$REM +             THE FIRST MODEL                                 +
$REM +-------------------------------------------------------------+
```

```
$MODEL='AF'
$SET(INPUT)
  CALL EIUB14(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU14(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU14(NF,KA,X,GA,NEXT)
$ENDSET
$COLLECTION='Y'; $REM : cycle for testing
$NEXT=3; $REM : number of steps in the cycle
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation
$MODERASE; $REM : clearing parameters of the first model

$REM +-------------------------------------------------------------+
$REM +             THE SECOND MODEL                                +
$REM +-------------------------------------------------------------+
$MODEL='AQ'
$SET(INPUT)
  CALL EIUB15(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU15(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU15(NF,KA,X,GA,NEXT)
$ENDSET
$NEXT=1; $REM : THE PROBLEM NUMBER
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$TSTART; $REM : start of time measurement
$METHOD; $REM : method specification and generation
$TSTOP;  $REM : print of the CPU time
$MODERASE; $REM : clearing parameters of the second model

$REM +-------------------------------------------------------------+
$REM +             THE THIRD MODEL                                 +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  CALL EIUB18(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU18(NF,KA,X,FA,NEXT)
$ENDSET


$SET(GMODELA)
  CALL EAGU18(NF,KA,X,GA,NEXT)
$ENDSET
```

```
$COLLECTION='Y'; $REM : cycle for testing
$NEXT=5; $REM : number of steps in the cycle
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation

$END; $REM : end of the program generation
```

The text output file follows:

```
 CLASS = VM - LM3   UPDATE = B   MODEL = AF   HESF = S   NF =     100
   1 NIT=  303 NFV=  364 NFG=  364  GRAD TOL  F= 0.3916196246E-14 G=0.229D-05
   2 NIT=  104 NFV=  168 NFG=  168  GRAD TOL  F=  10.69009184     G=0.993D-06
   3 NIT=   35 NFV=   38 NFG=   38  GRAD TOL  F= 0.5708941253E-10 G=0.567D-05
 TOTAL   NIT=     442   NFV=     570   NFG=     570   NDC=       0 *     3
         NCG=    9078   NLS=     125   NSC=       0   NUP=   25779
         NRS=       3   NMX=       0   NAD=       0   NRM=       0
 TIME= 0:00:00.09
 CLASS = VM - LM3   UPDATE = B   MODEL = AQ   HESF = S   NF =     100
   1 NIT=  306 NFV=  389 NFG=  389  GRAD TOL  F= 0.5452508290E-14 G=0.161D-05
 TIME= 0:00:00.08
 CLASS = VM - LM3   UPDATE = B   MODEL = AQ   HESF = S   NF =     100
   1 NIT=   38 NFV=   40 NFG=   40  GRAD TOL  F= 0.3359765546E-10 G=0.268D-05
   2 NIT=   34 NFV=   36 NFG=   36  GRAD TOL  F= 0.4552663103E-12 G=0.919D-06
   3 NIT=    4 NFV=    5 NFG=    5  GRAD TOL  F= 0.8693979881E-14 G=0.375D-07
   4 NIT=   12 NFV=   16 NFG=   16  GRAD TOL  F=  7.086281648     G=0.655D-06
   5 NIT=   69 NFV=  132 NFG=  132  GRAD TOL  F=  320.3198802     G=0.940D-05
 TOTAL   NIT=     157   NFV=     229   NFG=     229   NDC=       0 *     5
         NCG=    2517   NLS=      67   NSC=       0   NUP=    8284
         NRS=       5   NMX=       0   NAD=       0   NRM=       0
```

The third example demonstrates the use of the UFO control language for solving one optimization problem by using two different models: general objective function and sum of squares. Three optimization methods: the robust but less efficient heuristic pattern-search method ($CLASS='HM', $TYPE='P'), the variable metric method with numerical computation of gradients ($CLASS='VM') and the modified Gauss-Newton method with numerical computation of Jacobian matrices ($CLASS='GN', $UPDATE='F') are used. The input template has the form:

```
$REM +-------------------------------------------------------------+
$REM +             GLOBAL PARAMETERS                               +
$REM +-------------------------------------------------------------+
$REM : declaration of the user variables
$FLOAT W,WA,WB,WC,WM
$REM : output specifications
$LOUT=2
$MOUT=1
$NOUT=1



$REM +-------------------------------------------------------------+
$REM +             START OF THE PROGRAM GENERATION                 +
$REM +-------------------------------------------------------------+
```

```
$BATCH;  $REM : the batch mode
$GLOBAL; $REM : global declarations

$REM +-------------------------------------------------------------+
$REM +             THE FIRST MODEL                                 +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  X(1)=ONE ; X(2)=TWO ; X(3)=ONE
  X(4)=ONE ; X(5)=ONE ; X(6)=ONE
  XMAX=TEN
  FMIN=ZERO
$ENDSET
$SET(FMODELF)
  FF=ZERO
  DO 2 KA=1,20
    W=$DBLE(KA)/TEN
    WA=EXP(-W*X(1))
    WB=EXP(-W*X(2))
    WC=EXP(-W*X(3))
    WM=EXP(-W)-FIVE*EXP(-TEN*W)+THREE*EXP(-FOUR*W)
    FF=FF+(X(4)*WA-X(5)*WB+X(6)*WC-WM)**2
  2 CONTINUE
    FF=HALF*FF
$ENDSET
$NF=6

$REM +-------------------------------------------------------------+
$REM +             THE FIRST METHOD                                +
$REM +-------------------------------------------------------------+
$XDEL='1.0$P-2'
$TOLB='1.0$P-1'
$MIT=800
$MFV=8000
$CLASS='HM'
$TYPE='P'
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation
$METERASE; $REM : clearing parameters of the first method

$REM +-------------------------------------------------------------+
$REM +             THE SECOND METHOD                               +
$REM +-------------------------------------------------------------+
$TOLX='1.0$P-8'
$TOLF='1.0$P-8'
$TOLG='1.0$P-2'
$TOLB='1.0$P-2'
$MIT=800
$MFV=8000
$HESF='D'
$CLASS='VM'
$TYPE='L'
```

```
$DECOMP='I'
$NUMBER=1
$METHOD; $REM : method specification and generation
$METERASE; $REM : clearing parameters of the second method
$MODERASE; $REM : clearing parameters of the first model

$REM +---------------------------------------------------------------+
$REM +              THE SECOND MODEL                                 +
$REM +---------------------------------------------------------------+
$SET(INPUT)
  DO 3 KA=1,NA
    W=$DBLE(KA)/TEN
    AM(KA)=EXP(-W)-FIVE*EXP(-TEN*W)+THREE*EXP(-FOUR*W)
  3 CONTINUE
$ENDSET
$SET(FMODELA)
  W=$DBLE(KA)/TEN
  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
$NA=20
$NAL=0
$KBA=1
$MODEL='AQ'

$REM +---------------------------------------------------------------+
$REM +              THE THIRD METHOD                                 +
$REM +---------------------------------------------------------------+
$TOLX='1.0$P-16'
$TOLF='1.0$P-16'
$TOLG='1.0$P-8'
$TOLB='1.0$P-16'
$MIT=800
$MFV=8000
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=7
$UPDATE='F'
$JACA='D'
$HESF='D'
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation

$END; $REM : end of the program generation
```

The text output file follows:

```
 CLASS = HM - PN1   UPDATE = N   MODEL = FF   HESF = N   NF =       6
   0 NIT=   18 NFV=  176 NDC=    0  FV BOUND  F= 0.9758203329E-01
```

```
FF =  0.9758203329D-01
X  =  0.1577350877D+01  0.3818216079D+01  0.1711759499D+01  0.1324516197D+01
      0.1282806727D+01  0.1275838294D+01
CLASS = VM - LI1   UPDATE = B   MODEL = FF   HESF = D   NF =       6
  0 NIT=    9 NFV=   72 NFG=    0  GRAD TOL  F= 0.5384948810E-01 G=0.740D-02
FF =  0.5384948810D-01
X  =  0.2184135468D+01  0.5676842761D+01  0.1972958227D+01  0.1753256141D+01
      0.2957973274D+01  0.1724924127D+01
CLASS = GN - GM7   UPDATE = F   MODEL = AQ   HESF = D   NF =       6
  0 NIT=   14 NFV=  105 NFG=    0  FV BOUND  F= 0.3963563826E-30 G=0.171D-14
F  =  0.3963563826D-30
X  =  0.4000000000D+01  0.1000000000D+02  0.1000000000D+01  0.3000000000D+01
      0.5000000000D+01  0.1000000000D+01
```

4.2 The batch mode

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

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

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

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

The following combinations are possible:

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

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

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

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

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

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

A simple example of a batch input file was shown in Section 1.4. We repeat it here with some explanations:

```
$REM : model specifications
$SET(INPUT)
  X(1)=-1.2D0; X(2)= 1.0D0
$ENDSET
$SET(FMODELF)
  FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
$NF=2
$REM : the default method is used
$REM : print specifications
```

```
$MOUT=1
$NOUT=1
$REM : the batch mode is used
$BATCH
$REM : the standard form of the source program is used
$STANDARD
```

By using the macrovariable $INPUT, we specify the initial values of variables $x_1 = -1.2$ and $x_2 = 1.0$. By using the macrovariable $FMODELF, we specify the model function value (the model function gradient is not specified, it will be computed numerically). The macrovariable $NF defines the number of variables and $MOUT, $NOUT are print specifications. The macroinstruction $BATCH switches the mode to the batch mode. The macroinstruction $STANDARD defines the standard form of the UFO source program. Descriptions of more complicated problems are shown in Chapter 7.

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

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

or

```
$SET(FMODELF)
  $INCLUDE('FVAL.FOR')
$ENDSET
```

or

```
$SET(FMODELF)
  $SUBST('FVAL.FOR')
$ENDSET
```

The last possibility is useful if the model function value specification is written in a free format or it contains the BEL macroinstructions.

If we need to utilize user supplied subroutines, we can include them into the UFO source program by using the macrovariable $SUBROUTINES:

```
$SET(SUBROUTINES)
  user supplied subroutines
$ENDSET
```

In this case, some exceptions laid on the text of user supplied subroutines forced by the UFO preprocessor have to be satisfied. All comments have to begin with the character '*', the continuation line has to begin with the character '&', the character '$' has to be replaced by '$$' and the character ';' does not have to be present.

The batch input file should also contain optimization method selection. Fortunately, this selection is not critical since the optimization method can be chosen automatically by using knowledge bases contained in the UFO system templates. Here we will only demonstrate some possibilities. The following macrovariables have the greatest influence on the optimization method selection (see Chapter 3):

$FORM	- Form of optimization methods (recursive quadratic programming, primal interior point, primal-dual interior point, nonsmooth equarions).
$CLASS	- Class of optimization methods (heuristic, conjugate gradient, variable metric, variable metric with limited storage, modified Newton, truncated Newton, Gauss-Newton, quasi-Newton, quasi-Newton with limited storage, proximal bundle, bundle-Newton).
$TYPE	- Type of optimization methods (line search, trust region, Levenberg-Marquardt).

$DECOMP - Type of matrix decomposition (original matrix, Choleski decomposition, inversion).
$NUMBER - Individual methods for direction determination (various direct, various iterative).
$UPDATE - Type of variable metric or quasi-Newton update.
$MERIT - Type of line search merit function (penalty, exact penalty, filter).

A more detailed description of these choices together with other choices ($MET, $MET1, $MET2, $MET3, $MES, $MES1, $MES2, $MES3, $MOS, $MOS1, $MOS2, $MOS3) is given in Section 3.

4.3 The text dialogue mode

A switch to the text dialogue mode is realized by using the special substitution $TDIALOGUE. This is equivalent to the substitution $DIALOGUE in the UNIX version of the UFO system. If this is the case, a sequence of questions appear on the screen in the text form. Each question, which is placed in its own frame, consists of the macrovariable description usually followed by the list of its possible values. The name of a macrovariable together with its default value is written on the top of the frame. We have two possibilities for an answer. First, the required value can be entered from the keyboard. Secondly, we can press ENTER to choose the default value. After the assignment of a value to the macrovariable, a new question immediately appears on the screen until the last one is exhausted. The dialogue mode can be terminated by entering the character '!' from the keyboard. We demonstrate four questions as an example:

```
? INPUT ( ) ?
USER SUPPLIED INPUT:

    HERE THE STARTING POINT, BOUNDS FOR VARIABLES,
    TYPES OF CONSTRAINTS, THE STRUCTURE OF SPARSE PROBLEM,
    AND OTHER INPUT DATA HAVE TO BE SPECIFIED.
```

Here a user supplied input is expected. This is a text which should be entered from the keyboard.

```
? MODEL (FF) ?
TYPE OF OBJECTIVE FUNCTION
  FF - GENERAL FUNCTION
  FL - LINEAR FUNCTION
  FQ - QUADRATIC FUNCTION
  AF - SUM OF FUNCTIONS
  AQ - SUM OF SQUARES
  AP - SUM OF POWERS
  AM - MINIMAX
  DF - DIFFERENTIAL SYSTEM WITH GENERAL INTEGRAL CRITERION
  DQ - DIFFERENTIAL SYSTEM WITH INTEGRAL OF SQUARES
  NO - MODEL IS NOT SPECIFIED
```

Here an optimization model, i.e. a type of the objective function, is chosen. We have 10 possibilities, FF, FL, FQ, AF, AQ, AP, AM, DF, DQ, NO. The default value of the macrovariable $MODEL corresponding to the general objective function is FF. By pressing ENTER, the default value FF is accepted.

? NF (0) ?

NUMBER OF VARIABLES

Here the number of variables is expected. This is a positive integer. No default value is offered, i.e. we have to set a value. If this value is not a positive integer, the answer is ignored and the same question appears on the screen.

? FMIN (-1.0D 60) ?

LOWER BOUND FOR FUNCTION VALUE

Here a real constant is expected. By pressing ENTER the default value -1.0D 60 is accepted.

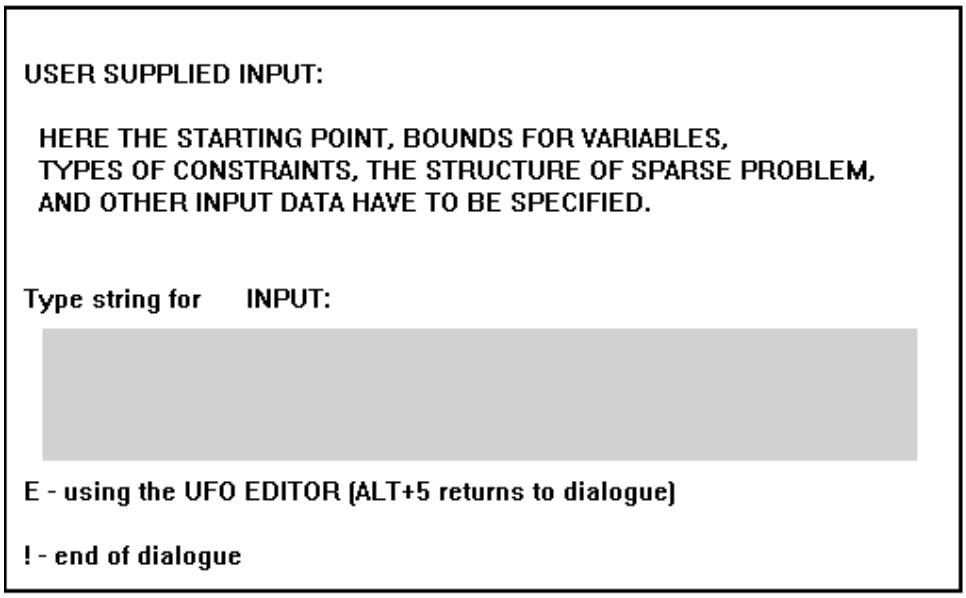
More details concerning a text dialogue mode are given in Appendix A, where a complete text dialogue concerning unconstrained minimization of the Rosenbrock function is shown.

4.4 The graphic dialogue mode

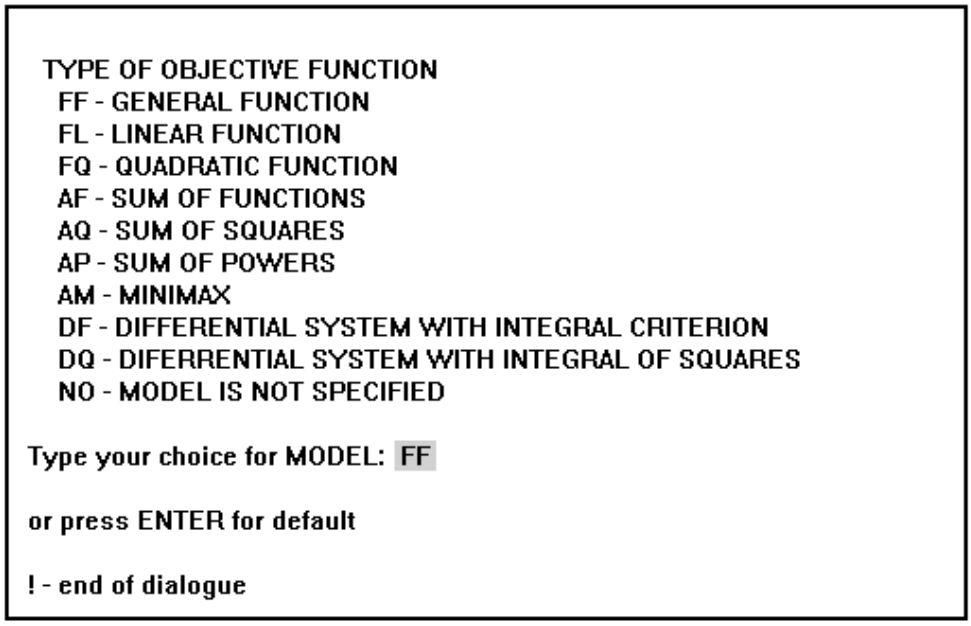
The graphic dialogue mode can be used only on PC computers under the MS DOS operating system using the Fortran Power Station compiler version 1, if $GRAPHICS=1, and under the MS (32 bit) Windows operating system using the Visual Fortran compiler version 4, if $GRAPHICS=4, or version 6, if $GRAPHICS=6 (macrovariable $GRAPHICS is defined in template `UZDCLP.I`, see Section B.7). For any other compiler, this possibility is not implemented, but we can use the text dialogue mode described in Section 4.3 A switch to the graphic dialogue mode is realized by using the special substitution $GDIALOGUE. This is an equivalent to the substitution $DIALOGUE in the PC version of the UFO system. If this is the case, a sequence of screens follows. Each screen realizes one question which is, in fact, the same as that in the text dialogue mode. Nevertheless, the graph dialogue mode has several advantages over the text one:

1. Information is better arranged on the screen.
2. The window for typing answers is in fact a simple editor. Therefore, the text can be easily corrected and a movement controlled by arrows is possible.
3. Application of the special UFO editor is possible for realizing more complicated answers. The UFO editor works with multiple windows so that an answer can be set up from several sources. Therefore a convenient utility of the batch mode can also be used in the dialogue mode.

To compare text and graphic dialogue modes, we again demonstrate the above four questions:

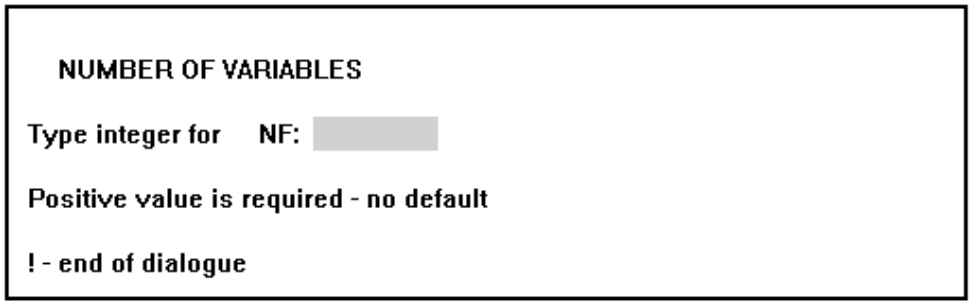


Here a user supplied input is expected. This is a text which should be written into the window displayed on the screen (followed by pressing ENTER). If this text is more complicated, we can use the UFO editor by typing the character 'E' and pressing ENTER. The return from the UFO editor to the graphic dialogue is realized by pressing <alt-5> (Section 1.2). The dialogue mode can be terminated by typing the character '!' and pressing ENTER.

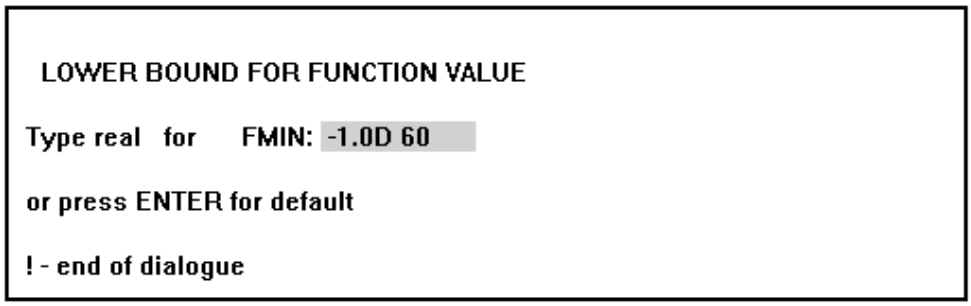


Here an optimization model, i.e. a type of the objective function, is chosen. We have 10 possibilities, FF, FL, FQ, AF, AQ, AP, AM, DF, DQ, NO. The default value of the macrovariable $MODEL, corresponding

to the general objective function, is FF. By pressing ENTER, the default value FF is accepted. If we want to choose a different possibility, it has to be written into the two-character window, followed by pressing ENTER. The dialogue mode can be terminated by typing the character '!' and pressing ENTER.



Here the number of variables is expected. This is a positive integer. No default value is offered, i.e. we have to enter a value. If this value is not a positive integer, the answer is ignored and another answer is expected. The dialogue mode can be terminated by typing the character '!' and pressing ENTER.



Here a real constant is expected. By pressing ENTER the default value -1.0D 60 is accepted. If we want to choose a different value, it has to be written into the twenty-character window, followed by pressing ENTER. The dialogue mode can be terminated by typing the character '!' and pressing ENTER.

5 Output possibilities in the UFO system

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

5.1 Basic screen output

The basic screen output can be used only if $GRAPH='N' and $DISPLAY='N'. In this case, individual rows corresponding to the iterations and the final results are printed on the screen consequently. A print level of the screen output is determined by using the macrovariables $MOUT and $NOUT. The macrovariable $MOUT can have the following values:

$MOUT= 0 -	Screen output is suppressed.
$MOUT=± 1 -	Standard output. The final results appear on the screen.
$MOUT=± 2 -	Extended output. Additional information from every iteration appears on the screen.
$MOUT=± 3 -	Extended output. Additional final results of linear or quadratic programming subproblems appear on the screen.
$MOUT=± 4 -	Extended output. Additional information from every iteration of linear or quadratic programming subproblems appears on the screen.

If $MOUT>0, printed results have the standard form while if $MOUT<0, additional information containing various method specifications and optimization options 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.

The basic screen output can be copied into file `P.OUT` (text file output) if $KOUT=0 and $LOUT>0 (see Section 5.7. In this case, macrovariable $LOUT can have the following values:

$LOUT= 0 -	Basic screen output is not copied into text file `P.OUT`.
$LOUT= 1 -	Basic screen output is copied into text file `P.OUT`. Standard line of the final results is printed.
$LOUT= 2 -	Basic screen output is copied into text file `P.OUT`. Modified line of the final results, containing the termination criterion, is printed.

Typical basic screen outputs are shown in Section 7. If we use values $LOUT=2, $MOUT=-2, $NOUT=2 in the problem specification introduced in Section 7.1, then text file `P.OUT` contains the following information.

```
CLASS = VM - LG1   UPDATE = B   MODEL = FF   HESF = D   NF =       5
    NIT=    0 NFV=    1 NFG=    1  F=  1.866666667     G=0.667D-01
    NIT=    1 NFV=    4 NFG=    4  F=  1.550000000     G=0.150D+00
    NIT=    2 NFV=    7 NFG=    7  F=  1.200000000     G=0.200D+00
    NIT=    3 NFV=    9 NFG=    9  F=  1.000000000     G=0.000D+00
  0 NIT=    3 NFV=    9 NFG=    9  GRAD TOL  F=  1.000000000     G=0.000D+00
FF = -0.1000000000D+01
X  =  0.1000000000D+01  0.2000000000D+01  0.3000000000D+01  0.4000000000D+01
      0.5000000000D+01
TIME= 0:00:00.00
```

```
OPTIONS:

MET   =  8   MET1  =  3  MET2  =  2   MET3  =  1  MET4  =  3   MET5  =  0
MOT   =  2   MOT1  =  0  MOT2  =  0   MOT3  =  0  MOT4  =  0   MOT5  =  0
MES   =  4   MES1  =  2  MES2  =  2   MES3  =  2  MES4  =  0   MES5  =  0
MOS   =  0   MOS1  =  0  MOS2  =  0   MOS3  =  0  MOS4  =  0   MOS5  =  0
MEP   =  0   MEP1  =  0  MEP2  =  0   MEP3  =  0  MEP4  =  0   MEP5  =  0
MED   =  0   MED1  =  0  MED2  =  0   MED3  =  0  MED4  =  0   MED5  =  0
MEG   =  0   MEG1  =  0  MEX   =  0   MEX1  =  0  MEQ   =  0   MEQ1  =  0
MFP   =  0   MFP1  =  0  MLP   =  0   MLP1  =  0  MQP   =  0   MQP1  =  0
KTERS =  3   INITD =  1  INITS =  2   INITH =  1  IREM  =  0   IADD  =  0

MIT   =     500     MIC   =       0   MRED  =      10     MF    =       0
MFV   =    1000     MCG   =       0   IRES1 =     999     MA    =       0
MFG   =   10000     MSTP  =       0   IRES2 =       0     MB    =       0

TOLX = 0.100D-11   EPS0 = 0.100D-05   ETA0 = 0.100D-14   ALF1 = 0.100D-09
TOLF = 0.100D-13   EPS1 = 0.100D-03   ETA1 = 0.100D-12   ALF2 = 0.100D+11
TOLG = 0.100D-05   EPS2 = 0.900D+00   ETA2 = 0.100D-14   ALF3 = 0.000D+00
TOLB =-0.100D+61   EPS3 = 0.000D+00   ETA3 = 0.000D+00   BET1 = 0.000D+00
TOLC = 0.000D+00   EPS4 = 0.000D+00   ETA4 = 0.000D+00   BET2 = 0.000D+00
TOLA = 0.000D+00   EPS5 = 0.000D+00   ETA5 = 0.000D+00   BET3 = 0.000D+00
TOLR = 0.000D+00   EPS6 = 0.000D+00   ETA6 = 0.000D+00   GAM1 = 0.000D+00
XDEL = 0.000D+00   EPS7 = 0.000D+00   ETA7 = 0.000D+00   GAM2 = 0.000D+00
XMAX = 0.100D+04   EPS8 = 0.500D+00   ETA8 = 0.000D+00   GAM3 = 0.000D+00
RPF1 = 0.000D+00   EPS9 = 0.100D-07   ETA9 = 0.100D+61   DEL1 = 0.000D+00
RPF2 = 0.000D+00   FMIN =-0.100D+61   FMAX = 0.100D+21   DEL1 = 0.000D+00
RPF3 = 0.000D+00                                         DEL3 = 0.000D+00
```

The first line contains specifications of the method used ($CLASS='VM', $TYPE='L', $DECOMP='G', $NUMBER=1, $UPDATE='B') and the problem characteristics ($MODEL='FF', $HESF='D', $NF=5). Then, information concerning individual iterations and the final results are printed (0 corresponds to $NEXT=0 and `GRAD TOL` is the cause of termination). Since $MOUT<0, the options (parameters of the method) follows.

5.2 Extended screen output

If we want to use an extended screen output, we have to set $DISPLAY='Y' (the default value is $DISPLAY='N'). This type of screen output consists of text pages which correspond to individual iterations and the final results. The final results are divided into several groups which can be displayed successively. We can change the displayed group by typing particular characters from the keyboard.

Change of the displayed group of the final results:

F - (function) :	Value of the objective function and statistics.
V - (variables) :	Values of variables if NF>0 (with their bounds if KBF>0).
A - (approximation) :	Values of approximating functions if NA>0 (with their prescribed values if KBA>0). Values of selected components of a solution of the set of ordinary differential equations at the prescribed mesh points if NE>0.
C - (constraints) :	Values of constraint functions if NC>0 (with their bounds if KBC>0).
D - (data) :	Data which specify the problem solved (sizes of problem and additional specifications).
O - (options) :	Options which specify the method used.

Exit:

Q - (quit) :	Exit from the extended screen output.

After typing each character we must use ENTER.

Besides these possibilities we can stop every iteration for scanning the iterative process. It is specified if we set $SCAN='Y' (the default value is $SCAN='N'). If $SCAN='N', the output of iterations is suppressed. Scanning of the iterative process can be terminated by using the character '!' from the keyboard.

5.3 Internal FORTRAN graphic environment

The internal graphic screen output can be used only on PC computers under the MS DOS operating system using the Fortran Power Station compiler version 1, if $GRAPHICS=1, and under the MS (32 bit) Windows operating system using the Visual Fortran compiler version 4, if $GRAPHICS=4, or version 6, if $GRAPHICS=6 (macrovariable $GRAPHICS is defined in template `UZDCLP.I`, see Section B.7). For any other compiler, this possibility is not implemented, but we can use the external screen output described in Section 5.4. If we want to use the internal graphic screen output, we have to set $GRAPH='Y' (the default value is $GRAPH='N'). In this case, both iterations and the final results appear in the graphic mode. In general, the internal graphic screen output is a sequence of screens which can be examined successively in a required order. A change of the screen is carried out by using the menu given on the top of this screen. We have three possibilities. First, the character displayed as a capital at the menu item can immediately be typed from the keyboard. Secondly, we can use arrows $\rightarrow$ and $\leftarrow$ (or keys + and - in MS (32 bit) Windows versions) which realize movement in the top menu. The underlined menu item is then selected by pressing ENTER. Finally, we can apply a mouse click to the menu item when DOS graphics is used. In the subsequent description, we focus our attention on the first possibility without a loss of generality.

The graphic form of the final results can be specified in detail by using macrovariables $PATH ('N'- no, 'Y'- yes, 'E'- extended), $MAP ('N'- no, 'Y'- yes, 'E'- extended), $HIL ('N'- no, 'Y'- yes) and $ISO ('N'- no, 'Y'- yes). The final results are divided into several groups which can be displayed successively. We can change the displayed group by typing the particular characters from the keyboard.

Change of the displayed group of the final results:

F - (function) :	Value of the objective function and statistics.
V - (variables) :	Values of variables if NF>0 (with their bounds if KBF>0).
A - (approximation) :	Values of approximating functions if NA>0 (with their prescribed values if KBA>0). Values of selected components of a solution of the set of ordinary differential equations at the prescribed mesh points if NE>0.
C - (constraints) :	Values of constraint functions if NC>0 (with their bounds if KBC>0).
D - (data) :	Data which specify the problem solved (sizes of problem and additional specifications).
O - (options) :	Options which specify the method used.
T - (path) :	Values of the objective function and selected variables (we can change these variables during the graphic output, if we have specified $PATH='E') in the last NPA iterations (only if $PATH='Y' or $PATH='E').

Exit:

Q - (quit) :	Exit from the graphic output.
X - (exit) :	Exit from the UFO system.

Besides these possibilities we can stop every iteration for scanning the iterative process. It is specified if we set $SCAN='Y' (the default value is $SCAN='N'). In every iteration, we can choose one of the possibilities F, V, A, C, D, O as in the case above. If we have chosen either V (variables) or A (approximation) or C (constraints), the intermediate results can be displayed graphically by typing G (graph) from the keyboard. In all these cases, we can execute a single iteration by typing SPACE merely. We can also

execute all iterations until the k-th one by typing J (jump) and entering the number k. Finally, by typing U (automatic), all remaining iterations are executed without scanning.

Besides text representations in the graphic mode, which are essentially like the ones in the extended screen output (with the choice $DISPLAY='Y'), we can chose several types of graphic data representation.

a) Graphic picture:

If we have chosen either V (variables) or A (approximation) or C (constraints), the results can be displayed graphically by typing G (graph) from the keyboard. A graphic picture appears on the screen in this case. It contains either values of variables with indices I, $1 \leq I \leq NF$, 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, the graphic picture contains a component (with the index VAR) of a solution of the set of ordinary differential equations at the mesh points AT(KA), $1 \leq KA \leq NA$. We have to define the index VAR from the keyboard in this case. The graphic picture can be changed by typing the particular characters from the keyboard.

Change of representation:

V - (values) :	Values are drawn.
O - (ordinates) :	Values and ordinates from zero axis are drawn.
C - (curve) :	Values are connected by a curve.
M - (mixed) :	Curve and ordinates are drawn.

Change of graph (if either KBF>0 or KBA>0 or KBC>0):

F - (functions) :	Either values of variables X(I), $1 \leq I \leq NF$, or values of the approximating functions AF(KA), $1 \leq KA \leq NA$, or values of the constraint functions CF(KC), $1 \leq KC \leq NC$, are demonstrated.
A - (approximation) :	Either values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, or values of the approximating functions AF(KA) together with their prescribed values AM(KA) , $1 \leq KA \leq NA$, or values of the constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq KC \leq NC$, are demonstrated.
D - (differences) :	Either the differences between variables and their bounds or the differences between the approximating functions and their prescribed values or the differences between the constraint functions and their bounds are demonstrated.

Continuation (if either NF > 200 or NA > 200 or NC > 200):

P - (previous) :	Previous set of at most 200 values is drawn.
N - (next) :	Next set of at most 200 values is drawn.

Choice of the next displayed iteration (only if $SCAN='Y'):

J - (jump) :	The iterative process is stopped at the k-th iteration. Number k is read from the keyboard.
U - (automatic) :	All remaining iterations are executed without scanning.

New graph or return:

W - (new) :	This possibility can be used only if NE>0. Then a new component (with a new index VAR) of a solution of the set of ordinary differential equations is drawn. We have to define a new index VAR from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

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

b) Two-dimensional orbit:

If NE>1, we can draw an orbit of two components of a solution of the set of ordinary differential equations by typing G (graph) from the keyboard. We have to define an index VAR for every selected component of a solution (according to the text appeared on the screen). The two-dimensional orbit can be changed by typing the particular characters from the keyboard.

Change of the orbit:

V - (values) :	Values are drawn.
C - (curves) :	Values are connected by a curve.

New orbit or return:

W - (new) :	New components of a solution of the set of ordinary differential equations are drawn. We have to define new two indices from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

c) Three-dimensional orbit:

If NE>2, then we can draw an orbit of three components of a solution of the set of ordinary differential equations by typing I (picture) from the keyboard. We have to define an index VAR for every selected component of a solution (according to the text appeared on the screen). The three-dimensional orbit can be changed by typing the particular characters from the keyboard.

Change of the orbit:

V - (values) :	Values are drawn.
C - (curves) :	Values are connected by a curve.
O - (rotate) :	Rotation of values or curves about a vertical axis by a subsequently entered angle Dfi.
T - (tilt) :	Tilting rotated values or curves by a subsequently entered angle Dtheta.
A - (axes) :	Drawing a picture with rotated and tilted axes.
S - (scale) :	Scaling of rotated and tilted values or curves to make full use of the screen.

New orbit or return:

W - (new) :	New components of a solution of the set of ordinary differential equations are drawn. We have to define new three indices from the keyboard in this case.
Q - (quit) :	Return to the displayed group of final results.

d) Colored map of the objective function:

If we have specified either $MAP='Y' or $MAP='E' (default value is $MAP='N'), we can draw a colored map of the objective function by typing M (map) from the keyboard. This picture can be changed by typing the 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 - (inverse) :	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 (only if $HIL='Y' is specified).
S - (isolines) :	Drawing contours of the objective function (only if $ISO='Y' is specified).

W - (new) : Selection of new variables and drawing a new colored map.
Q - (quit) : Return to the displayed group of final results.

If we set $MAP='Y', one picture for two variables is drawn. If we set $MAP='E', three pictures for all combinations of two from three variables are drawn. In both cases we have to define, from the keyboard, an index VAR and bounds XL(VAR), XU(VAR) for every variable used (according to the text appeared on the screen). Note that the choice $MAP='E' excludes the choices $HIL='Y' and $ISO='Y' so that the other pictures cannot be used.

e) Objective function surface:

If we have specified $HIL='Y' (default value is $HIL='N'), we can draw an objective function surface with respect to visibility by typing H (hills) from the keyboard. This picture can be changed by typing particular characters from the keyboard.

Change of the surface:

L - (linear) : Linear scale of the surface.
G - (logarithmic) : Logarithmic scale of the surface.
R - (refinement) : Refinement of the surface.
B - (back) : Back refinement of the surface.
O - (rotate) : Rotation of the surface about a vertical axis by a subsequently entered angle Dfi.
T - (tilt) : Tilting the rotated surface by a subsequently entered angle Dtheta.
F - (face) : Facing the rotated surface (drawing the rotated surface without tilting).
N - (inverse) : Surface of the objective function negation.

Another type of picture, new surface or return:

M - (map) : Drawing a colored map of the objective function (only if $MAP='Y' is specified).
S - (isolines) : Drawing contours of the objective function (only if $ISO='Y' is specified).
W - (new) : Selection of new variables and drawing new surface.
Q - (quit) : Return to the displayed group of final results.

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

f) Objective function contours:

If we have specified $ISO='Y' (default value is $ISO='N'), we can draw an objective function contours by typing S (isolines) from the keyboard. This picture can be changed by typing particular characters from the keyboard.

Change of contours:

L - (linear) : Linear scale of contours.
G - (logarithmic) : Logarithmic scale of contours.
R - (refinement) : Refinement of contours.
B - (back) : Back refinement of contours.
O - (color) : Coloring of contours and used levels.
N - (inverse) : Inverse coloring of contours and used levels.

Another type of picture, new contours or return:

M - (map) : Drawing a colored map of the objective function (only if $MAP='Y' is specified).
H - (hills) : Drawing an objective function surface with respect to visibility (only if $HIL='Y' is specified).
W - (new) : Selection of new variables and drawing a new surface.
Q - (quit) : Return to the displayed group of final results.

g) Graphic path of the objective function and selected variables:

If we have chosen T (path), we can display the values of the objective function as a function graph by typing G (graph) or draw the objective function contours with the path in the last NPA iterations by typing S (isolines). The graph can be changed in the same way as in a).

Change of contours:

L - (linear) :	Linear scale of contours.
G - (logarithmic) :	Logarithmic scale of contours.
R - (refinement) :	Refinement of contours.
B - (back) :	Back refinement of contours.
Z - (zoom) :	Zoom of the path for the number of last iterations entered.

Another type of picture, new contours or return:

W - (new) :	Selection of new variables and drawing new contours (only if we have specified $PATH='E').
Q - (quit) :	Return to the displayed group of final results.

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

5.4 Data for external graphic utilities

The external graphic screen output can be used in both the MS (32 bit) Windows and the Linux versions of the UFO system if $GRAPHICS = –1 (macrovariable $GRAPHICS is defined in template `UZDCLP.I`, see Section B.7). If we want to use the external graphic output, we have to set $GRAPH='Y' (the default value is $GRAPH='N'). In this case, the UFO system generates five output files `P.PAR`, `P.GRF`, `P.DIF`, `P.SUR`, `P.PTH` containing data for the external graphic packages (e.g. for the package JUFO [161]).

File `P.PAR` contains scalar input and output data, which correspond to displayed groups F - (function), D - (data), O - (options) of the final results used in the internal screen output. The following format is used for creating file `P.PAR`:

```
C
C     function
C
      WRITE(13,'(16I5)') NIT,NFV,NFG,NFH,NAV,NAG,NAH,NCV,NCG,NCH,NDECF,
     & NCGR,NRES,NREM,NADD,NEXT
      WRITE(13,'(4D20.12)') F,DMAX,GMAX
C
C     data
C
      WRITE(13,'(16I5)') NF,NA,NAL,MAL,NC,NCL,MCL,KDF,KBF,KSF,KCF,ISNF,
     & NORMF,KDA,KBA,KSA,KCA,ISNA,NORMA,KDC,KBC,KSC,KCC,ISNC,NORMC
C
C     options
C
      WRITE(13,'(16I5)') MET,MET1,MET2,MET3,MET4,MET5,MES,MES1,MES2,
     & MES3,MES4,MES5,MOT,MOT1,MOT2,MOT3,MOT4,MOT5,MOS,MOS1,MOS2,MOS3,
     & MOS4,MOS5,MEP,MEP1,MEP2,MEP3,MEP4,MEP5,MED,MED1,MED2,MED3,MED4,
     & MED5,MEX,MEX1,MEX2,MEX3,MEX4,MEX5,MEG,MEG1,MEQ,MEQ1,MFP,MFP1,
     & MLP,MLP1,MLP2,MQP,MQP1,MQP2,KTERS,INITD,INITS,INITH,IREM,IADD,
     & MRED,IRES1,IRES2,MIC,MIT,MFV,MFG,MCGR,MSTP
      WRITE(13,'(4D20.14)') XDEL,XMAX,FMIN,FMAX,RPF1,RPF2,RPF3,RPF4,
     & RPF5,EPS0,EPS1,EPS2,EPS3,EPS4,EPS5,EPS6,EPS7,EPS8,EPS9,ETA0,
```

```
     & ETA1,ETA2,ETA3,ETA4,ETA5,ETA6,ETA7,ETA8,ETA9,ALF1,ALF2,ALF3,
     & BET1,BET2,BET3,GAM1,GAM2,GAM3,DEL1,DEL2,DEL3,TOLX,TOLF,TOLG,
     & TOLB,TOLC,TOLA,TOLR
```

File `P.GRF` contains values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, values of the approximating functions AF(KA) together with their prescribed values AM(KA), $1 \leq KA \leq NA$, and values of the constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq KC \leq NC$. This corresponds to displayed groups V - (variables), A - (approximation), C - (constraints) of the final results used in the internal screen output. The following format is used for creating file `P.GRF`:

```
C
C     variables
C
      WRITE(11,'(2I5)') NF,KBF
      IF (NF.GT.0) THEN
      WRITE(11,'(4D20.12)')(X(I),I=1,NF)
      IF (KBF.GT.0) THEN
      WRITE(11,'(16I5)') (IX(I),I=1,NF)
      WRITE(11,'(4D20.12)')(XL(I),I=1,NF)
      WRITE(11,'(4D20.12)')(XU(I),I=1,NF)
      ENDIF
      ENDIF
C
C     approximation
C
      WRITE(11,'(2I5)') NA,KBA
      IF (NA.GT.0) THEN
      WRITE(11,'(4D20.12)')(AF(KA),KA=1,NA)
      IF (KBA.GT.0) THEN
      WRITE(11,'(4D20.12)')(AM(KA),KA=1,NA)
      ENDIF
      ENDIF
C
C     constraints
C
      WRITE(11,'(2I5)') NC,KBC
      IF (NC.GT.0) THEN
      WRITE(11,'(4D20.12)')(CF(KC),KC=1,NC)
      IF (KBC.GT.0) THEN
      WRITE(11,'(16I5)') (IC(KC),KC=1,NC)
      WRITE(11,'(4D20.12)')(CL(KC),KC=1,NC)
      WRITE(11,'(4D20.12)')(CU(KC),KC=1,NC)
      ENDIF
      ENDIF
```

More details about these quantities are given in Section 1.8.

File `P.DIF` contains data for drawing NED = min(NE, 3) curves and orbits corresponding to selected functions which are solutions of a system of NE ordinary differential equations. If NE > 3, we have to specify indices of the selected functions from the keyboard. The following format is used for creating file `P.DIF`:

```
C
C     differential equations
C
      WRITE(14,'(2I5)') NED,NA
      IF (NA.GE.1.AND.NED.GE.1) THEN
      WRITE(14,'(4D20.12)') (AT(KA),KA=1,NA)
      WRITE(14,'(2I5)') 1,IG
      WRITE(14,'(4D20.12)') (AY(IG,KA),KA=1,NA)
      IF (NED.GE.2) THEN
      WRITE(14,'(2I5)') 2,JG
      WRITE(14,'(4D20.12)') (AY(JG,KA),KA=1,NA)
      ENDIF
      IF (NED.GE.3) THEN
      WRITE(14,'(2I5)') 3,KG
      WRITE(14,'(4D20.12)') (AY(KG,KA),KA=1,NA)
      ENDIF
      ENDIF
```

Here NED = min(NE, 3) is the number of drawing curves or orbits, NA is the number of nodes (values of the independent variable), AT(KA) are values of the independent variable, IG is the index of the first drawn function, AY(IG,KA) are values of the first drawn function, JG is the index of the second drawn function, AY(JG,KA) are values of the second drawn function, KG is the index of the third drawn function and AY(KG,KA) are values of the third drawn function.

The generation of the remaining output files depends on values of macrovariables $MAP ('N'- no, 'Y'- yes, 'E'- extended) and $PATH ('N'- no, 'Y'- yes, 'E'- extended). File `P.SUR`, generated if $MAP<>'N', contains data for drawing one surface (if $MAP = 'Y') or three surfaces (if $MAP = 'E') corresponding to selected pair or pairs of variables. If $MAP<>'N', we have to specify indices of the selected variables and their lower and upper bounds from the keyboard. The following format is used for creating file `P.SUR`:

```
C
C     surface
C
      WRITE(12,'(2I5)') NCD,NGR
      IF (NGR.GE.1.AND.NCD.EQ.1) THEN
      WRITE(12,'(I5,I5,3D20.12)') 1,NGR,F,VALMIN,VALMAX
      WRITE(12,'(I5,I5,3D20.12)') IRE,IMIN,XIGR,XLIGR,XUIGR
      WRITE(12,'(I5,I5,3D20.12)') JRE,JMIN,XJGR,XLJGR,XUJGR
      WRITE(12,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      ENDIF
      IF (NGR.GE.1.AND.NCD.EQ.3) THEN
      WRITE(12,'(I5,I5,3D20.12)') 2,NGR,F,VALMIN,VALMAX
      WRITE(12,'(I5,I5,3D20.12)') IRE,IMIN,XIGR,XLIGR,XUIGR
      WRITE(12,'(I5,I5,3D20.12)') KRE,JMIN,XJGR,XLJGR,XUJGR
      WRITE(12,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      WRITE(12,'(I5,I5,3D20.12)') 3,NGR,F,VALMIN,VALMAX
      WRITE(12,'(I5,I5,3D20.12)') JRE,IMIN,XIGR,XLIGR,XUIGR
      WRITE(12,'(I5,I5,3D20.12)') KRE,JMIN,XJGR,XLJGR,XUJGR
      WRITE(12,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      ENDIF
```

Here NCD is the number of drawn surfaces (1 if $MAP = 'Y' or 3 if $MAP = 'E'), NGR is the number of nodes corresponding to every selected variable (NGR*NGR is the number of drawn function values), IRE is the index of the first selected variable, JRE is the index of the second selected variable, KRE is the index of

the third selected variable, F is the final function value (usually an approximation of the minimum value), VALMIN is the minimum function value of the selected surface, VALMAX is the maximum function value of the selected surface, IMIN is the first coordinate of the minimum in the selected surface, XLIGR is the minimum value of the first variable of the selected surface, XUIGR is the maximum value of the first variable of the selected surface, JMIN is the second coordinate of the minimum in the selected surface, XLJGR is the minimum value of the second variable of the selected surface, XUJGR is the maximum value of the second variable of the selected surface, and FL contains NGR*NGR drawn function values.

File `P.PTH`, generated if $PATH<>'N', contains data for drawing one path (if $PATH = 'Y') or three paths (if $PATH = 'E') corresponding to selected pair or pairs of variables. If $PATH<>'N', we have to specify indices of the selected variables from the keyboard. The following format is used for creating file `P.PTH`:

```
C
C     path
C
      WRITE(15,'(2I5)') NPD,NPA
      IF (NPA.GE.1.AND.NPD.GE.1) THEN
      WRITE(15,'(4D20.12)')(FPA(I),I=1,NPA)
      WRITE(15,'(2I5)') IGR,NPA
      WRITE(15,'(4D20.12)')(XIPA(I),I=1,NPA)
      ENDIF
      IF (NPA.GE.1.AND.NPD.GE.2) THEN
      WRITE(15,'(2I5)') JGR,NPA
      WRITE(15,'(4D20.12)')(XJPA(I),I=1,NPA)
      ENDIF
      IF (NPA.GE.1.AND.NPD.GE.3) THEN
      WRITE(15,'(2I5)') KGR,NPA
      WRITE(15,'(4D20.12)')(XKPA(I),I=1,NPA)
      ENDIF
      WRITE(15,'(2I5)') NCD,NGR
      IF (NGR.GE.1.AND.NCD.EQ.1) THEN
      WRITE(15,'(I5,I5,3D20.12)') 1,NGR,F,VALMIN,VALMAX
      WRITE(15,'(I5,I5,3D20.12)') IPA,IMIN,XIGR,XLIGR,XUIGR
      WRITE(15,'(I5,I5,3D20.12)') JPA,JMIN,XJGR,XLJGR,XUJGR
      WRITE(15,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      ENDIF
      IF (NGR.GE.1.AND.NCD.EQ.3) THEN
      WRITE(15,'(I5,I5,3D20.12)') 2,NGR,F,VALMIN,VALMAX
      WRITE(15,'(I5,I5,3D20.12)') IPA,IMIN,XIGR,XLIGR,XUIGR
      WRITE(15,'(I5,I5,3D20.12)') KPA,JMIN,XJGR,XLJGR,XUJGR
      WRITE(15,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      WRITE(15,'(I5,I5,3D20.12)') 3,NGR,F,VALMIN,VALMAX
      WRITE(15,'(I5,I5,3D20.12)') JPA,IMIN,XIGR,XLIGR,XUIGR
      WRITE(15,'(I5,I5,3D20.12)') KPA,JMIN,XJGR,XLJGR,XUJGR
      WRITE(15,'(4D20.12)')((FL(IVAR,JVAR),IVAR=1,NGR),JVAR=1,NGR)
      ENDIF
```

Here NPD is the number of drawn paths (1 if $PATH = 'Y' or 3 if $PATH = 'E'), NPA is the number of drawn points of every path (usually the number of iterations), FPA are function values in drawn points, IPA is the index of the first selected variable, XIPA are values of the first selected variable in drawn points, JPA is the index of the second selected variable, XJPA are values of the second selected variable in drawn points, KPA is the index of the third selected variable, XKPA are values of the third selected variable in drawn points. The other quantities have the same meaning as in the file `P.SUR`.

5.5 Interface to the MATLAB graphic environment

The MATLAB graphic output can be used in both the MS (32 bit) Windows and the Linux versions of the UFO system if $GRAPHICS = –2 (macrovariable $GRAPHICS is defined in template `UZDCLP.I`, see Section B.7). If we want to use the MATLAB graphic output, we have two possibilities.

The first possibility is the use of internal graphic ($GRAPH='I' or, equivalently, $GRAPH='Y'). In this case, the MATLAB environment is used interactively in a similar way as the internal Fortran environment, but the graphic screen and control statements are different. The answers on the queries concerning particular type of the graph are summarized. After confirming generating the graph, the Matlab Command Window will appear together with the corresponding generated figure. This window can be closed by pressing the Enter key.

The second possibility is the use of external graphic ($GRAPH='E'). In this case, the UFO system generates an output file `P.m` serving as an input file for the MATLAB graphic environment. Based on the screen interface, the user is asked to specify several parameters in form of answers on the system queries. The parameters are then summarized and after confirming generating the graph, the corresponding MATLAB code is generated.

Note that the default value is $GRAPH='N'.

If INTERNAL GRAPHIC is selected, i.e. $GRAPH='I' or $GRAPH='Y', a list of all possible queries appearing on the screen follows. Depending on the values of variables NF, NA, NC, NGR, NE, the following main menu appears on the screen (see below the main menu for differential equations):

```
V = VARIABLES
A = APPROXIMATIONS
C = CONSTRAINTS
S = ISOLINES/MAP/SURFACE
P = PATH WITH ISOLINES/MAP
X = PATH OF VARIABLES
F = PATH OF FUNCTION VALUES
E = EXIT
```

The choices P, X, F concern drawing paths if $PATH<>'N'. If $PATH='Y', then, at first, we have to specify indexes of visualized variables using the queries

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
```

If V is specified in the main menu, the following queries concerning the values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, appear on the screen:

```
GRAPH OF VARIABLES:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
  NF = [number_of_variables_NF]
  START INDEX (1..[NF]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NF]):  [ENTER = [NF]]
VARIABLES: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

When KBF=0, the statements corresponding the box constraints are omitted.

If A is specified in the main menu, the following queries concerning the values of approximating functions AF(KA) together with their prescribed values AM(KA), $1 \leq KA \leq NA$, appear on the screen:

```
GRAPH OF APPROXIMATIONS:
  F = AF, M = AF,AM, D = AF-AM, R = RETURN  [ENTER = R]
```

```
TYPE OF GRAPH FOR AF:
  P = POINTS, C = CURVE, L = LINES  [ENTER = L]
  NA = [number_of_approximating_functions_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
APPROXIMATIONS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

When KBA=0, the statements corresponding the array AM are omitted.

If C is specified in the main menu, the following queries concerning the values of constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq \mathrm{KC} \leq \mathrm{NC}$, appear on the screen:

```
GRAPH OF CONSTRAINTS:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
  NC = [number_of_constraints_NC]
  START INDEX (1..[NC]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NC]):  [ENTER = [NC]]
CONSTRAINTS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If S is specified in the main menu, the following queries concerning the reliefs appear on the screen:

```
INDEX OF THE 1-ST VARIABLE:
X(IGR) = the_value_of_X(IGR)
XL(IGR) =
XU(IGR) =
INDEX OF THE 2-ND VARIABLE:
X(JGR) = the_value_of_X(JGR)
XL(JGR) =
XU(JGR) =
GRAPH OF:  I = ISOLINES, M = MAP, S = SURFACE,
           N = NEW INDEXES, R = RETURN  [ENTER = R]
```

Here IGR, JGR are indexes of the visualized variables and XL(IGR), XU(IGR), XL(JGR), XU(JGR) are bounds which specify the domain of the drawn relief. If N is specified, new indexes IGR, JGR can be selected. If I or M or S is specified, the following queries appear:

```
  NUMBER OF CONTOUR LEVELS:  [ENTER = 64]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
GRAPH OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

Note that in case of S, the query concerning the number of contour levels is omitted.

If P is specified in the main menu, the queries concerning the path with isolines or map appearing on the screen depend on macrovariable $PATH:

```
INDEX OF THE 1-ST VARIABLE:
INDEX OF THE 2-ND VARIABLE:
PATH WITH: I = ISOLINES, M = MAP, N = NEW INDEXES, R = RETURN  [ENTER = R]
```

If $PATH='E', the user must specify the indexes of variables while if $PATH='Y', the queries concerning indexes are omitted because they were specified before. If I or M is specified, the following queries appear:

```
  TYPE OF PATH: P = POINTS, C = CURVE, B = BOTH  [ENTER = B]
  NUMBER OF CONTOUR LEVELS:  [ENTER = 64]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
  NUMBER OF POINTS: (MAX max_number_of_points)  [ENTER = max_number_of_points]
PATH WITH [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If X is specified in the main menu, the following queries concerning the path of iterations appear on the screen:

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
INDEX OF THE 3-RD VISUALIZED VARIABLE: KGR =
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF X(IGR)
  2 = ITERATIONS OF X(JGR)
  3 = ITERATIONS OF X(KGR)
  4 = ITERATIONS OF X(IGR), X(JGR)
  5 = ITERATIONS OF X(IGR), X(KGR)
  6 = ITERATIONS OF X(JGR), X(KGR)
  7 = ITERATIONS OF X(IGR), X(JGR), X(KGR)
  0 = RETURN  [ENTER = 0]
```

Here IGR, JGR, KGR are indexes of the visualized variables. If $PATH='Y', the queries concerning indexes are omitted because they were specified before and only options concerning indexes IGR,JGR appear. If 1-7 is specified, the following queries appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
ITERATIONS OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If F is specified in the main menu, the following queries concerning the path of function values appear on the screen:

```
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF F, 0 = RETURN  [ENTER = 0]
```

If 1 is specified, the following queries appear:

```
  TYPE OF GRAPH: 1 = POINTS, 2 = CURVE  [ENTER = 2]
VALUES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If E is specified in the main menu, the program is terminated. Note that if RETURN (R or 0) is specified in particular submenus, then a return to the main menu is performed.

In case of differential equations, the main menu depends on the value of variable NE. One can draw trajectories, corresponding orbits (if NE $>$ 1) and the picture (if NE $>$ 2). If NE $>$ 3, the user must also specify indexes IE, JE, KE of the selected solution functions. The following main menu appears on the screen:

```
INDEX OF THE 1-ST VISUALIZED DIFFERENTIAL EQUATION: IE =
INDEX OF THE 2-ND VISUALIZED DIFFERENTIAL EQUATION: JE =
INDEX OF THE 3-RD VISUALIZED DIFFERENTIAL EQUATION: KE =
T = 1D TRAJECTORY
```

```
O = 2D ORBIT
P = 3D PICTURE
N = NEW INDEXES
E = EXIT
```

Note that depending on the value of NE, only the corresponding possible options appear. If N is specified, new indexes can be chosen.

If T is specified in the main menu, the following queries concerning 1D trajectories appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = TRAJECTORY OF YA(IE)
  2 = TRAJECTORY OF YA(JE)
  3 = TRAJECTORY OF YA(KE)
  4 = TRAJECTORIES OF YA(IE), YA(JE)
  5 = TRAJECTORIES OF YA(IE), YA(KE)
  6 = TRAJECTORIES OF YA(JE), YA(KE)
  7 = TRAJECTORIES OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE < 3, only trajectories of YA with a corresponding number of indexes are considered. If 1-7 is specified, the following queries appear:

```
  NA = [number_of_mesh_points_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
TRAJECTORIES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If O is specified in the main menu, the following queries concerning 2D orbits appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = ORBIT OF YA(IE), YA(JE)
  2 = ORBIT OF YA(IE), YA(KE)
  3 = ORBIT OF YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE $= 2$, only the first option is possible. If 1-3 is specified, the following query appears:

```
ORBIT OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If P is specified in the main menu, the following queries concerning 3D picture appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = PICTURE OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

If 1 is specified, the following query appears:

```
PICTURE OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If E is specified in the main menu, the program is terminated. Note that if RETURN (0) is specified in particular submenus, then a return to the main menu is performed.

If EXTERNAL GRAPHIC is selected, i.e. $GRAPH='E', a list of all possible queries appearing on the screen follows.

If $PATH='Y' (two variables IGR,JGR) or $PATH='E' (three variables IGR,JGR,KGR), then at first the user is asked to specify the visualized variables for the paths:

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
INDEX OF THE 3-RD VISUALIZED VARIABLE: KGR =
```

Now, depending on the values of variables NF, NA, NC, NGR, NE, the user is asked to specified the types of graphs and their parameters that will be generated into the file `P.m`. Note that the choice RETURN (R or 0, see below) from the main menu will cause a switch to the main menu of the next graph and from the submenu will cause a return to the main menu of the current graph.

If NF > 0, the query

```
GRAPH OF VARIABLES:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
```

concerning the values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, appears. If the answer is P or C, the following queries concerning the parameters of the graph will appear:

```
  NF = [number_of_variables_NF]
  START INDEX (1..[NF]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NF]):  [ENTER = [NF]]
VARIABLES: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[P = POINTS]`):

```
figure;
il=[ indices_of_lower_bounds_for_variables ];
xl=[ values_of_lower_bounds_for_variables ];
iu=[ indices_of_upper_bounds_for_variables ];
xu=[ values_of_upper_bounds_for_variables ];
ix=[ indices_of_variables ];
xi=[ values_of_variables ];
h = plot(il,xl,iu,xu,ix,xi);
set(h,{'LineStyle'},{'none'});
set(h,{'Marker'},{'^';'v';'o'});
set(h,{'MarkerSize'},{6;6;6});
set(h,{'MarkerEdgeColor'},{'b';'b';'r'});
set(h,{'MarkerFaceColor'},{'b';'b';'r'});
for i=1:length(il)
ll=line([il(i) il(i)],[xl(i) xi(il(i))]);
set(ll,{'LineWidth'},{1});
set(ll,{'Color'},{'b'});
end
for i=1:length(iu)
lu=line([iu(i) iu(i)],[xu(i) xi(iu(i))]);
```

```
set(lu,{'LineWidth'},{1});
set(lu,{'Color'},{'b'});
end
axis([0 length(ix)+1 minimum_in_values maximum_in_values]);
tit=title('Variables');
xl=xlabel('Indices of variables');
yl=ylabel('Bounds and values of variables');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMVAR01;
```

When KBF=0, the statements corresponding the box constraints are omitted.

If NA $>$ 0, the query

```
GRAPH OF APPROXIMATIONS:
  F = AF, M = AF,AM, D = AF-AM, R = RETURN  [ENTER = R]
```

concerning the values of approximating functions AF(KA) together with their prescribed values AM(KA), $1 \leq$ KA $\leq$ NA, appears. If the answer for the graph is F or M or D, the following queries concerning the parameters of the graph will appear:

```
TYPE OF GRAPH FOR AF:
  P = POINTS, C = CURVE, L = LINES  [ENTER = L]
  NA = [number_of_approximating_functions_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
APPROXIMATIONS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in cases of `[D = AF-AM]` and `[L = LINES]`):

```
figure;
ii=1:number_of_drawn_values;
dif=[ values_of_differences_between_AF(KA)_and_AM(KA) ];
h=plot(ii,dif);
set(h,{'LineStyle'},{'none'});
for i=1:length(ii)
hl=line([ii(i) ii(i)],[0 dif(i)]);
set(hl,{'Color'},{'r'});
end
axis([0 length(ii)+1 minimum_in_values maximum_in_values]);
tit=title('Approximations');
xl=xlabel('Indices of approximations');
yl=ylabel('Differences of approximations');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMAPP09;
```

When KBA=0, the statements corresponding the array AM are omitted.

If NC > 0, the query

```
GRAPH OF CONSTRAINTS:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
```

concerning the values of constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq$ KC $\leq$ NC, appears. If the answer is P or C, the following queries concerning the parameters of the graph will appear:

```
  NC = [number_of_constraints_NC]
  START INDEX (1..[NC]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NC]):  [ENTER = [NC]]
CONSTRAINTS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[P = POINTS]`):

```
figure;
il=[ indices_of_lower_bounds_for_constraints ];
cl=[ values_of_lower_bounds_for_constraints ];
iu=[ indices_of_upper_bounds_for_constraints ];
cu=[ values_of_upper_bounds_for_constraints ];
ic=[ indices_of_constraints ];
ci=[ values_of_constraints ];
h = plot(il,cl,iu,cu,ic,ci);
set(h,{'LineStyle'},{'none'});
set(h,{'Marker'},{'^';'v';'o'});
set(h,{'MarkerSize'},{6;6;6});
set(h,{'MarkerEdgeColor'},{'b';'b';'r'});
set(h,{'MarkerFaceColor'},{'b';'b';'r'});
for i=1:length(il)
ll=line([il(i) il(i)],[cl(i) ci(il(i))]);
set(ll,{'LineWidth'},{1});
set(ll,{'Color'},{'b'});
end
for i=1:length(iu)
lu=line([iu(i) iu(i)],[cu(i) ci(iu(i))]);
set(lu,{'LineWidth'},{1});
set(lu,{'Color'},{'b'});
end
axis([0 length(ic)+1 minimum_in_values maximum_in_values]);
tit=title('Constraints');
xl=xlabel('Indices of constraints');
yl=ylabel('Bounds and values of constraints');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMCON01;
```

Generation of reliefs depends on the value of macrovariable $MAP ('N'- no, 'Y'- yes, 'E'- extended). If $MAP='Y', then one relief (isolines, color map or surface) is generated, while if $MAP='E', then three reliefs are generated as is specified by the following queries:

```
GRAPH OF MAPS:
  S = ISOLINES/MAP/SURFACE, R = RETURN  [ENTER = R]
```

If S is specified, then these queries follow:

```
INDEX OF THE 1-ST VARIABLE:
X(IGR) = the_value_of_X(IGR)
XL(IGR) =
XU(IGR) =
INDEX OF THE 2-ND VARIABLE:
X(JGR) = the_value_of_X(JGR)
XL(JGR) =
XU(JGR) =
INDEX OF THE 3-RD VARIABLE:
X(KGR) = the_value_of_X(KGR)
XL(KGR) =
XU(KGR) =
GRAPH OF:  1 = ISOLINES, 2 = MAP, 3 = SURFACE, 9 = NEW INDEXES, 0 = RETURN
  [ 1 2 3 12 13 23 123 9, ENTER = 0 ]
```

Here IGR, JGR, KGR are indexes of the visualized variables and XL(IGR), XU(IGR), XL(JGR), XU(JGR), XL(KGR), XU(KGR) are bounds which specify the domain of the drawn relief. If 9 is specified, new indexes IGR, JGR, KGR can be chosen. If 1 or 2 or 3 or 12 or 13 or 23 or 123 is specified, the following queries will appear:

```
  NUMBER OF CONTOUR LEVELS:  [ENTER = 64]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
GRAPH OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated. If the answer for the graph contains 1 (isolines), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
figure;
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=64;
contour(x,y,f,level);
mn=min(f(:));
mx=max(f(:));
```

```
set(gca, 'CLim', [mn, mx]);
colorbar;
hold on;
pi=plot(xm,ym);
set(pi,{'Marker'},{'s'});
set(pi,{'MarkerSize'},{10});
set(pi,{'MarkerEdgeColor'},{'k'});
set(pi,{'MarkerFaceColor'},{'k'});
axis([xligr xuigr xljgr xujgr]);
tit=title('Isolines');
xl=xlabel('X(1)');
yl=ylabel('X(2)');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMISO01;
```

If the answer for the graph contains 2 (color map), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
figure;
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=64;
contourf(x,y,f,level);
mn=min(f(:));
mx=max(f(:));
set(gca, 'CLim', [mn, mx]);
colorbar;
hold on;
pm=plot(xm,ym);
set(pm,{'Marker'},{'s'});
set(pm,{'MarkerSize'},{10});
set(pm,{'MarkerEdgeColor'},{'w'});
set(pm,{'MarkerFaceColor'},{'w'});
axis([xligr xuigr xljgr xujgr]);
tit=title('Map');
xl=xlabel('X(1)');
yl=ylabel('X(2)');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
```

```
set(yl,'FontSize',15);
print -depsc PMMAP01;
```

If the answer for the graph contains 3 (surface), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
figure;
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
surfc(x,y,f);
mn=min(f(:));
mx=max(f(:));
set(gca, 'CLim', [mn, mx]);
colorbar;
xlim([xligr xuigr]);
ylim([xljgr xujgr]);
zlim('auto');
tit=title('Surface');
xl=xlabel('X(1)');
yl=ylabel('X(2)');
zl=zlabel('F');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
set(zl,'FontSize',15,'Rotation',0);
pos = get(zl,'Position');
set(zl,'Position', pos + [-0.1, 0, 0]);
print -depsc PMSUR01;
```

Here NGR is the number of nodes corresponding to every selected variable (NGR*NGR is the number of drawn function values), XLIGR is the lower bound of the first variable of the selected surface specified by the user, XUIGR is the upper bound of the first variable of the selected surface specified by the user, XLJGR is the lower bound of the second variable of the selected surface specified by the user, XUJGR is the upper bound of the second variable of the selected surface specified by the user, F contains NGR*NGR drawn function values, XM is the value of the solution point whose index is the same as the index of the first variable and YM is the value of the solution point whose index is the same as the index of the second variable. The solution point [XM,YM] is drawn only if it is contained in the box defined by the values XLIGR, XUIGR, XLJGR, XUJGR and is marked as a black square in case of isolines and as a white square in case of the color map. Note that if $MAP='E', then isolines, color map, and/or surface (if requested) with indices IGR, KGR and JGR, KGR are concurrently generated.

The last part of the code concerns drawing paths if $PATH<>'N'. We can draw iterations of up to three variables X(IGR), X(JGR), X(KGR) and the function value F (depending on the variable $PATH) starting from the initial point and going to the solution point.

The following query concerning iterations will appear on the screen:

```
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF X(IGR)
  2 = ITERATIONS OF X(JGR)
  3 = ITERATIONS OF X(KGR)
  4 = ITERATIONS OF X(IGR), X(JGR)
  5 = ITERATIONS OF X(IGR), X(KGR)
  6 = ITERATIONS OF X(JGR), X(KGR)
  7 = ITERATIONS OF X(IGR), X(JGR), X(KGR)
  0 = RETURN  [ENTER = 0]
```

If the answer for the graph of iterations of X(.) is 1-7, the following query concerning the parameter of the graph will appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
ITERATIONS OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, `[KGR=3]`, `[7 = ITERATIONS OF X(1), X(2), X(3)]`, `[2 = CURVE]`):

```
figure;
x=1:number_of_drawn_values;
x1=[ values_of_the_first_plotting_variable_X(1)_in_each_iteration ];
x2=[ values_of_the_second_plotting_variable_X(2)_in_each_iteration ];
x3=[ values_of_the_third_plotting_variable_X(3)_in_each_iteration ];
h=plot(x,x1,x,x2,x,x3);
set(h,{'LineStyle'},{'-';'--';'-.'});
set(h,{'LineWidth'},{2;2;2});
set(h,{'Color'},{'b';'r';[0 0.5 0]});
tit=title('Iterations of {\color{blue}X(1)}, {\color{red}X(2)},
                                          {\color[rgb]{0 0.5 0}X(3)}');
xl=xlabel('Iteration');
yl=ylabel('Values');
set(gca,'FontSize',20);
set(tit,'FontSize',25);
set(xl,'FontSize',20);
set(yl,'FontSize',20);
axis([1 number_of_drawn_values minimum_in_values maximum_in_values]);
grid on;
print -depsc PMPTH72;
```

The following query concerning function values will appear on the screen:

```
GRAPH OF FUNCTION VALUES:
  1 = VALUES OF F, 0 = RETURN  [ENTER = 0]
```

If the answer for the graph of function values F is 1, the following query concerning the parameter of the graph will appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
VALUES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[2 = CURVE]`):

```
figure;
x=1:number_of_drawn_values;
f=[ function_values_F_in_each_iteration ];
h=plot(x,f);
set(h,{'LineStyle'},{'-'});
set(h,{'LineWidth'},{2});
set(h,{'Color'},{'b'});
tit=title('Iterations of F');
xl=xlabel('Iteration');
yl=ylabel('Values');
set(gca,'FontSize',20);
set(tit,'FontSize',25);
set(xl,'FontSize',20);
set(yl,'FontSize',20);
axis([1 number_of_drawn_values minimum_in_values maximum_in_values]);
grid on;
print -depsc PMPTH02;
```

Moreover, we can draw one path (if $PATH = 'Y') or three paths (if $PATH = 'E') of iterations together with isolines or the color map that correspond to selected pairs of variables. If $PATH<>'N', we have to specify indexes of the selected variables from the keyboard (the very first query, see above). Lower and upper bounds are determined automatically. The following queries appear on the screen:

```
GRAPH OF PATH WITH:
  1 = ISOLINES, 2 = MAP, 0 = RETURN  [ 1 2 12, ENTER = 0 ]
```

If the answer is 1 or 2 or 12, the following queries concerning the parameters of the graph will appear:

```
  TYPE OF PATH: P = POINTS, C = CURVE, B = BOTH  [ENTER = B]
  NUMBER OF CONTOUR LEVELS:  [ENTER = 64]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
  NUMBER OF POINTS: (MAX max_number_of_points)  [ENTER = max_number_of_points]
PATH WITH [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated. If the answer for the graph of path with contains 1 (isolines), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, and the default values):

```
figure;
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values_F ];
x1=[ values_of_the_first_plotting_variable_X(1) ];
x2=[ values_of_the_second_plotting_variable_X(2) ];
xm=value_of_the_first_component_of_the_solution;
```

```
ym=value_of_the_second_component_of_the_solution;
level=64;
contour(x,y,f,level);
mn=min(f(:));
mx=max(f(:));
set(gca, 'CLim', [mn, mx]);
colorbar;
hold on;
h=plot(x1,x2);
set(h,{'LineStyle'},{'-'});
set(h,{'LineWidth'},{1});
set(h,{'Color'},{'k'});
set(h,{'Marker'},{'o'});
set(h,{'MarkerSize'},{4});
set(h,{'MarkerEdgeColor'},{'k'});
set(h,{'MarkerFaceColor'},{'k'});
hold on;
p=plot(xm,ym);
set(p,{'Marker'},{'s'});
set(p,{'MarkerSize'},{10});
set(p,{'MarkerEdgeColor'},{'k'});
set(p,{'MarkerFaceColor'},{'k'});
axis([xligr xuigr xljgr xujgr]);
tit=title('Path with isolines (number_of_points)');
xl=xlabel('X(1)');
yl=ylabel('X(2)');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMPWI13;
```

If the answer for the graph of path with contains 2 (color map), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, and the default values):

```
figure;
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values_F ];
x1=[ values_of_the_first_plotting_variable_X(1) ];
x2=[ values_of_the_second_plotting_variable_X(2) ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=64;
contourf(x,y,f,level);
mn=min(f(:));
```

```
mx=max(f(:));
set(gca, 'CLim', [mn, mx]);
colorbar;
hold on;
h=plot(x1,x2);
set(h,{'LineStyle'},{'-'});
set(h,{'LineWidth'},{1});
set(h,{'Color'},{'w'});
set(h,{'Marker'},{'o'});
set(h,{'MarkerSize'},{4});
set(h,{'MarkerEdgeColor'},{'w'});
set(h,{'MarkerFaceColor'},{'w'});
hold on;
p=plot(xm,ym);
set(p,{'Marker'},{'s'});
set(p,{'MarkerSize'},{10});
set(p,{'MarkerEdgeColor'},{'w'});
set(p,{'MarkerFaceColor'},{'w'});
axis([xligr xuigr xljgr xujgr]);
tit=title('Path with map (number_of_points)');
xl=xlabel('X(1)');
yl=ylabel('X(2)');
set(gca,'FontSize',15);
set(tit,'FontSize',25);
set(xl,'FontSize',15);
set(yl,'FontSize',15);
print -depsc PMPWM13;
```

Here NGR is the number of nodes corresponding to every selected variable (NGR*NGR is the number of drawn function values), XLIGR is the lower bound of the first variable of the selected surface, XUIGR is the upper bound of the first variable of the selected surface, XLJGR is the lower bound of the second variable of the selected surface, XUJGR is the upper bound of the second variable of the selected surface, F contains NGR*NGR drawn function values, X1 are the values of the first selected variable in drawn points, X2 are the values of the second selected variable in drawn points, XM is the value of the solution point whose index is the same as the index of the first variable and YM is the value of the solution point whose index is the same as the index of the second variable. The path is drawn in black in case of isolines and in white in case of the color map. Note that if $PATH='E', then path with isolines and/or color map (if requested) with indices IGR, KGR and JGR, KGR are concurrently generated.

If a system of NE ordinary differential equations is solved, then NED = min(NE, 3) trajectories, corresponding orbits (if NE > 1) and the picture (if NE > 2) can be drawn. If NE > 3, the user must specify indexes IE, JE, KE of the selected solution functions:

```
INDEX OF THE 1-ST VISUALIZED DIFFERENTIAL EQUATION: IE =
INDEX OF THE 2-ND VISUALIZED DIFFERENTIAL EQUATION: JE =
INDEX OF THE 3-RD VISUALIZED DIFFERENTIAL EQUATION: KE =
```

Now, first, the queries concerning the trajectories will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = TRAJECTORY OF YA(IE)
  2 = TRAJECTORY OF YA(JE)
  3 = TRAJECTORY OF YA(KE)
  4 = TRAJECTORIES OF YA(IE), YA(JE)
```

```
  5 = TRAJECTORIES OF YA(IE), YA(KE)
  6 = TRAJECTORIES OF YA(JE), YA(KE)
  7 = TRAJECTORIES OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE < 3, the options of trajectories of YA with missing indexes are omitted. If the answer is 1-7, the following queries concerning the parameters of the graph will appear:

```
  NA = [number_of_mesh_points_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
TRAJECTORY OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]`, `[KE=3]` and `[7 = TRAJECTORIES OF YA(1), YA(2), YA(3)]`):

```
figure;
x=[ values_of_the_independent_variable_AT_mesh_points ];
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
y3=[ values_of_the_third_drawn_function_YA(3) ];
h = plot(x,y1,x,y2,x,y3);
set(h,{'LineWidth'},{2;2;2});
set(h,{'LineStyle'},{'-';'--';'-.'});
set(h,{'Color'},{'b';'r';[0 0.5 0]});
tit=title('Trajectories of {\color{blue}YA(1)}, {\color{red}YA(2)},
                                                {\color[rgb]{0 0.5 0}YA(3)}');
xl=xlabel('Time');
yl=ylabel('Values of YA');
set(gca,'FontSize',20);
set(tit,'FontSize',25);
set(xl,'FontSize',20);
set(yl,'FontSize',20);
axis([minimum_in_x maximum_in_x minimum_in_y's maximum_in_y's]);
grid on;
print -depsc PMTRA07;
```

Here AT are the values of the independent variable (mesh points).

Further, the queries concerning the orbits will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = ORBIT OF YA(IE), YA(JE)
  2 = ORBIT OF YA(IE), YA(KE)
  3 = ORBIT OF YA(JE), YA(KE)
  4 = ORBITS OF YA(IE), YA(JE); YA(IE), YA(KE)
  5 = ORBITS OF YA(IE), YA(JE); YA(JE), YA(KE)
  6 = ORBITS OF YA(IE), YA(KE); YA(JE), YA(KE)
  7 = ORBITS OF YA(IE), YA(JE); YA(IE), YA(KE); YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE $= 2$, only the option of orbit of YA(IE), YA(JE) is considered. If the answer is 1-7, the following query concerning only confirmation of generating the graph will appear:

```
ORBIT OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]` and `[1 = ORBIT OF YA(1), YA(2)]`):

```
figure;
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
h = plot(y1,y2);
set(h,{'LineWidth'},{2});
set(h,{'LineStyle'},{'-'});
set(h,{'Color'},{'b'});
tit=title('Orbit of YA(1), YA(2)');
xl=xlabel('YA(1)');
yl=ylabel('YA(2)');
set(gca,'FontSize',20);
set(tit,'FontSize',25);
set(xl,'FontSize',20);
set(yl,'FontSize',20);
grid on;
axis([minimum_in_y1 maximum_in_y1 minimum_in_y2 maximum_in_y2]);
print -depsc PMORB01;
```

Finally, the queries concerning the picture will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = PICTURE OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

If the answer is 1, the following query concerning only confirmation of generating the graph will appear:

```
PICTURE OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.m` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]`, `[KE=3]`:

```
figure;
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
y3=[ values_of_the_third_drawn_function_YA(3) ];
h = plot3(y1,y2,y3);
set(h,{'LineWidth'},{2});
set(h,{'LineStyle'},{'-'});
set(h,{'Color'},{'b'});
tit=title('Picture of YA(1), YA(2), YA(3)');
xl=xlabel('YA(1)');
yl=ylabel('YA(2)');
zl=zlabel('YA(3)');
set(gca,'FontSize',20);
set(tit,'FontSize',25);
set(xl,'FontSize',20);
set(yl,'FontSize',20);
```

```
set(zl,'FontSize',20);
grid on;
box on;
axis([minimum_in_y1 maximum_in_y1 minimum_in_y2 maximum_in_y2
                                  minimum_in_y3 maximum_in_y3]);
print -depsc PMPIC01;
```

When the file P.m is prepared, we can type MATGO to start the MATLAB system. Procedure MATGO.BAT assures that graphic pictures P*.eps (or P*.png in case $OBR=2) are generated using the instructions written in the file P.m.

5.6 Interface to the SCILAB graphic environment

The SCILAB graphic output can be used in both the MS (32 bit) Windows and the Linux versions of the UFO system if $GRAPHICS = –3 (macrovariable $GRAPHICS is defined in template `UZDCLP.I`, see Section B.7). If we want to use the SCILAB graphic output, we have two possibilities.

The first possibility is the use of internal graphic ($GRAPH='I' or, equivalently, $GRAPH='Y'). In this case, the SCILAB environment is used interactively in a similar way as the internal Fortran environment, but the graphic screen and control statements are different. The answers on the queries concerning particular type of the graph are summarized. After confirming generating the graph, the Scilab Command Window will appear together with the corresponding generated figure. This window can be closed by pressing the Enter key.

The second possibility is the use of external graphic ($GRAPH='E'). In this case, the UFO system generates an output file `P.sci` serving as an input file for the SCILAB graphic environment. Based on the screen interface, the user is asked to specify several parameters in form of answers on the system queries. The parameters are then summarized and after confirming generating the graph, the corresponding SCILAB code is generated.

Note that the default value is $GRAPH='N'.

If INTERNAL GRAPHIC is selected, i.e. $GRAPH='I' or $GRAPH='Y', a list of all possible queries appearing on the screen follows. Depending on the values of variables NF, NA, NC, NGR, NE, the following main menu appears on the screen (see below the main menu for differential equations):

```
V = VARIABLES
A = APPROXIMATIONS
C = CONSTRAINTS
S = ISOLINES/MAP/SURFACE
P = PATH WITH ISOLINES/MAP
X = PATH OF VARIABLES
F = PATH OF FUNCTION VALUES
E = EXIT
```

The choices P, X, F concern drawing paths if $PATH<>'N'. If $PATH='Y', then, at first, we have to specify indexes of visualized variables using the queries

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
```

If V is specified in the main menu, the following queries concerning the values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, appear on the screen:

```
GRAPH OF VARIABLES:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
```

```
  NF = [number_of_variables_NF]
  START INDEX (1..[NF]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NF]):  [ENTER = [NF]]
VARIABLES: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

When KBF=0, the statements corresponding the box constraints are omitted.

If A is specified in the main menu, the following queries concerning the values of approximating functions AF(KA) together with their prescribed values AM(KA), $1 \leq KA \leq NA$, appear on the screen:

```
GRAPH OF APPROXIMATIONS:
  F = AF, M = AF,AM, D = AF-AM, R = RETURN  [ENTER = R]
TYPE OF GRAPH FOR AF:
  P = POINTS, C = CURVE, L = LINES  [ENTER = L]
  NA = [number_of_approximating_functions_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
APPROXIMATIONS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

When KBA=0, the statements corresponding the array AM are omitted.

If C is specified in the main menu, the following queries concerning the values of constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq KC \leq NC$, appear on the screen:

```
GRAPH OF CONSTRAINTS:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
  NC = [number_of_constraints_NC]
  START INDEX (1..[NC]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NC]):  [ENTER = [NC]]
CONSTRAINTS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If S is specified in the main menu, the following queries concerning the reliefs appear on the screen:

```
INDEX OF THE 1-ST VARIABLE:
X(IGR) = the_value_of_X(IGR)
XL(IGR) =
XU(IGR) =
INDEX OF THE 2-ND VARIABLE:
X(JGR) = the_value_of_X(JGR)
XL(JGR) =
XU(JGR) =
GRAPH OF:  I = ISOLINES, M = MAP, S = SURFACE,
           N = NEW INDEXES, R = RETURN  [ENTER = R]
```

Here IGR, JGR are indexes of the visualized variables and XL(IGR), XU(IGR), XL(JGR), XU(JGR) are bounds which specify the domain of the drawn relief. If N is specified, new indexes IGR, JGR can be selected. If I or M or S is specified, the following queries appear:

```
  NUMBER OF CONTOUR LEVELS:  [ENTER = 16]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
GRAPH OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

Note that in case of S, the query concerning the number of contour levels is omitted.

If P is specified in the main menu, the queries concerning the path with isolines or map appearing on the screen depend on macrovariable $PATH:

```
INDEX OF THE 1-ST VARIABLE:
INDEX OF THE 2-ND VARIABLE:
PATH WITH: I = ISOLINES, M = MAP, N = NEW INDEXES, R = RETURN  [ENTER = R]
```

If $PATH='E', the user must specify the indexes of variables while if $PATH='Y', the queries concerning indexes are omitted because they were specified before. If I or M is specified, the following queries appear:

```
  TYPE OF PATH: P = POINTS, C = CURVE, B = BOTH  [ENTER = B]
  NUMBER OF CONTOUR LEVELS:  [ENTER = 16]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
  NUMBER OF POINTS: (MAX max_number_of_points)  [ENTER = max_number_of_points]
PATH WITH [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If X is specified in the main menu, the following queries concerning the path of iterations appear on the screen:

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
INDEX OF THE 3-RD VISUALIZED VARIABLE: KGR =
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF X(IGR)
  2 = ITERATIONS OF X(JGR)
  3 = ITERATIONS OF X(KGR)
  4 = ITERATIONS OF X(IGR), X(JGR)
  5 = ITERATIONS OF X(IGR), X(KGR)
  6 = ITERATIONS OF X(JGR), X(KGR)
  7 = ITERATIONS OF X(IGR), X(JGR), X(KGR)
  0 = RETURN  [ENTER = 0]
```

Here IGR, JGR, KGR are indexes of the visualized variables. If $PATH='Y', the queries concerning indexes are omitted because they were specified before and only options concerning indexes IGR,JGR appear. If 1-7 is specified, the following queries appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
ITERATIONS OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If F is specified in the main menu, the following queries concerning the path of function values appear on the screen:

```
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF F, 0 = RETURN  [ENTER = 0]
```

If 1 is specified, the following queries appear:

```
  TYPE OF GRAPH: 1 = POINTS, 2 = CURVE  [ENTER = 2]
VALUES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If E is specified in the main menu, the program is terminated. Note that if RETURN (R or 0) is specified in particular submenus, then a return to the main menu is performed.

In case of differential equations, the main menu depends on the value of variable NE. One can draw trajectories, corresponding orbits (if NE $>$ 1) and the picture (if NE $>$ 2). If NE $>$ 3, the user must also specify indexes IE, JE, KE of the selected solution functions. The following main menu appears on the screen:

```
INDEX OF THE 1-ST VISUALIZED DIFFERENTIAL EQUATION: IE =
INDEX OF THE 2-ND VISUALIZED DIFFERENTIAL EQUATION: JE =
INDEX OF THE 3-RD VISUALIZED DIFFERENTIAL EQUATION: KE =
T = 1D TRAJECTORY
O = 2D ORBIT
P = 3D PICTURE
N = NEW INDEXES
E = EXIT
```

Note that depending on the value of NE, only the corresponding possible options appear. If N is specified, new indexes can be chosen.

If T is specified in the main menu, the following queries concerning 1D trajectories appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = TRAJECTORY OF YA(IE)
  2 = TRAJECTORY OF YA(JE)
  3 = TRAJECTORY OF YA(KE)
  4 = TRAJECTORIES OF YA(IE), YA(JE)
  5 = TRAJECTORIES OF YA(IE), YA(KE)
  6 = TRAJECTORIES OF YA(JE), YA(KE)
  7 = TRAJECTORIES OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE $<$ 3, only trajectories of YA with a corresponding number of indexes are considered. If 1-7 is specified, the following queries appear:

```
  NA = [number_of_mesh_points_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
TRAJECTORIES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If O is specified in the main menu, the following queries concerning 2D orbits appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = ORBIT OF YA(IE), YA(JE)
  2 = ORBIT OF YA(IE), YA(KE)
  3 = ORBIT OF YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE = 2, only the first option is possible. If 1-3 is specified, the following query appears:

```
ORBIT OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If P is specified in the main menu, the following queries concerning 3D picture appear on the screen:

```
DIFFERENTIAL EQUATIONS:
  1 = PICTURE OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

If 1 is specified, the following query appears:

```
PICTURE OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

If E is specified in the main menu, the program is terminated. Note that if RETURN (0) is specified in particular submenus, then a return to the main menu is performed.

If EXTERNAL GRAPHIC is selected, i.e. $GRAPH='E', a list of all possible queries appearing on the screen follows.

If $PATH='Y' (two variables IGR,JGR) or $PATH='E' (three variables IGR,JGR,KGR), then at first the user is asked to specify the visualized variables for the paths:

```
INDEX OF THE 1-ST VISUALIZED VARIABLE: IGR =
INDEX OF THE 2-ND VISUALIZED VARIABLE: JGR =
INDEX OF THE 3-RD VISUALIZED VARIABLE: KGR =
```

Now, depending on the values of variables NF, NA, NC, NGR, NE, the user is asked to specified the types of graphs and their parameters that will be generated into the file `P.sci`. Note that the choice RETURN (R or 0, see below) from the main menu will cause a switch to the main menu of the next graph and from the submenu will cause a return to the main menu of the current graph.

If NF > 0, the query

```
GRAPH OF VARIABLES:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
```

concerning the values of variables X(I) together with their bounds XL(I) and XU(I), $1 \leq I \leq NF$, appears. If the answer is P or C, the following queries concerning the parameters of the graph will appear:

```
  NF = [number_of_variables_NF]
  START INDEX (1..[NF]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NF]):  [ENTER = [NF]]
VARIABLES: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[P = POINTS]`):

```
ncf=ncf+1;
scf(ncf);
il=[ indices_of_lower_bounds_for_variables ];
xl=[ values_of_lower_bounds_for_variables ];
iu=[ indices_of_upper_bounds_for_variables ];
xu=[ values_of_upper_bounds_for_variables ];
ix=[ indices_of_variables ];
xi=[ values_of_variables ];
plot(iu,xu,il,xl,ix,xi);
p=get("hdl");
p.children(1).line_mode="off";
p.children(1).mark_mode="on";
```

```
p.children(1).mark_size=6;
p.children(1).mark_style=9;
p.children(1).mark_foreground=5;
p.children(1).mark_background=5;
p.children(2).line_mode="off";
p.children(2).mark_mode="on";
p.children(2).mark_size=12;
p.children(2).mark_style=6;
p.children(2).mark_foreground=2;
p.children(2).mark_background=2;
p.children(3).line_mode="off";
p.children(3).mark_mode="on";
p.children(3).mark_size=12;
p.children(3).mark_style=7;
p.children(3).mark_foreground=2;
p.children(3).mark_background=2;
for i=1:length(il)
plot([il(i) il(i)],[xl(i) xi(il(i))]);
pl=get("hdl");
pl.children.line_mode="on";
pl.children.polyline_style=1;
pl.children.line_style=1;
pl.children.thickness=1;
pl.children.foreground=2;
end
for i=1:length(iu)
plot([iu(i) iu(i)],[xu(i) xi(iu(i))]);
pu=get("hdl");
pu.children.line_mode="on";
pu.children.polyline_style=1;
pu.children.line_style=1;
pu.children.thickness=1;
pu.children.foreground=2;
end
a=gca();
a.font_size=3;
a.zoom_box=[0, minimum_in_values, length(ix)+1, maximum_in_values];
title('Variables','fontsize',5);
xlabel('Indices of variables','fontsize',3);
ylabel('Bounds and values of variables','fontsize',3);
xs2eps(ncf,'PSVAR01');
```

When KBF=0, the statements corresponding the box constraints are omitted.

If NA $>$ 0, the query

```
GRAPH OF APPROXIMATIONS:
  F = AF, M = AF,AM, D = AF-AM, R = RETURN  [ENTER = R]
```

concerning the values of approximating functions AF(KA) together with their prescribed values AM(KA), $1 \leq$ KA $\leq$ NA, appears. If the answer for the graph is F or M or D, the following queries concerning the parameters of the graph will appear:

```
TYPE OF GRAPH FOR AF:
```

```
  P = POINTS, C = CURVE, L = LINES  [ENTER = L]
  NA = [number_of_approximating_functions_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
APPROXIMATIONS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in cases of `[D = AF-AM]` and `[L = LINES]`):

```
ncf=ncf+1;
scf(ncf);
ii=1:number_of_drawn_values;
dif=[ values_of_differences_between_AF(KA)_and_AM(KA) ];
for i=1:length(ii)
plot([ii(i) ii(i)],[0 dif(i)],"Color","red","Thickness",0.5);
end
a=gca();
a.font_size=3;
a.zoom_box=[0, minimum_in_values length(ii)+1, maximum_in_values];
title('Approximations','fontsize',5);
xlabel('Indices of approximations','fontsize',3);
ylabel('Differences of approximations','fontsize',3);
xs2eps(ncf,'PSAPP09');
```

When KBA=0, the statements corresponding the array AM are omitted.

If NC $>$ 0, the query

```
GRAPH OF CONSTRAINTS:
  P = POINTS, C = CURVE, R = RETURN  [ENTER = R]
```

concerning the values of constraint functions CF(KC) together with their bounds CL(KC) and CU(KC), $1 \leq$ KC $\leq$ NC, appears. If the answer is P or C, the following queries concerning the parameters of the graph will appear:

```
  NC = [number_of_constraints_NC]
  START INDEX (1..[NC]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NC]):  [ENTER = [NC]]
CONSTRAINTS: [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[P = POINTS]`):

```
ncf=ncf+1;
scf(ncf);
il=[ indices_of_lower_bounds_for_constraints ];
cl=[ values_of_lower_bounds_for_constraints ];
iu=[ indices_of_upper_bounds_for_constraints ];
cu=[ values_of_upper_bounds_for_constraints ];
ic=[ indices_of_constraints ];
ci=[ values_of_constraints ];
plot(iu,cu,il,cl,ic,ci);
```

```
p=get("hdl");
p.children(1).line_mode="off";
p.children(1).mark_mode="on";
p.children(1).mark_size=6;
p.children(1).mark_style=9;
p.children(1).mark_foreground=5;
p.children(1).mark_background=5;
p.children(2).line_mode="off";
p.children(2).mark_mode="on";
p.children(2).mark_size=12;
p.children(2).mark_style=6;
p.children(2).mark_foreground=2;
p.children(2).mark_background=2;
p.children(3).line_mode="off";
p.children(3).mark_mode="on";
p.children(3).mark_size=12;
p.children(3).mark_style=7;
p.children(3).mark_foreground=2;
p.children(3).mark_background=2;
for i=1:length(il)
plot([il(i) il(i)],[cl(i) ci(il(i))]);
pl=get("hdl");
pl.children.line_mode="on";
pl.children.polyline_style=1;
pl.children.line_style=1;
pl.children.thickness=1;
pl.children.foreground=2;
end
for i=1:length(iu)
plot([iu(i) iu(i)],[cu(i) ci(iu(i))]);
pu=get("hdl");
pu.children.line_mode="on";
pu.children.polyline_style=1;
pu.children.line_style=1;
pu.children.thickness=1;
pu.children.foreground=2;
end
a=gca();
a.font_size=3;
a.zoom_box=[0, minimum_in_values, length(ic)+1, maximum_in_values];
title('Constraints','fontsize',5);
xlabel('Indices of constraints','fontsize',3)    ;
ylabel('Bounds and values of constraints','fontsize',3);
xs2eps(ncf,'PSCON1');
```

Generation of reliefs depends on the value of macrovariable $MAP ('N'- no, 'Y'- yes, 'E'- extended). If $MAP='Y', then one relief (isolines, color map or surface) is generated, while if $MAP='E', then three reliefs are generated as is specified by the following queries:

```
GRAPH OF MAPS:
  S = ISOLINES/MAP/SURFACE, R = RETURN  [ENTER = R]
```

If S is specified, then these queries follow:

```
INDEX OF THE 1-ST VARIABLE:
X(IGR) = the_value_of_X(IGR)
XL(IGR) =
XU(IGR) =
INDEX OF THE 2-ND VARIABLE:
X(JGR) = the_value_of_X(JGR)
XL(JGR) =
XU(JGR) =
INDEX OF THE 3-RD VARIABLE:
X(KGR) = the_value_of_X(KGR)
XL(KGR) =
XU(KGR) =
GRAPH OF:  1 = ISOLINES, 2 = MAP, 3 = SURFACE, 9 = NEW INDEXES, 0 = RETURN
  [ 1 2 3 12 13 23 123 9, ENTER = 0 ]
```

Here IGR, JGR, KGR are indexes of the visualized variables and XL(IGR), XU(IGR), XL(JGR), XU(JGR), XL(KGR), XU(KGR) are bounds which specify the domain of the drawn relief. If 9 is specified, new indexes IGR, JGR, KGR can be chosen. If 1 or 2 or 3 or 12 or 13 or 23 or 123 is specified, the following queries will appear:

```
  NUMBER OF CONTOUR LEVELS:  [ENTER = 16]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
GRAPH OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated. If the answer for the graph contains 1 (isolines), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
ncf=ncf+1;
scf(ncf);
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=16;
xset("fpf"," ");
ff=f';
contour(x,y,ff,level);
plot(xm,ym);
pi=get("hdl");
pi.children.line_mode="off";
pi.children.mark_mode="on";
pi.children.mark_style=11;
pi.children.mark_size=10;
pi.children.mark_foreground=-1;
```

```
pi.children.mark_background=-1;
fi=gcf();
fi.color_map=jetcolormap(level);
mn=min(f);
mx=max(f);
colorbar(mn,mx);
ai=gca();
ai.font_size=3;
ai.box="on";
ai.zoom_box=[xligr, xljgr, xuigr, xujgr];
tit=title('Isolines','fontsize',5);
xlabel('X(1)','fontsize',3);
ylabel('X(2)','fontsize',3);
xs2eps(ncf,'PSISO01');
```

If the answer for the graph contains 2 (color map), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
ncf=ncf+1;
scf(ncf);
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_X(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=16;
xset("fpf"," ");
ff=f';
Sgrayplot(x,y,ff);
plot(xm,ym);
pm=get("hdl");
pm.children.line_mode="off";
pm.children.mark_mode="on";
pm.children.mark_style=11;
pm.children.mark_size=10;
pm.children.mark_foreground=-2;
pm.children.mark_background=-2;
fm=gcf();
fm.color_map=jetcolormap(level);
mn=min(f);
mx=max(f);
colorbar(mn,mx);
am=gca();
am.font_size=3;
am.auto_ticks="on";
am.box="on";
```

```
am.zoom_box=[xligr, xljgr, xuigr, xujgr];
tit=title('Map','fontsize',5);
xlabel('X(1)','fontsize',3);
ylabel('X(2)','fontsize',3);
xs2eps(ncf,'PSMAP01');
```

If the answer for the graph contains 3 (surface), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`):

```
ncf=ncf+1;
scf(ncf);
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable;
xuigr=upper_bound_of_the_first_variable;
xljgr=lower_bound_of_the_second_variable;
xujgr=upper_bound_of_the_second_variable;
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
xset("fpf"," ");
surf(x,y,f);
fs=gcf();
fs.color_map=jetcolormap(16);
mn=min(f);
mx=max(f);
colorbar(mn,mx);
as=gca();
as.font_size=3;
as.zoom_box=[xligr, xljgr, xuigr, xujgr, mn, mx];
tit=title('Surface','fontsize',5);
xlabel('X(1)','fontsize',3);
ylabel('X(2)','fontsize',3);
zlabel('F','fontsize',3);
xs2eps(ncf,'PSSUR01');
```

Here NGR is the number of nodes corresponding to every selected variable (NGR*NGR is the number of drawn function values), XLIGR is the lower bound of the first variable of the selected surface specified by the user, XUIGR is the upper bound of the first variable of the selected surface specified by the user, XLJGR is the lower bound of the second variable of the selected surface specified by the user, XUJGR is the upper bound of the second variable of the selected surface specified by the user, F contains NGR*NGR drawn function values, XM is the value of the solution point whose index is the same as the index of the first variable and YM is the value of the solution point whose index is the same as the index of the second variable. The solution point [XM,YM] is drawn only if it is contained in the box defined by the values XLIGR, XUIGR, XLJGR, XUJGR and is marked as a black square in case of isolines and as a white square in case of the color map. Note that if $MAP='E', then isolines, color map, and/or surface (if requested) with indices IGR, KGR and JGR, KGR are concurrently generated.

The last part of the code concerns drawing paths if $PATH<>'N'. We can draw iterations of up to three variables X(IGR), X(JGR), X(KGR) and the function value F (depending on the variable $PATH) starting from the initial point and going to the solution point.

The following query concerning iterations will appear on the screen:

```
GRAPH OF ITERATIONS:
  1 = ITERATIONS OF X(IGR)
  2 = ITERATIONS OF X(JGR)
  3 = ITERATIONS OF X(KGR)
  4 = ITERATIONS OF X(IGR), X(JGR)
  5 = ITERATIONS OF X(IGR), X(KGR)
  6 = ITERATIONS OF X(JGR), X(KGR)
  7 = ITERATIONS OF X(IGR), X(JGR), X(KGR)
  0 = RETURN  [ENTER = 0]
```

If the answer for the graph of iterations of X(.) is 1-7, the following query concerning the parameter of the graph will appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
ITERATIONS OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, `[KGR=3]`, `[7 = ITERATIONS OF X(1), X(2), X(3)]`, `[2 = CURVE]`):

```
ncf=ncf+1;
scf(ncf);
x=1:number_of_drawn_values;
x1=[ values_of_the_first_plotting_variable_X(1)_in_each_iteration ];
x2=[ values_of_the_first_plotting_variable_X(2)_in_each_iteration ];
x3=[ values_of_the_first_plotting_variable_X(3)_in_each_iteration ];
plot(x,x1,x,x2,x,x3);
e=gce();
e.children(1).line_mode="on";
e.children(1).mark_mode="off";
e.children(1).polyline_style=1;
e.children(1).line_style=4;
e.children(1).thickness=2;
e.children(1).foreground=13;
e.children(2).line_mode="on";
e.children(2).mark_mode="off";
e.children(2).polyline_style=1;
e.children(2).line_style=3;
e.children(2).thickness=2;
e.children(2).foreground=5;
e.children(3).line_mode="on";
e.children(3).mark_mode="off";
e.children(3).polyline_style=1;
e.children(3).line_style=1;
e.children(3).thickness=2;
e.children(3).foreground=2;
title('$\textrm{Iterations of \textcolor{blue}{X(1)}, \textcolor{red}{X(2)},
                              \textcolor{0,0.5,0}{X(3)}}$','fontsize',5);
set(gca(),"grid",[1 1]);
a=gca();
a.font_size=4;
a.zoom_box=[1 minimum_in_values number_of_drawn_values maximum_in_values];
a.thickness=0.5;
```

```
xlabel('Iteration','fontsize',4);
ylabel('Values','fontsize',4);
xs2eps(ncf,'PSPTH72');
```

The following query concerning function values will appear on the screen:

```
GRAPH OF FUNCTION VALUES:
  1 = VALUES OF F, 0 = RETURN  [ENTER = 0]
```

If the answer for the graph of function values F is 1, the following query concerning the parameter of the graph will appear:

```
  TYPE OF GRAPH: P = POINTS, C = CURVE  [ENTER = C]
VALUES OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[2 = CURVE]`):

```
ncf=ncf+1;
scf(ncf);
x=1:number_of_drawn_values;
f=[ function_values_F_in_each_iteration ];
plot(x,f);
e=gce();
e.children.line_mode="on";
e.children.mark_mode="off";
e.children.polyline_style=1;
e.children.line_style=1;
e.children.thickness=2;
e.children.foreground=2;
title('Iterations of F','fontsize',5);
set(gca(),"grid",[1 1]);
a=gca();
a.font_size=4;
a.zoom_box=[1 minimum_in_values number_of_drawn_values maximum_in_values];
a.thickness=0.5;
xlabel('Iteration','fontsize',4);
ylabel('Values','fontsize',4);
xs2eps(ncf,'PSPTH02');
```

Moreover, we can draw one path (if \$PATH = 'Y') or three paths (if \$PATH = 'E') of iterations together with isolines or the color map that correspond to selected pairs of variables. If \$PATH<>'N', we have to specify indexes of the selected variables from the keyboard (the very first query, see above). Lower and upper bounds are determined automatically. The following queries appear on the screen:

```
GRAPH OF PATH WITH:
  1 = ISOLINES, 2 = MAP, 0 = RETURN  [ 1 2 12, ENTER = 0 ]
```

If the answer is 1 or 2 or 12, the following queries concerning the parameters of the graph will appear:

```
  TYPE OF PATH: P = POINTS, C = CURVE, B = BOTH  [ENTER = B]
  NUMBER OF CONTOUR LEVELS:  [ENTER = 16]
  SCALE: 1 = LINEAR, 2 = LOGARITHMIC  [ENTER = 1]
```

```
  NUMBER OF POINTS: (MAX max_number_of_points)  [ENTER = max_number_of_points]
PATH WITH [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated. If the answer for the graph of path with contains 1 (isolines), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, and the default values):

```
ncf=ncf+1;
scf(ncf);
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values_F ];
x1=[ values_of_the_first_plotting_variable_X(1) ];
x2=[ values_of_the_second_plotting_variable_X(2) ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=16;
xset("fpf"," ");
ff=f';
contour(x,y,ff,level);
plot(xm,ym);
p=get("hdl");
p.children.line_mode="off";
p.children.mark_mode="on";
p.children.mark_style=11;
p.children.mark_size=10;
p.children.mark_foreground=-1;
p.children.mark_background=-1;
plot(x1,x2);
e=gce();
e.children.line_mode="on";
e.children.mark_mode="on";
e.children.polyline_style=1;
e.children.line_style=1;
e.children.thickness=2;
e.children.foreground=-1;
e.children.mark_style=9;
e.children.mark_size=4;
e.children.mark_foreground=-1;
e.children.mark_background=-1;
fw=gcf();
fw.color_map=jetcolormap(level);
mn=min(f);
mx=max(f);
colorbar(mn,mx);
```

```
a=gca();
a.font_size=3;
a.box="on";
a.zoom_box=[xligr, xljgr, xuigr, xujgr];
title('Path with isolines (number_of_points)','fontsize',5);
xlabel('X(1)','fontsize',4);
ylabel('X(2)','fontsize',4);
xs2eps(ncf,'PSPWI13');
```

If the answer for the graph of path with contains 2 (color map), the code has the form (e.g. in case of `[IGR=1]`, `[JGR=2]`, and the default values):

```
ncf=ncf+1;
scf(ncf);
ngr=number_of_nodes;
xligr=lower_bound_of_the_first_variable_XL(1);
xuigr=upper_bound_of_the_first_variable_XU(1);
xljgr=lower_bound_of_the_second_variable_XL(2);
xujgr=upper_bound_of_the_second_variable_XU(2);
stx=(xuigr-xligr)/(ngr-1);
sty=(xujgr-xljgr)/(ngr-1);
x=xligr:stx:xuigr;
y=xljgr:sty:xujgr;
f=[ drawn_function_values_F ];
x1=[ values_of_the_first_plotting_variable_X(1) ];
x2=[ values_of_the_second_plotting_variable_X(2) ];
xm=value_of_the_first_component_of_the_solution;
ym=value_of_the_second_component_of_the_solution;
level=16;
xset("fpf"," ");
ff=f';
Sgrayplot(x,y,ff);
plot(xm,ym);
p=get("hdl");
p.children.line_mode="off";
p.children.mark_mode="on";
p.children.mark_style=11;
p.children.mark_size=10;
p.children.mark_foreground=-2;
p.children.mark_background=-2;
plot(x1,x2);
e=gce();
e.children.line_mode="on";
e.children.mark_mode="on";
e.children.polyline_style=1;
e.children.line_style=1;
e.children.thickness=2;
e.children.foreground=-2;
e.children.mark_style=9;
e.children.mark_size=4;
e.children.mark_foreground=-2;
e.children.mark_background=-2;
fw=gcf();
```

```
fw.color_map=jetcolormap(level);
mn=min(f);
mx=max(f);
colorbar(mn,mx);
a=gca();
a.font_size=3;
a.auto_ticks="on";
a.box="on";
a.zoom_box=[xligr, xljgr, xuigr, xujgr];
title('Path with map (number_of_points)','fontsize',5);
xlabel('X(1)','fontsize',4);
ylabel('X(2)','fontsize',4);
xs2eps(ncf,'PSPWM13');
```

Here NGR is the number of nodes corresponding to every selected variable (NGR*NGR is the number of drawn function values), XLIGR is the lower bound of the first variable of the selected surface, XUIGR is the upper bound of the first variable of the selected surface, XLJGR is the lower bound of the second variable of the selected surface, XUJGR is the upper bound of the second variable of the selected surface, F contains NGR*NGR drawn function values, X1 are the values of the first selected variable in drawn points, X2 are the values of the second selected variable in drawn points, XM is the value of the solution point whose index is the same as the index of the first variable and YM is the value of the solution point whose index is the same as the index of the second variable. The path is drawn in black in case of isolines and in white in case of the color map. Note that if $PATH='E', then path with isolines and/or color map (if requested) with indices IGR, KGR and JGR, KGR are concurrently generated.

If a system of NE ordinary differential equations is solved, then NED = min(NE, 3) trajectories, corresponding orbits (if NE > 1) and the picture (if NE > 2) can be drawn. If NE > 3, the user must specify indexes IE, JE, KE of the selected solution functions:

```
INDEX OF THE 1-ST VISUALIZED DIFFERENTIAL EQUATION: IE =
INDEX OF THE 2-ND VISUALIZED DIFFERENTIAL EQUATION: JE =
INDEX OF THE 3-RD VISUALIZED DIFFERENTIAL EQUATION: KE =
```

Now, first, the queries concerning the trajectories will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = TRAJECTORY OF YA(IE)
  2 = TRAJECTORY OF YA(JE)
  3 = TRAJECTORY OF YA(KE)
  4 = TRAJECTORIES OF YA(IE), YA(JE)
  5 = TRAJECTORIES OF YA(IE), YA(KE)
  6 = TRAJECTORIES OF YA(JE), YA(KE)
  7 = TRAJECTORIES OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE < 3, the options of trajectories of YA with missing indexes are omitted. If the answer is 1-7, the following queries concerning the parameters of the graph will appear:

```
  NA = [number_of_mesh_points_NA]
  START INDEX (1..[NA]):  [ENTER = 1]
  STOP INDEX ([start_index]..[NA]):  [ENTER = [NA]]
TRAJECTORY OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]`, `[KE=3]` and `[7 = TRAJECTORIES OF YA(1), YA(2), YA(3)]`):

```
ncf=ncf+1;
scf(ncf);
x=[ values_of_the_independent_variable_AT_mesh_points ];
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
y3=[ values_of_the_third_drawn_function_YA(3) ];
plot(x,y1,x,y2,x,y3);
set(gca(),"grid",[1 1]);
a=gca();
a.font_size=4;
a.zoom_box=[minimum_in_x minimum_in_y's maximum_in_x maximum_in_y's];
a.thickness=0.5;
e=gce();
e.children(1).polyline_style=1;
e.children(1).line_style=4;
e.children(1).thickness=2;
e.children(1).foreground=13;
e.children(2).polyline_style=1;
e.children(2).line_style=3;
e.children(2).thickness=2;
e.children(2).foreground=5;
e.children(3).polyline_style=1;
e.children(3).line_style=1;
e.children(3).thickness=2;
e.children(3).foreground=2;
title('$\textrm{Trajectories of \textcolor{blue}{YA(1)}, \textcolor{red}{YA(2)},
                              \textcolor{0,0.5,0}{YA(3)}}$','fontsize',5);
xlabel('Time','fontsize',4);
ylabel('Values of YA','fontsize',4);
xs2eps(ncf,'PSTRA07');
```

Here AT are the values of the independent variable (mesh points).

Further, the queries concerning the orbits will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = ORBIT OF YA(IE), YA(JE)
  2 = ORBIT OF YA(IE), YA(KE)
  3 = ORBIT OF YA(JE), YA(KE)
  4 = ORBITS OF YA(IE), YA(JE); YA(IE), YA(KE)
  5 = ORBITS OF YA(IE), YA(JE); YA(JE), YA(KE)
  6 = ORBITS OF YA(IE), YA(KE); YA(JE), YA(KE)
  7 = ORBITS OF YA(IE), YA(JE); YA(IE), YA(KE); YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

Note that if NE = 2, only the option of orbit of YA(IE), YA(JE) is considered. If the answer is 1-7, the following query concerning only confirmation of generating the graph will appear:

```
ORBIT OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]` and `[1 = ORBIT OF YA(1), YA(2)]`):

```
ncf=ncf+1;
scf(ncf);
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
plot(y1,y2);
set(gca(),"grid",[1 1]);
a=gca();
a.font_size=4;
a.zoom_box=[minimum_in_y1 minimum_in_y2 maximum_in_y1 maximum_in_y2];
a.thickness=0.5;
e=gce();
e.children(1).polyline_style=1;
e.children(1).line_style=1;
e.children(1).thickness=2;
e.children(1).foreground=2;
title('Orbit of YA(1), YA(2)','fontsize',5);
xlabel('YA(1)','fontsize',4);
ylabel('YA(2)','fontsize',4);
xs2eps(ncf,'PSORB01');
```

Finally, the queries concerning the picture will appear:

```
DIFFERENTIAL EQUATIONS:
  1 = PICTURE OF YA(IE), YA(JE), YA(KE)
  0 = RETURN  [ENTER = 0]
```

If the answer is 1, the following query concerning only confirmation of generating the graph will appear:

```
PICTURE OF [parameters_of_the_graph_specified_by_the_user]
  GENERATE GRAPH? 'N' = NO  [ENTER = YES]
```

After confirming generating the graph, a corresponding part of file `P.sci` is generated and has the form (e.g. in case of `[IE=1]`, `[JE=2]`, `[KE=3]`:

```
ncf=ncf+1;
scf(ncf);
y1=[ values_of_the_first_drawn_function_YA(1) ];
y2=[ values_of_the_second_drawn_function_YA(2) ];
y3=[ values_of_the_third_drawn_function_YA(3) ];
param3d(y1,y2,y3);
set(gca(),"grid",[1 1]);
a=gca();
a.font_size=4;
a.zoom_box=[minimum_in_y1 minimum_in_y2 maximum_in_y1 maximum_in_y2
                                        minimum_in_y3 maximum_in_y3];
a.thickness=0.5;
e=gce();
e.polyline_style=1;
e.line_style=1;
e.thickness=2;
e.foreground=color('blue');
```

```
title('Picture of YA(1), YA(2), YA(3)','fontsize',5);
xlabel('YA(1)','fontsize',4);
ylabel('YA(2)','fontsize',4);
zlabel('YA(3)','fontsize',4);
xs2eps(ncf,'PSPIC01');
```

When the file P.sci is prepared, we can type SCIGO to start the SCILAB system. Procedure SCIGO.BAT assures that graphic pictures P*.eps (or P*.png in case $OBR=2) are generated using the instructions written in the file P.sci.

5.7 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 determine what is printed and the macrovariable $LOUT has an influence on the extent of the print.

The macrovariable $KOUT can have the following values:

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

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

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

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

$LOUT= 0 -	Basic output. The basic information (one row if $KOUT=1) is printed in each accepted iteration.
$LOUT=± 1 -	Extended output. Additional scalars, together with the vector of variables, are printed.
$LOUT=± 2 -	Extended output. Additional vectors (usually gradients) are printed.
$LOUT=± 3 -	Extended output. Additional matrices (usually Hessian matrices) are printed.
$LOUT=± 4 -	The most extended output. All useful data are printed.

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

The macrovariable $LOUT has an additional significance. If $KOUT=0 and $LOUT>0, a copy of the basic screen output is performed (see Section 5.1). If $KOUT=0 and $LOUT<0, paper saving print for line printers 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 text file output, we use values $KOUT=5, $LOUT=0 in the problem specification introduced in Section 7.1. In this case, text file `P.OUT` contains the following information.

```
UNCONSTRAINED MAXIMIZATION USING UFO SYSTEM
-------------------------------------------

OPTIMIZATION SUBROUTINE : U1FDU1
DIRECTION DETERMINATION : UDDLG1
STEP SIZE DETERMINATION : US1L01
FUNCTION  DETERMINATION : UF1F01
GRADIENT  DETERMINATION :
H MATRIX  DETERMINATION :
VARIABLE METRIC  UPDATE : UUDBG1

PROBLEM
-------
NF=    5                 KDF=  1 KSF=  1 KCF=  2 KBF=  2 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

NIT=    0 NFV=    1 NFG=    1 F=0.187D+01 G=0.667D-01 D=0.000D+00

-------------------------------------------
DIRECTION DETERMINATION USING UDDLG1
   0  ALF=0.000D+00 SIG=0.000D+00
END OF DIRECTION DETERMINATION : G.M. POS
G   =   0.11547D+00  S   =   0.11547D+00  P   =  -0.13333D-01
STEPSIZE SELECTION USING US1L01  : P/(G*S) = -.100D+01
       R=0.000D+00 S=0.115D+00 F=0.186666666667D+01 P=-.133D-01 B=0.150D+02
0   0  R=0.100D+01 D=0.115D+00 F=0.185288395062D+01 P=-.142D-01 INIT-1
1  -1  R=0.400D+01 D=0.462D+00 F=0.180590617284D+01 P=-.171D-01 BISECT.
1  -2  R=0.150D+02 D=0.173D+01 F=0.155000000000D+01 P=-.300D-01 BISECT.
       R=0.150D+02 D=0.173D+01 F=0.155000000000D+01
END OF STEPSIZE SELECTION : R BOUND
VARIABLE METRIC UPDATE USING UUDBG1
GAM=0.000D+00 RHO=0.000D+00 PAR=0.000D+00 BET=0.000D+00
A  =0.000D+00 B  =-.250D+00 C  =0.000D+00
END OF VARIABLE METRIC UPDATE : B - NEG.
CONSTRAINT ADDITION USING UYADS1
NEW= -3  N=  2
END OF CONSTRAINT ADDITION : N =   2   IER =   0

NIT=    1 NFV=    4 NFG=    4 F=0.155D+01 G=0.150D+00 D=0.333D+00
UNIT      G.M. POS  R BOUND   NORMAL
-------------------------------------------
DIRECTION DETERMINATION USING UDDLG1
   0  ALF=0.000D+00 SIG=0.000D+00
END OF DIRECTION DETERMINATION : G.M. POS
G   =   0.21213D+00  S   =   0.21213D+00  P   =  -0.45000D-01
STEPSIZE SELECTION USING US1L01  : P/(G*S) = -.100D+01
       R=0.000D+00 S=0.212D+00 F=0.155000000000D+01 P=-.450D-01 B=0.667D+01
0   0  R=0.100D+01 D=0.212D+00 F=0.150387500000D+01 P=-.473D-01 INIT-1
1  -1  R=0.400D+01 D=0.849D+00 F=0.135200000000D+01 P=-.540D-01 BISECT.
1  -2  R=0.667D+01 D=0.141D+01 F=0.120000000000D+01 P=-.600D-01 CUBIC
       R=0.667D+01 D=0.141D+01 F=0.120000000000D+01
```

```
END OF STEPSIZE SELECTION : R BOUND
VARIABLE METRIC UPDATE USING UUDBG1
GAM=0.000D+00 RHO=0.000D+00 PAR=0.000D+00 BET=0.000D+00
A  =0.000D+00 B  =-.100D+00 C  =0.000D+00
END OF VARIABLE METRIC UPDATE : B - NEG.
CONSTRAINT ADDITION USING UYADS1
NEW= -4  N=  1
END OF CONSTRAINT ADDITION : N =   1   IER =   0
 NIT=    2 NFV=    7 NFG=    7 F=0.120D+01 G=0.200D+00 D=0.250D+00
UNIT      G.M. POS  R BOUND   UPDATE
-----------------------------------------
DIRECTION DETERMINATION USING UDDLG1
   0  ALF=0.000D+00 SIG=0.000D+00
END OF DIRECTION DETERMINATION : G.M. POS
G   =   0.20000D+00  S   =   0.20000D+00  P   =  -0.40000D-01
STEPSIZE SELECTION USING US1L01  : P/(G*S) = -.100D+01
       R=0.000D+00 S=0.200D+00 F=0.120000000000D+01 P=-.400D-01 B=0.500D+01
0   0  R=0.100D+01 D=0.200D+00 F=0.116000000000D+01 P=-.400D-01 INIT-1
1  -1  R=0.500D+01 D=0.100D+01 F=0.100000000000D+01 P=-.400D-01 CUBIC
       R=0.500D+01 D=0.100D+01 F=0.100000000000D+01
END OF STEPSIZE SELECTION : R BOUND
VARIABLE METRIC UPDATE USING UUDBG1
GAM=0.000D+00 RHO=0.000D+00 PAR=0.000D+00 BET=0.000D+00
A  =0.000D+00 B  =0.000D+00 C  =0.000D+00
END OF VARIABLE METRIC UPDATE : B - NEG.
CONSTRAINT ADDITION USING UYADS1
NEW= -5  N=  0
END OF CONSTRAINT ADDITION : N =   0   IER =   0
NIT=    3 NFV=    9 NFG=    9 F=0.100D+01 G=0.000D+00 D=0.200D+00
UNIT      G.M. POS  R BOUND   UPDATE
-----------------------------------------

FINAL RESULTS
-------------

FF= -0.1000000000D+01
X =  0.1000000000D+01  0.2000000000D+01  0.3000000000D+01  0.4000000000D+01
     0.5000000000D+01

TERMINATION:   4  GRAD TOL  F=0.100D+01 G=0.000D+00 D=0.200D+00

STATISTICS
----------
NIT  =    3   NDEC =    0
NFV  =    9   NAV  =    0   NCV  =    0   NRES =    3
NFG  =    9   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 the line containing the starting function value and gradient norm (zero iteration), results of individual iterations follow. Since $KOUT=5, information concerning direction determination, step-size selection, variable

metric update and constraint handling is printed. The last two lines describing every iteration contain the current number of iterations NIT, function evaluations NFV, gradient evaluations NFG, the function value F, the gradient norm G, the step-size D and the information concerning the status of the current iteration (`UNIT`, `G.M. POS`, `R BOUND`, `UPDATE`). After results of individual iterations, the final results are printed. They contain the cause of termination ITERM=4 (GRAD TOL) corresponding to the attainment of the required gradient norm. Furthermore, 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 final number of iterations NIT, (objective, particular, constraint) function evaluations (NFV, NAV, NCV), (objective, particular, constraint) gradient evaluations (NFG, NAG, NCG), (objective, particular, constraint) Hessian matrix evaluations (NFH, NAH, NCH), the number of decompositions NDEC, the number of restarts NRES, and the numbers of constraint deletions NREM and additions NADD.

5.8 User supplied output

The UFO system allows utilizing both the user supplied output subroutines and the post-processing subroutines. These subroutines can be inserted in the UFO source program by using the macrovariable $OUTPUT:

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

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

5.9 Storing final results

If we set $OUTPUTDATA='Y', the final values of variables X(I), $1 \leq$ I$\leq$ NF, are stored in file `P.DAT`. Similarly, if we set $INPUTDATA='Y', the values of variables X(I), $1 \leq$ I$\leq$ NF, from file `P.DAT` are used as input data for a new optimization process.

5.10 Other output files

The UFO system uses three other output files `P.DIM`, `P.SIF` and `P.PAT` which contain additional information about the problem solved. File `P.DIM` shows us the problem dimension. It contains the numbers of variables, approximating functions, constraints and also the numbers of nonzero elements in sparse structures. For example, if we apply the UFO system to the input file `TSIF21.UFO`, then the file `P.DIM` contains the following text:

```
PROBLEM: NEXT =       0
NUMBER OF VARIABLES:         NF  =     450
NUMBER OF CONSTRAINTS:       NC  =     360
NUMBER OF NONZERO ELEMENTS: MC   =    1576
NUMBER OF NONZERO ELEMENTS: MHC  =      55
NUMBER OF NONZERO ELEMENTS: MCH  =    4736
NUMBER OF NONZERO ELEMENTS: M    =    2779
```

File `P.SIF` contains information concerning SIF files of the CUTE environment (Section 6.6). This file is generated by the SIF decoder. For example, if we apply the UFO system to the input file `TSIF21.UFO`, the file `P.SIF` contains the following text:

```
Problem name: BRITGAS
The objective function uses      1 nonlinear group

There are    360 nonlinear equality constraints

There are    426 variables bounded only from below
There are     24 variables bounded from below and above
```

File `P.PAT` is a text file containing graphical expressions of sparsity patterns of the Hessian and the Jacobian matrices (Section 6.5).

5.11 Error messages

If we use the specification $MOUT>0 (basic screen output), then nonstandard terminations are indicated. The message consists of the three parts: the name of a critical subroutine, the number of a message, and an explanation text. For example, if the number of iterations is exceeded, we obtain the following message:

```
  0  NIT= 500  NFV=2545  NFG= 0  NDC=1556  NCG= 0  F= .122D+06  G= .112D+05
UYFUT1: (-2) MAXIMUM NUMBER OF ITERATIONS
```

Error messages are very useful especially in case the problem dimension is invalid. For example, if the number of nonzero elements in the Jacobian matrix is specified incorrectly, then we obtain the message:

```
  0  NIT=   0  NFV=   0  NFG= 0  NDC=   0  NCG= 0  F= .000D+00  G= .000D+00
UZLMIN: (78) LACK OF SPACE : MA TOO SMALL
ACTUAL VALUE:     298   -   DECLARED VALUE:     250
```

Here `UZLMIN` is the subroutine where the error was detected, 78 is the error number and `MA TOO SMALL` is the explanation. In this case, additional information (`ACTUAL VALUE` and `DECLARED VALUE`) is given.

The following table presents all UFO error messages (error numbers and explanations):

```
-1   -  MAXIMUM NUMBER OF FUNCTION EVALUATIONS : MFV TOO SMALL
-2   -  MAXIMUM NUMBER OF ITERATIONS : MIT TOO SMALL
-3   -  MAXIMUM NUMBER OF CYCLES : MIC TOO SMALL
 1   -  BAD DECOMPOSITION
 2   -  BAD INTERVAL IN THE OLC DIRECTION DETERMINATION
 3   -  MAXIMUM NUMBER OF STEPS IN THE OLC DIRECTION DETERMINATION
 4   -  BREAKDOWN IN THE ITERATIVE METHOD
 5   -  BREAKDOWN IN THE ITERATIVE METHOD
 6   -  MAXIMUM NUMBER OF REDUCTIONS
 7   -  NEGATIVE DIRECTIONAL DERIVATIVE
 8   -  BAD INTERVAL FOR INTERPOLATION
 9   -  BAD PREDICTION IN THE TRUST REGION METHOD
10   -  RESTART
11   -  FEASIBLE SOLUTION DOES NOT EXIST
12   -  BOUNDED SOLUTION DOES NOT EXIST
13   -  FEASIBLE SOLUTION DOES NOT EXIST
14   -  FEASIBLE TRUST REGION DOES NOT EXIST
15   -  INVALID SITUATION IN CONSTRAINT HANDLING
16   -  INVALID SITUATION IN CONSTRAINT HANDLING
17   -  LACK OF SPACE IN CONSTRAINT HANDLING : MMAX TOO SMALL
18   -  LACK OF SPACE IN CONSTRAINT HANDLING : MMAX TOO SMALL
19   -  BAD INPUT DATA
```

```
20   -  BAD INPUT DATA
21   -  UXSGFD: NAU IS DECLARED TOO SMALL
22   -  UXSGFD: NZ IS DECLARED TOO SMALL
23   -  UXSGFD: JACOBIAN MATRIX IS TOO UNSTABLE
24   -  UXSGFD: JACOBIAN MATRIX IS SINGULAR
25   -  UXSGFD: NZ IS TOO SMALL FOR THE FACTOR
26   -  UXSGFD: NAU IS TOO SMALL FOR THE FACTOR
27   -  UXSGFD: NZ IS TOO SMALL FOR DATA MANIPULATIONS AFTER FACTORIZATION
28   -  UXSGFD: COLUMN SCHEME FOR THE FACTOR IS NOT CREATED: LACK OF SPACE
29   -  BAD INPUT DATA
30   -  BAD INPUT DATA
31   -  UXSGUM: NAU IS DECLARED TOO SMALL
32   -  UXSGUM: JACOBIAN MATRIX IS SINGULAR
33   -  UXSGUM: FACTOR IS BADLY CONDITIONED
34   -  UXSGUM: LITTLE SPACE FOR L-UPDATES
35   -  UXSGUM: JACOBIAN MATRIX SINGULARITY IS FACED
36   -  UNBOUNDENESS IS FACED
37   -  UKLTS3: ROWS ARE NOT SPECIFIED
38   -  UKLTS3: COLUMNS ARE NOT SPECIFIED
39   -  UKLTS3: TYPE IS NOT SPECIFIED
40   -  UKLTS3: TYPC IS NOT DEFINED
41   -  LACK OF SPACE FOR THE CHOLESKI FACTOR : MMAX TOO SMALL
42   -  LACK OF SPACE FOR A SYMBOLIC FACTORIZATION : MMAX TOO SMALL
43   -  LACK OF SPACE FOR THE FILL-IN
44   -  LACK OF SPACE FOR NUMERICAL DIFFERENTIATION : M TOO SMALL
45   -  STRUCTURAL SINGULARITY DURING INCOMPLETE LU FACTORIZATION
46   -  INVALID STRUCTURE FOR INCOMPLETE LU FACTORIZATION
47   -  LACK OF SPACE IN NUMERICAL DIFFERENTIATION : NVAR TOO SMALL
48   -  LACK OF SPACE IN THE INCOMPLETE DECOMPOSITION : MMAX TOO SMALL
49   -  LACK OF SPACE IN THE SCHUR COMPLEMENT : MMAX TOO SMALL
50   -  LACK OF SPACE FOR THE FACTOR
51   -  INSUFFICIENT STORAGE FOR NONZERO SUBSCRIPTS
52   -  LACK OF SPACE IN THE FRONTAL SCHEME
53   -  ERROR IN THE FRONTAL SCHEME
54   -  LACK OF SPACE IN THE FRONTAL SCHEME
55   -  ERROR IN THE FRONTAL SCHEME
56   -  LACK OF SPACE IN THE INTEGER FIELD
57   -  LACK OF SPACE IN THE REAL FIELD
58   -  ZERO INDEX
59   -  DIMENSION ERROR
60   -  LACK OF SPACE IN THE WORKING FIELD
61   -  INVALID MATRIX ORDER
62   -  NUMBER OF NONZEROS SMALLER THAN ZERO
63   -  INVALID ENTRIES IN THE INPUT MATRIX
64   -  INCONSISTENT MEMORY
65   -  LACK OF SPACE IN THE INTEGER FIELD : MMAX TOO SMALL
66   -  LACK OF SPACE IN THE REAL FIELD : MMAX TOO SMALL
67   -  INVALID LU FACTORS
68   -  MAXIMUM INTEGER TOO SMALL
69   -  INVALID INPUTS
70   -  ZERO PIVOT WHEN DEFINITENESS IS DECLARED
71   -  CHANGE IN SIGN OF PIVOT ENCOUNTERED
```

```
 72  -  SINGULARITY DETECTED
 73  -  NONZERO ELEMENT IGNORED
 74  -  PIVOT HAS DIFFERENT SIGN FROM THE PREVIOUS ONE
 75  -  LACK OF SPACE : M TOO SMALL
 76  -  LACK OF SPACE : MAH TOO SMALL
 77  -  LACK OF SPACE : MCH TOO SMALL
 78  -  LACK OF SPACE : MA TOO SMALL
 79  -  LACK OF SPACE : MC TOO SMALL
 80  -  LACK OF SPACE : NF TOO SMALL
 81  -  LACK OF SPACE : NA TOO SMALL
 82  -  LACK OF SPACE : NC TOO SMALL
 83  -  SIMPLE BOUNDS ARE NOT PERMITTED
 84  -  INEQUALITY CONSTRAINTS ARE NOT PERMITTED
 85  -  TOO MANY DENSE ROWS : ND TOO SMALL
 86  -  LACK OF SPACE : MHA TOO SMALL
 87  -  LACK OF SPACE : MHC TOO SMALL
 88  -  LINEAR DEPENDENCE OF ACTIVE CONSTRAINTS
 89  -  INFEASIBLE SOLUTION
 90  -  MAXIMUM NUMBER OF SIMPLEX ITERATIONS : MIS TOO SMALL
 91  -  DIFFERENTIAL EQUATION IS UNSTABLE
 92  -  MAXIMUM NUMBER OF INTEGRATION STEPS EXCEEDED
 93  -  TOO SMALL INTEGRATION STEP
 94  -  DIFFERENTIAL EQUATION IS STIFF
 95  -  SINGULAR JACOBIAN IN IMPLICIT INTEGRATION METHOD
 96  -  LACK OF SPACE IN DIFFERENTIAL EQUATION SOLVER
 97  -  LACK OF SPACE : MMAX TOO SMALL
 98  -  MAXIMUM NUMBER OF MINOR CYCLES : MIQ TOO SMALL
 99  -  MAXIMUM NUMBER OF MAJOR CYCLES : MAQ TOO SMALL
100  -  LACK OF SPACE : MF TOO SMALL
101  -  LARGE LAGRANGIAN FUNCTION RETURN
102  -  INVALID TERMINATION: EPS9 TOO SMALL
103  -  FEASIBILITY RESTORATION PHASE IS EXPECTED
104  -  LACK OF SPACE FOR THE HESSIAN MATRIX : MMAX TOO SMALL
105  -  LACK OF SPACE FOR THE PRECONDITIONER : MMAX TOO SMALL
106  -  EQUATION SOLVER FAILS
107  -  EIGENPROBLEM WAS NOT SOLVED
108  -  ARPACK: INPUT ERROR
109  -  ARPACK: NO EIGENVALUE CONVERGED
110  -  LACK OF SPACE : NCC TOO SMALL
111  -  TWO-SIDE BOX CONSTRAINT OCCURED
112  -  TWO-SIDE GENERAL CONSTRAINT OCCURED
113  -  EQUALITY IN COMPLEMENTARITY CONSTRAINT OCCURED
114  -  COUPLED COMPLEMENTARITY CONSTRAINT IS MISSING
115  -  WRONG INDEX IN COMPLEMENTARITY BOX CONSTRAIN
116  -  WRONG INDEX IN COMPLEMENTARITY GENERAL CONSTRAINT
117  -  DUPLICATE INDEX IN COMPLEMENTARITY BOX CONSTRAINT
118  -  DUPLICATE INDEX IN COMPLEMENTARITY GENERAL CONSTRAINT
119  -  EQUALITY CONSTRAINTS ARE NOT PERMITTED
```

6 Special tools of the UFO system

The UFO system allows to use automatic differentiation and contains special tools that facilitate the user's activity. There are special tools for checking problem descriptions, collections of problems for testing optimization methods, subroutines for printing sparsity patterns and modules realizing the interface to the CUTE environment.

6.1 Automatic differentiation

If derivatives of the model function FF are not given, i.e., if only macrovariable $FMODELF is defined, then either numerical or automatic differentiation is used if is necessary. The choice of the kind of differentiation is specified by the macrovariable $IADF:

$IADF=0 - Derivatives of the model function are computed by numerical differentiation.

$IADF=1 - The first order derivatives of the model function are computed by using the reverse mode of automatic differentiation. New macrovariable $FGMODELF, which defines the value FF and the gradient GF of the model function is created and the original macrovariable $FMODELF is canceled. The second order derivatives are computed numerically, if they are required.

$IADF=2 - The first order derivatives of the model function are computed by using the reverse mode of automatic differentiation. New macrovariable $FGMODELF, which defines the value FF and the gradient GF of the model function is created. The reverse mode is followed by the forward mode for computation of the second order derivatives. New macrovariable $HMODELF, which defines the Hessian matrix HF of the model function is created. Finally, the original macrovariable $FMODELF is canceled.

Automatic differentiation is realized in the first phase by the UFO control language preprocessor. All variables contained in macrovariable $FMODELF are redefined and all expressions are transformed in the way that also the sequence of elementary operations and their parameters are stored. The list of elementary operations is used in the reverse mode of automatic differentiation by the subroutine `RVRSWP` (if $IADF=1) or `RVRSWPH` (if $IADF=2). The subroutines which realize elementary operations are included to the UFO source program.

If the automatic differentiation is chosen ($IADF>0), there are limitations concerning Fortran 77 expressions in the macrovariable $FMODELF:

- If an expression contains array $FLOAT W(100), say, with elements depending on the vector of variables, then the user has to declare corresponding array INTEGER IAD_W(100) in the input file `*.UFO` (before statements $GLOBAL or $STANDARD).
- Calls of functions and subroutines (with exceptions of intrinsic Fortran 77 functions in the generic form, i.e., `SQRT`, `EXP`, `LOG`, `LOG10`, `SIN`, `COS`, `TAN`, `ASIN`, `ACOS`, `ATAN`, `SINH`, `COSH`, `TANH`) are not permitted.
- Names of variables cannot contain digits.
- Statements containing scalar variables, e.g., `W = W * X(I)`, cannot be used in a cycle. It is necessary to declare array `W(*)` and write `W(I+1) = W(I) * X(I)` instead.
- Blanks cannot be used in `WRITE` statement. Arguments of functions are not replaced by transformed variables in `WRITE` statement.
- Blanks cannot be used in `IF` and `ELSE IF` statements. Only comparisons of numbers and variables are permitted (expressions are forbidden). Arguments of functions and indices of arrays are not replaced by transformed variables in `IF` and `ELSE IF` statements.
- Computed `GO TO` statement cannot be used.

Automatic differentiation can also be used for computation of derivatives of approximating functions and constraint functions described by macrovariables $FMODELA and $FMODELC, respectively. In these

cases, the kind of differentiation is specified by macrovariables $IADA and $IADC. The meaning of these macrovariables and the limitations concerning Fortran 77 expressions in macrovariables $FMODELA and $FMODELC are the same as those for macrovariables $IADF and $FMODELF described above.

Automatic differentiation can be used for dense problems at present, i.e., it does not work if $JACA='S', $JACC='S', $HESF='S', $HESF='B'.

6.2 Checking external subroutines

The values, gradients, Hessian matrices of the model function or the approximating functions or the constraint functions are specified by using the macrovariables $FMODELF, $GMODELF, $HMODELF or $FMODELA, $GMODELA, $HMODELA or $FMODELC, $GMODELC, $HMODELC, respectively. Sometimes the correctness of these models needs to be checked up. If this is the case, then both the analytical and the numerical differentiation can be compared. The checking of optimization problems can be specified by using the macrovariable $TEST. If $TEST='N', no checking is performed. If $TEST='Y', both the analytical and the numerical differentiation are executed before optimization is started (at the initial starting point) and the derivatives obtained are printed. Only the derivatives that are analytically specified (the first, the second) are checked. If $TEST='A', the checking is performed after the optimization is finished (at the final optimum point). Finally, if $TEST='O', only checking is performed and optimization is not started. The output of checking an optimization problem has the following form:

```
STANDARD TEST OF EXTERNAL SUBROUTINES
-------------------------------------
PROBLEM NO    1

PROBLEM
-------
NF = 2                    KDF =  2  KSF = 1  KCF = 2  NORMF = 0
NA = 0  NAL = 0  MAL = 0  KDA = -1  KSA = 0  KCA = 0  NORMA = 0
NC = 3  NCL = 0  MCL = 0  KDC =  1  KSC = 0  KCC = 2  NORMC = 0

PARAMETERS
----------
X    =   -.2000000000D+01    .1000000000D+01

DERIVATIVES
-----------
FF A =    .9090000000D+03
GF N =   -.2405999822D+04   -.6000004263D+03
GF A =   -.2406000000D+04   -.6000000000D+03
HF N =    .4402000148D+04    .8000000070D+03    .2000000002D+03
HF A =    .4402000000D+04    .8000000000D+03    .2000000000D+03

FC A =   -.1000000000D+01
GC N =    .1000000000D+01    .2000000032D+01
GC A =    .1000000000D+01    .2000000000D+01

FC A =    .5000000000D+01
GC N =   -.4000000070D+01    .9999999930D+00
GC A =   -.4000000000D+01    .1000000000D+01

FC A =    .5000000000D+01
GC N =   -.4000000070D+01    .2000000042D+01
GC A =   -.4000000000D+01    .2000000000D+01
```

Here the letter 'N' indicates a numerical differentiation and the letter 'A' indicates an analytical differentiation.

6.3 Testing optimization methods

The UFO system contains a great number of subroutines (collections of test problems) which serve for testing optimization methods. All of these subroutines begin with the letter 'E' (external). The input subroutines (containing starting points, types of constraints, sparsity patterns, etc.) have the second letter 'I', the third letter 'U' or 'L' or 'N' for unconstrained or linearly constrained or nonlinearly constrained problems, respectively, and the fourth letter 'D' or 'S' or 'B' for dense or sparse or partially separable problems, respectively. The model specification subroutines have the second letter 'F' or 'A' or 'C' or 'E' or 'Y' for model functions or approximating functions or constraint functions or state functions or initial functions, respectively, the third letter 'F' or 'G' or 'B' or 'H' for values or gradients or values together with gradients or Hessian matrices, respectively, and the fourth letter 'U' or 'D' or 'S' or 'B' for universal or dense or sparse or partitioned problems, respectively. The last two digits specify individual test problem collections. When we want to carry out a test of the method selected, we define macrovariable $COLLECTION and set $NEXT=number_of_test_problems. The following specifications can be used:

`$COLLECTION='N'` - test is suppressed.
`$COLLECTION='Y'` - test is performed and basic output is printed.
`$COLLECTION='E'` - test is performed and extended output is printed.
`$COLLECTION='P'` - test is performed, basic output is printed and data for performance profiles are prepared.

The default value is $COLLECTION='N'.

Tests corresponding to individual test problems collections are realized by using the following batch input files (asterisks denote possible additional letters):

`TEST01*.UFO` - tests for unconstrained optimization (25 dense problems from [53], [190]). External subroutines `EIUD01`, `EFFU01`, `EFGU01`, `EFHD01` are used.

`TEST02*.UFO` - tests for the sum of squares minimization (30 dense problems from [257]). External subroutines `EIUD02`, `EAFU02`, `EAGU02`, `EAHD02` are used.

`TEST03*.UFO` - tests for linearly constrained optimization (15 dense problems from [151]). External subroutines `EILD03`, `EFFU03`, `EFGU03` are used.

`TEST04*.UFO` - tests for medium-size linear programming (6 dense problems). External subroutine `EILD04` is used.

`TEST05*.UFO` - tests for medium-size quadratic programming (5 dense problems). External subroutine `EILD05` is used.

`TEST06*.UFO` - tests for minimax (25 dense problems from [233]). External subroutines `EIUD06`, `EAFU06`, `EAGU06`, `EAHD06` are used.

`TEST07*.UFO` - tests for inequality constrained nonlinear programming (34 dense problems from [151]). External subroutines `EIND07`, `EFFU07`, `EFGU07`, `ECFU07`, `ECGU07` are used.

`TEST08*.UFO` - tests for equality constrained nonlinear programming (31 dense problems from [151]). External subroutines `EIND08`, `EFFU08`, `EFGU08`, `ECFU08`, `ECGU08` are used.

`TEST09*.UFO` - tests for unconstrained global optimization (13 problems from [384]). External subroutines `EIUD09`, `EFFU09`, `EFGU09` are used.

`TEST10*.UFO` - tests for unconstrained optimization (15 sparse problems from [225]). External subroutines `EIUS10`, `EFFU10`, `EFGU10`, `EFHS10` are used.

`TEST11*.UFO` - tests for unconstrained optimization (58 sparse problems from the CUTE collection [25], see also [218]). External subroutines `EIUD11`, `EIUX11`, `EFBU11`, `EFFU11`, `EFGU11` are used.

TEST12*.UFO - tests for unconstrained optimization (73 sparse problems from [7]). External subroutines EIUD12, EFBU12, EFFU12, EFGU12 are used.

TEST13*.UFO - tests for linearly constrained optimization (6 sparse problems). External subroutines EILS13, EINS13, EFFU13, EFGU13 are used.

TEST14*.UFO - tests for the sum of functions minimization (22 sparse problems from [225]). External subroutines EIUD14, EIUS14, EIUB14, EFFU14, EFGU14, EAFU14, EAGU14, EAHB14, EBFU14, EBGU14 are used.

TEST15*.UFO - tests for the sum of squares minimization (24 sparse problems from [225]). External subroutines EIUD15, EIUB15, EAFU15, EAGU15, EABU15, EAHB15, EBFU15, EBGU15 are used.

TEST16*.UFO - tests for nonlinear equations solutions (32 dense problems). External subroutines EIUD16, EAFU16, EAGU16 are used.

TEST17*.UFO - tests for nonlinear equations solutions (30 dense problems). External subroutines EIUD17, EAFU17, EAGU17 are used.

TEST18*.UFO - tests for nonlinear equations (44 sparse problems from [225]). External subroutines EIUD18, EIUB18, EAFU18, EAGU18 are used.

TEST19*.UFO - tests for nonsmooth unconstrained optimization (25 dense problems from [233]). External subroutines EIUD19, EFFU19, EFGU19, EFHD19 are used.

TEST20*.UFO - tests for equality constrained sparse nonlinear programming (18 sparse problems from [225]). External subroutines EIUD20, EIUB20, EIUS20, EIND20, EINS20, EFFU20, EFGU20, EAFU20, EAGU20, ECFU20, ECGU20 are used.

TEST21*.UFO - tests for inequality constrained sparse nonlinear programming (18 sparse problems from [225]). External subroutines EIUD20, EIUB20, EIUS20, EIND20, EINS20, EFFU20, EFGU20, EAFU20, EAGU20, ECFU20, ECGU20 are used.

TEST22*.UFO - tests for linearly constrained minimax optimization (20 dense problems from [233]). External subroutines EIUD22, EAFU22, EAGU22, EAHD22 are used.

TEST23*.UFO - extended tests for unconstrained optimization (74 dense problems from [225]). External subroutines EIUD23, EFFU23, EFGU23 are used.

TEST24*.UFO - extended tests for the sum of squares minimization (80 dense problems from [53], [190], [225], [257]). External subroutines EIUD24, EAFU24, EAGU24 are used.

TEST25*.UFO - extended tests for the sum of functions minimization (82 sparse problems from [216]). External subroutines EIUD25, EIUB25, EIUS25, EFBU25, EFFU25, EFGU25, EAFU25, EAGU25 are used.

TEST26*.UFO - extended tests for the sum of squares minimization (60 sparse problems from [225]). External subroutines EIUB26, EAFU26, EAGU26 are used.

TEST27*.UFO - extended tests for the sum of squares minimization (82 dense problems from [225]). External subroutines EIUD27, EAFU27, EAGU27 are used.

TEST28*.UFO - extended tests for unconstrained optimization (92 dense problems from [53], [190], [225], [257]). External subroutines EIUD28, EIUN28, EFFU28, EFGU28, EFBU28 are used.

TEST29*.UFO - tests for nonsmooth unconstrained optimization (31 dense problems). External subroutines EIUD29, EFFU29, EFGU29, EFBU29 are used.

TEST30*.UFO - tests for optimization of dynamical systems (4 dense problems). External subroutines EIUD30, EEFU30, EEGU30, EYFU30, EYGU30 are used.

TEST31*.UFO - tests for nonstiff differential equations (6 dense problems). External subroutines EIUD31, EEFU31 are used.

TEST32*.UFO - tests for large-scale linear programming (18 sparse problems). External subroutine EILS32, EISS32 are used.

TEST33*.UFO - tests for large-scale quadratic programming (11 sparse problems). External subroutines EILS33, EIQS33 are used.

TEST34*.UFO	- tests for large-scale linear programming (18 sparse problems). External subroutine EINS20 is used.
TEST35*.UFO	- tests for large-scale quadratic programming (30 sparse problems). External subroutine EIQS35 is used.
TEST36*.UFO	- extended tests for nonlinear equations solutions (94 dense problems). External subroutines EIUD36, EAFU36, EAGU36 are used.
TEST37*.UFO	- extended tests for nonlinear equations solutions (64 dense problems). External subroutines EIUD37, EAFU37, EAGU37 are used.
TEST38*.UFO	- tests for stiff differential equations (4 dense problems). External subroutines EIUD38, EEFU38, EEDU38 are used.

In these batch input files, all necessary macrovariables are defined and the external subroutines are called. The external subroutines with the last two digits 01, ..., 38 are briefly described in the text files E01.TXT, ..., E38.TXT. The external subroutines with the last two digits 01, ..., 22 and 29, ..., 35 contain original test problems. The remaining external subroutines are their various combinations.

To demonstrate the use of the test input file we perform a test of the sum of squares minimization by using a hybrid method realized as a trust region method. The test input file TEST02.UFO has the following form:

```
$REM : model specifications
$SET(INPUT)
  CALL EIUD02(NF,NA,NAL,X,FMIN,XMAX,NEXT,IEXT,IERR)
  IF(NEXT.EQ.10) XMAX=1.0$P 1
  IF(IERR.NE. 0) GO TO $$ENDTEST
$ENDSET
$SET(FMODELA)
  CALL EAFU02(NF,KA,X,FA,NEXT)
$ENDSET
$NF=12
$NA=400
$MODEL='AQ'
$REM : method specifications
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=7
$UPDATE='F'
$REM : precision specifications
$TOLX='1.0$P-16'
$TOLF='1.0$P-14'
$TOLB='1.0$P-16'
$TOLG='1.0$P-6'
$REM : print specifications
$MOUT=1
$REM : the cycle for testing 30 test problems is generated
$COLLECTION='YES'
$NEXT=30
$REM : the batch mode is used
$BATCH
$REM : the standard form of the source program is used
$STANDARD
```

The result (screen output) obtained has the following form (each row corresponds to one test problem and

the last row is the summary):

```
 CLASS = GN - GM7   UPDATE = F   MODEL = AQ   HESF = D   NF =       2
   1 NIT=    12 NFV=    41 NFG=     0  FV BOUND  F= 0.2465190329E-31 G=0.222D-15
   2 NIT=    20 NFV=    69 NFG=     0  GRAD TOL  F=  24.49212684     G=0.667D-06
   3 NIT=    33 NFV=   102 NFG=     0  FV BOUND  F= 0.2035629458E-22 G=0.581D-06
   4 NIT=    14 NFV=    47 NFG=     0  FV BOUND  F=  0.000000000     G=0.000D+00
   5 NIT=     6 NFV=    21 NFG=     0  GRAD TOL  F= 0.1422753549E-15 G=0.807D-07
   6 NIT=    11 NFV=    41 NFG=     0  GRAD TOL  F=  62.18109118     G=0.927D-06
   7 NIT=     7 NFV=    32 NFG=     0  FV BOUND  F= 0.2748725380E-26 G=0.728D-12
   8 NIT=     5 NFV=    24 NFG=     0  GRAD TOL  F= 0.4107438653E-02 G=0.293D-08
   9 NIT=     1 NFV=     8 NFG=     0  GRAD TOL  F= 0.5639663969E-08 G=0.177D-07
  10 NIT=   235 NFV=   954 NFG=     0  STEP TOL  F=  43.97292759     G=0.178D-03
  11 NIT=    80 NFV=   329 NFG=     0  FV BOUND  F= 0.7813056212E-24 G=0.350D-08
  12 NIT=    12 NFV=    53 NFG=     0  FV BOUND  F= 0.3624914786E-20 G=0.107D-09
  13 NIT=    10 NFV=    55 NFG=     0  GRAD TOL  F= 0.1686647891E-09 G=0.247D-06
  14 NIT=    40 NFV=   213 NFG=     0  GRAD TOL  F= 0.7422398347E-15 G=0.538D-06
  15 NIT=    11 NFV=    62 NFG=     0  GRAD TOL  F= 0.1537528024E-03 G=0.511D-06
  16 NIT=    24 NFV=   134 NFG=     0  FV   TOL  F=  42911.10081     G=0.108D-03
  17 NIT=    22 NFV=   139 NFG=     0  GRAD TOL  F= 0.2732447349E-04 G=0.127D-06
  18 NIT=    17 NFV=   128 NFG=     0  GRAD TOL  F= 0.1023149745E-15 G=0.273D-07
  19 NIT=    13 NFV=   169 NFG=     0  GRAD TOL  F= 0.2192388901E-01 G=0.379D-07
  20 NIT=     9 NFV=   130 NFG=     0  GRAD TOL  F= 0.2361205389E-09 G=0.138D-06
  21 NIT=    12 NFV=   171 NFG=     0  FV BOUND  F= 0.1479114197E-30 G=0.222D-15
  22 NIT=    10 NFV=   143 NFG=     0  GRAD TOL  F= 0.5059943643E-09 G=0.247D-06
  23 NIT=    19 NFV=   264 NFG=     0  GRAD TOL  F= 0.4393003951E-04 G=0.361D-06
  24 NIT=    17 NFV=   244 NFG=     0  GRAD TOL  F= 0.3081274513E-03 G=0.820D-06
  25 NIT=    10 NFV=   143 NFG=     0  FV BOUND  F= 0.1248755103E-25 G=0.190D-11
  26 NIT=     9 NFV=   133 NFG=     0  GRAD TOL  F= 0.1376154725E-06 G=0.491D-07
  27 NIT=     6 NFV=    91 NFG=     0  FV BOUND  F= 0.9458284017E-18 G=0.142D-08
  28 NIT=     7 NFV=   104 NFG=     0  GRAD TOL  F= 0.2107337939E-10 G=0.379D-06
  29 NIT=     2 NFV=    39 NFG=     0  GRAD TOL  F= 0.7987932563E-13 G=0.203D-06
  30 NIT=     5 NFV=    78 NFG=     0  FV BOUND  F= 0.2322757795E-26 G=0.181D-12
 TOTAL   NIT=      679   NFV=    4161   NFG=       0   NDC=    1549 *    29
         NCG=        0   NLS=       0   NSC=       0   NUP=     670
         NRS=        0   NMX=       0   NAD=       0   NRM=       0
```

If $COLLECTION='Y' or $COLLECTION='E', we can define values of various parameters depending on the value of NEXT, where $1 \leq$ NEXT $\leq$ $NEXT is the variable of the cycle. This can be realized using macrovariable $INITCYCLE by setting

```
$SET(INITCYCLE)
  Here the Fortran statements are written defining values of
  method parameters using the current value of variable NEXT.
$ENDSET
```

To demonstrate the use of the macrovariable $INITCYCLE, we test the performance of a limited memory variable metric method for five values of the parameter $ETA3 using the CUTE problem BDQRTIC (section 6.6). The batch input file `CYCLE.UFO` has the following form:

```
$SIF='BDQRTIC'
$CLASS='CD'
$TYPE='L'
```

```
$DECOMP='V'
$NUMBER=5
$COLLECTION='Y'
$NEXT=5
$SET(INITCYCLE)
  ETA3=-$DBLE(NEXT)*8.0D-2
$ENDSET
$SET(OUTPUT)
  WRITE(2,*)' ETA3 =   ', ETA3
$ENDSET
$MOUT=1
$BATCH
$STANDARD
```

The result (screen output) obtained has the following form:

```
 CLASS = CD - LV5   UPDATE = N   MODEL = FF   HESF = N     NF =   5000
   1 NIT=  308 NFV=  458 NFG=  458  MAX  INT  F=  20006.25688     G=0.173D-03
  ETA3 =    -8.000000000000000E-002
 TIME= 0:00:00.82
   2 NIT=  183 NFV=  269 NFG=  269  MAX  INT  F=  20006.25688     G=0.537D-03
  ETA3 =    -0.160000000000000
 TIME= 0:00:00.51
   3 NIT=  263 NFV=  360 NFG=  360  FV   TOL  F=  20006.25688     G=0.376D-03
  ETA3 =    -0.240000000000000
 TIME= 0:00:00.67
   4 NIT=  252 NFV=  327 NFG=  327  FV   TOL  F=  20006.25688     G=0.247D-02
  ETA3 =    -0.320000000000000
 TIME= 0:00:00.62
   5 NIT=  258 NFV=  351 NFG=  351  FV   TOL  F=  20006.25688     G=0.486D-02
  ETA3 =    -0.400000000000000
 TIME= 0:00:00.65
 TOTAL   NIT=    1264   NFV=    1765   NFG=    1765   NDC=       0 *     5
         NCG=       0   NLS=     490   NSC=       0   NUP=       0
         NRS=       6   NMX=       0   NAD=       0   NRM=       0
 TIME= 0:00:03.29
```

6.4 Computation of performance profiles

Performance profiles introduced in [97] are a very suitable way for comparison of optimization method, since they take into account results corresponding to individual problems. The performance profile $\rho_m(\tau)$ is defined by the formula

$$\rho_m(\tau) = \frac{\text{number of problems where } \log_2(\tau_{p,m}) \leq \tau}{\text{total number of problems}}$$

with $0 \leq \tau \leq \overline{\tau}$, where $\tau_{p,m}$ is the performance ratio of the number of function evaluations (or the time) required to solve problem p by method m to the lowest number of function evaluations (or the time) required to solve problem p. The ratio $\tau_{p,m}$ is set to infinity (or some large number) if method m fails to solve problem p. The value of $\rho_m(\tau)$ at $\tau = 0$ gives the percentage of test problems for which the method m is the best and the value for τ large enough is the percentage of test problems that method m can solve. The relative efficiency and reliability of each method can be directly seen from the performance profiles: the higher is the particular curve the better is the corresponding method.

Data for performance profiles are prepared if we set $COLLECTION='P'. In this case, values of NIT (numbers of iterations), NFV (numbers of function evaluations), NFG (numbers of gradient evaluations), NCGR (numbers of inner iterations), TIME (CPU time in hundredths of seconds) are saved for individual problems in file P.PER. We can specify a name of the method tested by the statement $NAME='name_of_the_method'. Before tests are carried out, the old file P.PER is automatically deleted. This deletion can be suppressed by setting $APPEND='Y'. Then the new results of individual tests are appended to the old file P.PER. The file P.PER is a source for the computation of performance profiles contained in the file P.PRO. This file can be generated immediately or created by special procedures. The corresponding choice is specified by the macrovariable $PROFILE, which can have the following values:

$PROFILE='NO' - performance profiles (file P.PRO) are not generated immediately.
$PROFILE='NIT' - performance profiles for NIT are generated immediately.
$PROFILE='NFV' - performance profiles for NFV are generated immediately.
$PROFILE='NFG' - performance profiles for NFG are generated immediately.
$PROFILE='NCGR' - performance profiles for NCGR are generated immediately.
$PROFILE='TIME' - performance profiles for TIME are generated immediately.

The default value is $PROFILE='NO'. If macrovariable $PROFILE has one of the values NIT, NFV, NFG, NCGR, TIME, performance profiles for this quantity are computed immediately and we can specify the upper bound $\overline{\tau}$ of the parameter τ by the statement $BOUND='upper_bound' (e.g., $BOUND='3.0D0'). The default value is $BOUND='2.0D0' and the maximum value is $BOUND='5.0D0' (this value is used if we set $BOUND>'5.0D0'). The resulting profiles are saved in file P.PRO. More advantageous possibility is to set $PROFILE='NO' and compute performance profiles by special procedures from data saved in file P.PER (prepared by the UFO system if $COLLECTION='P'). In this case, we use a DOS command line (in the UFO directory) and type

PROFILE quantity upper_bound,

where parameter quantity is one of the values NIT, NFV, NFG, NCGR, TIME and upper_bound is a real number (upper bound for τ). Then, procedure PROFILE is called, which generates file P.PRO from data contained in file P.PER.

File P.PER can be also used for preparing the graphical representations of performance profiles. File P.TEX, containing data for the environment picture in the typesetting language LATEX, is generated if we type

PROFTEX quantity upper_bound size,

where parameters quantity and upper_bound have the same meaning as in the procedure PROFILE and size is either 1 (one picture per a page width) or 2 (two pictures per a page width). Then procedure PROFTEX is called to generate files P.PRO and P.tex from data contained in file P.PER. Furthermore, file P.m, containing data for the MATLAB plotter, is generated if we type

PROFMAT quantity upper_bound,

where parameters quantity and upper_bound have the same meaning as in the procedure PROFILE. Then procedure PROFMAT is called, to generate files P.PRO and P.m from data contained in P.PER. Subsequently, calling P in the MATLAB environment, the graphic file P.eps is created. Finally, file P.sci, containing data for the SCILAB plotter, is generated if we type

PROFSCI quantity upper_bound,

where parameters quantity and upper_bound have the same meaning as in the procedure PROFILE. Then procedure PROFSCI is called, to generate files P.PRO and P.sci from data contained in P.PER. Subsequently, calling exec P.sci in the SCILAB environment, the graphic file P.eps is created. Procedures PROFILE,

PROFTEX, PROFMAT and PROFSCI, written in Fortran language, are parts of the UFO system.

The following example demonstrates the computation of performance profiles for three limited-memory variable metric methods: LV1 - M1 [272], LV6 - M1 [351] and LV8 - M1 [352]. The input template P.UFO has the form:

```
$REM +-------------------------------------------------------------+
$REM +             MODEL DESCRIPTION                               +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  INITS=2
  IF (NEXT.EQ. 2) INITS=1
  IF (NEXT.EQ. 3) INITS=1
  IF (NEXT.EQ. 4) INITS=1
  IF (NEXT.EQ.12) INITS=1
  IF (NEXT.EQ.30) INITS=1
  IF (NEXT.EQ.35) INITS=1
  IF (NEXT.EQ.46) INITS=1
  IF (NEXT.EQ.56) INITS=1
  IF (NEXT.EQ.65) INITS=1
  IF (NEXT.EQ.75) INITS=1
  CALL EIUD25(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
  IF (NEXT.EQ.57) GO TO $$ENDTEST
  IF (NEXT.EQ.58) GO TO $$ENDTEST
  IF (NEXT.EQ.60) GO TO $$ENDTEST
  IF (NEXT.EQ.61) GO TO $$ENDTEST
  IF (NEXT.EQ.67) GO TO $$ENDTEST
  IF (NEXT.EQ.68) GO TO $$ENDTEST
  IF (NEXT.EQ.69) GO TO $$ENDTEST
  IF (NEXT.EQ.70) GO TO $$ENDTEST
  IF (NEXT.EQ.79) GO TO $$ENDTEST
  IF (IERR.NE. 0) GO TO $$ENDTEST
$ENDSET
$SET(FGMODELF)
  CALL EFBU25(NF,X,FF,GF,NEXT)
$ENDSET
$MF=5
$NF=1000
$COLLECTION='P'; $REM : computation of performance profiles
$NEXT=82; $REM : number of test problems
$PROFILE='NO'; $REM : performance profiles are not computed immediately
$BOUND='2.0$P 0'; $REM : upper bound for performance ratio
$DELETION='Y'; $REM : preliminary deletion of P.PER

$REM +-------------------------------------------------------------+
$REM +             START OF THE PROGRAM GENERATION                 +
$REM +-------------------------------------------------------------+
$BATCH;  $REM : the batch mode
$GLOBAL; $REM : global declarations
```

```
$REM +---------------------------------------------------------------+
$REM +             THE FIRST METHOD                                  +
$REM +---------------------------------------------------------------+
$REM : parameters of the first method
$CLASS='CD'
$TYPE='L'
$DECOMP='V'
$NUMBER=1
$MIT=20000
$MFV=20000
$MFG=20000
$SET(CONST)
  IF (NEXT.EQ.40) XMAX=8.0$P 0
  IF (NEXT.EQ.45) XMAX=5.0$P 0
  IF (NEXT.EQ.48) XMAX=2.8$P 0
  IF (NEXT.EQ.50) XMAX=1.0$P 1
  IF (NEXT.EQ.51) XMAX=7.5$P 0
  IF (NEXT.EQ.52) XMAX=2.3$P 0
$ENDSET
$NAME='LV1 - M1'; $REM : name of the first method
$INITIATION; $REM : model specification and generation
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation
$RUNERASE; $REM : clearing parameters of the first run

$REM +---------------------------------------------------------------+
$REM +             THE SECOND METHOD                                 +
$REM +---------------------------------------------------------------+
$REM : parameters of the third method
$CLASS='CD'
$TYPE='L'
$DECOMP='V'
$NUMBER=6
$MET3=1
$MIT=20000
$MFV=20000
$MFG=20000
$SET(CONST)
  IF (NEXT.EQ.40) XMAX=8.0$P 0
  IF (NEXT.EQ.45) XMAX=1.2$P 0
  IF (NEXT.EQ.48) XMAX=9.0$P 0
  IF (NEXT.EQ.51) XMAX=1.0$P 0
  IF (NEXT.EQ.52) XMAX=2.3$P 0
$ENDSET
$NAME='LV6 - M1'; $REM : name of the third method
$INITIATION; $REM : run specification
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation
$RUNERASE; $REM : clearing parameters of the second run
```

```
$REM +-------------------------------------------------------------+
$REM +             THE THIRD METHOD                                +
$REM +-------------------------------------------------------------+
$REM : parameters of the third method
$CLASS='CD'
$TYPE='L'
$DECOMP='V'
$NUMBER=8
$MIT=20000
$MFV=20000
$MFG=20000
$SET(CONST)
  IF (NEXT.EQ.40) XMAX=1.0$P 1
  IF (NEXT.EQ.45) XMAX=2.0$P 0
  IF (NEXT.EQ.48) XMAX=6.0$P 0
  IF (NEXT.EQ.51) XMAX=1.0$P 0
  IF (NEXT.EQ.52) XMAX=3.4$P 0
$ENDSET
$NAME='LV8 - M1'; $REM : name of the third method
$INITIATION; $REM : run specification
$INPUT;  $REM : the user input
$METHOD; $REM : method specification and generation

$END; $REM : end of the program generation

$REM CALL PROFILE NTM 3.0D0
```

Here $COLLECTION='P' specifies computation of performance profiles, $NEXT=82 determines the number of test problems and $PROFILE='NO' specifies that performance profiles (file `P.PRO`) is not generated immediately. Other specifications are explained in Section 4. Using the above template, the UFO system creates file `P.PER`. Each row of this file contain (for a given method and a given test problem) five integer numbers `NIT`, `NFV`, `NFG`, `NCGR`, `TIME` written in the format '(20X,5I10)'. The file `P.PRO` generated using procedures `PROFILE` (or `PROFTEX`, `PROFMAT`, `PROFSCI`) has the form:

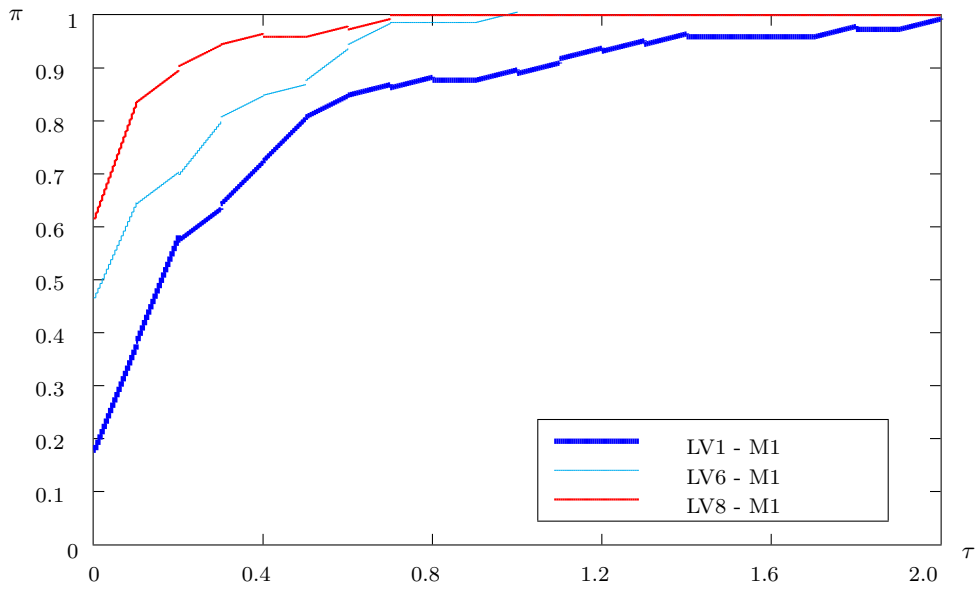
```
PERFORMANCE PROFILES:   NFV    BOUND =    2.0000   STEP =    0.1000   NMETH =  3

METHOD     1   LV1 - M1     MEAN VALUE =    0.8206   EFFICIENCY =    0.1764

    0.1781    0.3836    0.5753    0.6438    0.7260    0.8082    0.8493    0.8630
    0.8767    0.8767    0.8904    0.9178    0.9315    0.9452    0.9589    0.9589
    0.9589    0.9589    0.9726    0.9726    0.9863

METHOD     2   LV6 - M1     MEAN VALUE =    0.9165   EFFICIENCY =    0.4505

    0.4658    0.6438    0.6986    0.8082    0.8493    0.8767    0.9452    0.9863
    0.9863    0.9863    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000

METHOD     3   LV8 - M1     MEAN VALUE =    0.9615   EFFICIENCY =    1.0000

    0.6164    0.8356    0.9041    0.9452    0.9589    0.9589    0.9726    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
    1.0000    1.0000    1.0000    1.0000    1.0000
```

The file P.TEX, generated by the procedure PROFTEX, contains the following statements:

```
 \documentclass{article}
 \usepackage{epsfig,color}
 \def\plinr#1#2#3{{\linethickness{.8pt}\color{red}
 \multiput(#1,#2)(.4,#3){20}{\line(1,0){.4}\line(0,1){#3}}}}
 \def\plinc#1#2#3{{\linethickness{.4pt}\color{cyan}
 \multiput(#1,#2)(.4,#3){20}{\line(1,0){.4}\line(0,1){#3}}}}
 \def\plinb#1#2#3{{\linethickness{2pt}\color{blue}
 \multiput(#1,#2)(.4,#3){20}{\line(1,0){.4}\line(0,1){#3}}}}
 \def\ping#1#2#3{{\linethickness{2pt}\color{green}
 \multiput(#1,#2)(.4,#3){20}{\line(1,0){.4}\line(0,1){#3}}}}
 \begin{document}
 \setlength{\unitlength}{0.70mm}
 \begin{picture}(200,80)(0,0)
 \put(20,20){\framebox(160,100)}{ }
 \put(104,24.5){\framebox(66,19){ }}
 \plinb{107}{39.5}{0}\plinb{115}{39.5}{0}%
 \plinc{107}{34.0}{0}\plinc{115}{34.0}{0}%
 \plinr{107}{28.5}{0}\plinr{115}{28.5}{0}%
\put(55,0){\large \bf Performance profile for NFV }
 \multiput(52, 20)(32,0){4}{\line(0, 1){2}}
 \multiput(52,120)(32,0){4}{\line(0,-1){2}}
 \put(19,13){\footnotesize 0}
 \put( 48,13){\footnotesize 0.4}
 \put( 80,13){\footnotesize 0.8}
 \put(112,13){\footnotesize 1.2}
 \put(144,13){\footnotesize 1.6}
 \put(174,13){\footnotesize 2.0}
 \put(184,17.5){\small $\tau$}
 \multiput( 20,30)(0,10){9}{\line( 1,0){2}}
 \multiput(180,30)(0,10){9}{\line(-1,0){2}}
 \put(4,119){\small $\pi$}
 \put(15,17.5){\footnotesize 0}\put(15,117.5){\footnotesize 1}
 \put(9,28){\footnotesize 0.1}\put(9,38){\footnotesize 0.2}
 \put(9,48){\footnotesize 0.3}\put(9,58){\footnotesize 0.4}
 \put(9,68){\footnotesize 0.5}\put(9,78){\footnotesize 0.6}
 \put(9,88){\footnotesize 0.7}\put(9,98){\footnotesize 0.8}
 \put(9,108){\footnotesize 0.9}
\put(132,37.0){\footnotesize LV1 - M1  }
\plinb{ 20.}{ 37.8}{1.0}%
\plinb{ 28.}{ 58.4}{1.0}%
\plinb{ 36.}{ 77.5}{0.3}%
\plinb{ 44.}{ 84.4}{0.4}%
\plinb{ 52.}{ 92.6}{0.4}%
\plinb{ 60.}{100.8}{0.2}%
\plinb{ 68.}{104.9}{0.1}%
\plinb{ 76.}{106.3}{0.1}%
\plinb{ 84.}{107.7}{0.0}%
\plinb{ 92.}{107.7}{0.1}%
\plinb{100.}{109.0}{0.1}%
\plinb{108.}{111.8}{0.1}%
```

```
\plinb{116.}{113.2}{0.1}%
\plinb{124.}{114.5}{0.1}%
\plinb{132.}{115.9}{0.0}%
\plinb{140.}{115.9}{0.0}%
\plinb{148.}{115.9}{0.0}%
\plinb{156.}{115.9}{0.1}%
\plinb{164.}{117.3}{0.0}%
\plinb{172.}{117.3}{0.1}%
\put(132,31.5){\footnotesize LV6 - M1  }
\plinc{ 20.}{ 66.6}{0.9}%
\plinc{ 28.}{ 84.4}{0.3}%
\plinc{ 36.}{ 89.9}{0.5}%
\plinc{ 44.}{100.8}{0.2}%
\plinc{ 52.}{104.9}{0.1}%
\plinc{ 60.}{107.7}{0.3}%
\plinc{ 68.}{114.5}{0.2}%
\plinc{ 76.}{118.6}{0.0}%
\plinc{ 84.}{118.6}{0.0}%
\plinc{ 92.}{118.6}{0.1}%
\plinc{100.}{120.0}{0.0}%
\plinc{108.}{120.0}{0.0}%
\plinc{116.}{120.0}{0.0}%
\plinc{124.}{120.0}{0.0}%
\plinc{132.}{120.0}{0.0}%
\plinc{140.}{120.0}{0.0}%
\plinc{148.}{120.0}{0.0}%
\plinc{156.}{120.0}{0.0}%
\plinc{164.}{120.0}{0.0}%
\plinc{172.}{120.0}{0.0}%
\put(132,26.0){\footnotesize LV8 - M1  }
\plinr{ 20.}{ 81.6}{1.1}%
\plinr{ 28.}{103.6}{0.3}%
\plinr{ 36.}{110.4}{0.2}%
\plinr{ 44.}{114.5}{0.1}%
\plinr{ 52.}{115.9}{0.0}%
\plinr{ 60.}{115.9}{0.1}%
\plinr{ 68.}{117.3}{0.1}%
\plinr{ 76.}{120.0}{0.0}%
\plinr{ 84.}{120.0}{0.0}%
\plinr{ 92.}{120.0}{0.0}%
\plinr{100.}{120.0}{0.0}%
\plinr{108.}{120.0}{0.0}%
\plinr{116.}{120.0}{0.0}%
\plinr{124.}{120.0}{0.0}%
\plinr{132.}{120.0}{0.0}%
\plinr{140.}{120.0}{0.0}%
\plinr{148.}{120.0}{0.0}%
\plinr{156.}{120.0}{0.0}%
\plinr{164.}{120.0}{0.0}%
\plinr{172.}{120.0}{0.0}%
 \end{picture}
 \end{document}
```

The picture obtained by the LATEX environment has the following graphical form:

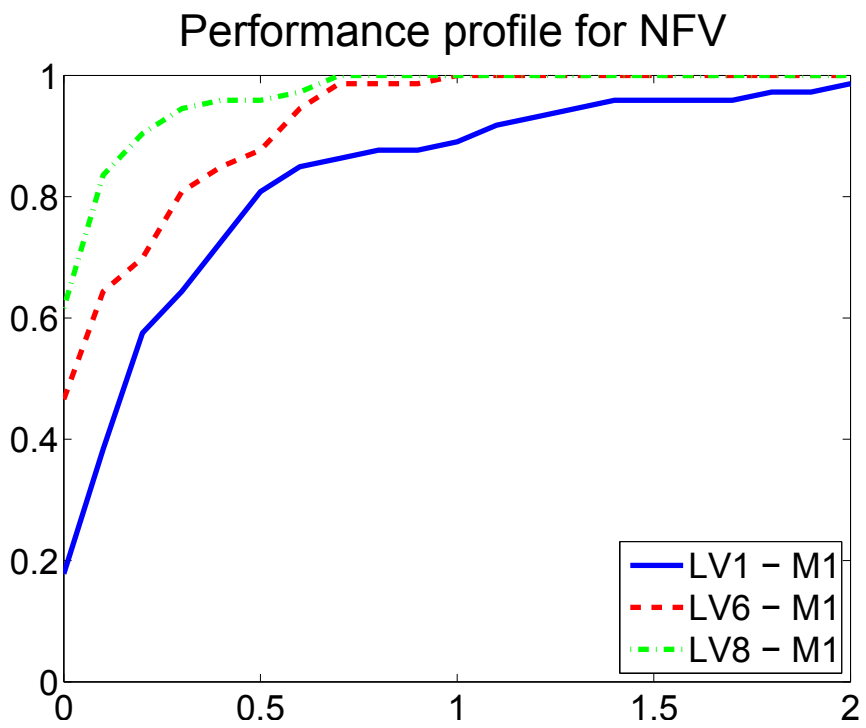


Performance profile for NFV

The file `P.m`, generated by the procedure `PROFMAT`, contains the following statements:

```
x=0: 0.10000: 2.00000;
y1=[ ...
 0.17810 0.38360 0.57530 0.64380 0.72600 0.80820 0.84930 0.86300 0.87670 0.87670 ...
 0.89040 0.91780 0.93150 0.94520 0.95890 0.95890 0.95890 0.95890 0.97260 0.97260 ...
 0.98630 ...
];
y2=[ ...
 0.46580 0.64380 0.69860 0.80820 0.84930 0.87670 0.94520 0.98630 0.98630 0.98630 ...
 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 ...
 1.00000 ...
];
y3=[ ...
 0.61640 0.83560 0.90410 0.94520 0.95890 0.95890 0.97260 1.00000 1.00000 1.00000 ...
 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 ...
 1.00000 ...
];
h = plot(x,y1,x,y2,x,y3);
set(h,{'LineWidth'},{3;3;3});
set(h,{'LineStyle'}, {'-';'--';'-.'});
set(h,{'color'},{'b';'r';'g'});
leg=legend(h,'LV1 - M1  ','LV6 - M1  ','LV8 - M1  ','location','SouthEast');
tit=title('Performance profile for NFV ');
set(gca,'FontSize',20);
set(leg,'FontSize',20);
set(tit,'FontSize',25);
print -depsc P
```

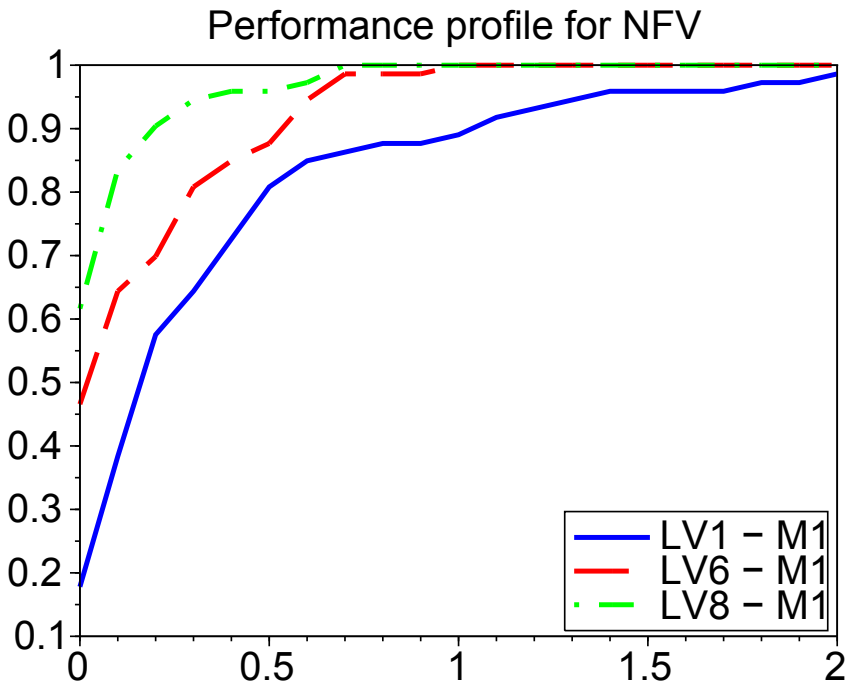
The picture obtained by the MATLAB plotter has the following graphical form:



The file `P.sci`, generated by the procedure `PROFSCI`, contains the following statements:

```
x=0: 0.10000: 2.00000;
y1=[ ...
 0.17810 0.38360 0.57530 0.64380 0.72600 0.80820 0.84930 0.86300 0.87670 0.87670 ...
 0.89040 0.91780 0.93150 0.94520 0.95890 0.95890 0.95890 0.95890 0.97260 0.97260 ...
 0.98630 ...
];
y2=[ ...
 0.46580 0.64380 0.69860 0.80820 0.84930 0.87670 0.94520 0.98630 0.98630 0.98630 ...
 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 ...
 1.00000 ...
];
y3=[ ...
 0.61640 0.83560 0.90410 0.94520 0.95890 0.95890 0.97260 1.00000 1.00000 1.00000 ...
 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 1.00000 ...
 1.00000 ...
];
plot(x,y1,'-b',x,y2,'--r',x,y3,'-.g');
a=gca();
a.font_size=5;
e=gce();
e.children(1).thickness=3;
e.children(2).thickness=3;
e.children(3).thickness=3;
leg=legend(['LV1 - M1  ';'LV6 - M1  ';'LV8 - M1  '],4);
leg.font_size=5;
title('Performance profile for NFV ','fontsize',5);
xs2eps(0,'P');
```

The picture obtained by the SCILAB plotter has the following graphical form:



6.5 Printing sparsity patterns

If the sparsity patterns of the Hessian matrix or the Jacobian matrices are specified (section 2.3, 2.6, 2.14), then these sparsity patterns can be represented in the text file `P.PAT`. In this case we set $PATTERN='Y' (the default value is $PATTERN='N'). The zero elements of sparse metrices are expressed by points and the nonzero elements by asterisks. If we use the batch input file `P.UFO` of the form

```
$SET(INPUT)
  CALL EIUB25(NF,NA,MA,X,IAG,JAG,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU25(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU25(NF,IAG,JAG,KA,X,GA,NEXT)
$ENDSET
$MODEL='AF'
$NF=40
$NA=400
$MA=4000
$JACA='S'
$HESF='S'
$CLASS='VM'
$TYPE='L'
```

```
$DECOMP='M'
$NUMBER=1
$UPDATE='B'
$NEXT=63
$PATTERN='Y'
$BATCH
$STANDARD
```

then the output file P.PAT contains the following text

```
HESSIAN MATRIX HF   NEXT = 63

***..**....*.....................
****..***....*.....................
*****..***....*.....................
.*****..***....*.....................
..****...***....*.....................
...***....**.....*.....................
**....***...**....*.....................
***...****..***....*.....................
.***..*****..***....*.....................
..***..*****..***....*.....................
...***..****...***....*.....................
....**...***....**.....*.....................
*.....**....***...**....*.....................
.*....***...****..***....*.....................
..*....***..*****..***....*.....................
...*....***..*****..***....*.....................
....*....***..****...***....*.....................
.....*....**...***....**.....*.....................
......*.....**....***...**....*.....................
.......*....***...****..***....*.....................
........*....***..*****..***....*.....................
.........*....***..*****..***....*.....................
..........*....***..****...***....*.....................
............*....**...***....**.....*.....................
............*.....**....***...**.....................
.............*....***...****..***.....................
..............*....***..*****..***.....................
...............*....***..*****..***.....................
................*....***..****...***.....................
.................*....**...***....**.....................
...................*.....**....***.....................
....................*....***...****.....................
.....................*....***..*****.....................
......................*....***..*****.....................
.......................*....***..****.....................
........................*....**...***.....................
```

```
JACOBIAN MATRIX AG   NEXT = 63

**...*.........................
***....*.......................
.***....*......................
..***....*.....................
...***....*....................
....**.....*...................
*.....**....*..................
.*....***....*.................
..*....***....*................
...*....***....*...............
....*....***....*..............
.....*....**.....*.............
......*.....**....*............
.......*....***....*...........
........*....***....*..........
.........*....***....*.........
..........*....**.....*........
...........*.....**....*.......
............*....***....*......
.............*....***....*.....
..............*....***....*....
...............*....**.....*...
................*.....**....*..
.................*....***....*.
..................*....**.....*
...................*.....**....
....................*....***...
.....................*....***..
......................*....***.
.......................*....***
........................*....**
```

6.6 Interface to the CUTE environment

The CUTE environment [25] contains the most famous collection of problems for testing optimization methods. The individual test problems are described by the special so-called SIF format. Therefore, a SIF decoder is necessary. Such a decoder is part of the CUTE collection, but this version can only be used for a relatively small set of optimization codes (e.g. for the LANCELOT [52] code). Since the UFO system has a special nature the original SIF decoder had to be modified. This modification consists in replacing the subroutine `SDLANC` by the subroutine `SDUFO` and in preparing new interface subroutines `SIUBXX`, `SIUDXX`, `SIUSXX`, `SINBXX`, `SINDXX`, `SINSXX`, `SFFUXX`, `SFGUXX`, `SFFGUX`, `SCFUXX`, `SCGUXX`, `SCFGUX`, `SAFUXX`, `SAGUXX`, `SAFGUX` (instead of `USETUP`, `UFN`, `UGR`, `CSETUP`, `CFN`, `CGR` etc.).

The CUTE collection is not distributed with the UFO system. The SIF files together with SIF decoder subroutines can to be obtained from their authors (they are also available on the INTERNET address `http://www.cuter.rl.ac.uk/`). The special interface subroutines listed above are exceptions (they are

contained in the library CUTELIB.LIB). The SIF files should be stored in the subdirectory SIF of the main UFO directory.

If we want to use a SIF file for testing the UFO system methods, it suffices to write the macroinstruction $SIF='SIF_file_name' in the input batch file. For example, if we want to use problem BRITGAS for testing interior point methods for sparse inequality constrained nonlinear programming problems, then the problem specification (input batch file) has the following form.

```
$SIF='BRITGAS'
$JACC='S'
$FORM='SI'
$XMAX='5.0D0'
$BATCH
$STANDARD
```

Here BRITGAS is a name of the SIF file, $JACC='S' means that the sparse Jacobian matrix is considered and SI is the form of recursive quadratic programming methods (Section 3.32). Statement XMAX='5.0D0' serves for bounding the stepsize. The problem solution (basic screen output) has the following form:

```
 CLASS = MN - LI3   UPDATE = N   MODEL = FF   HESF = S     NF =    450
     NIC=  0 NIT=   0 NFV=    1 NFG=   19 F=0.240D+00 C=0.200D+01 G=0.220D+00
     NIC=  0 NIT=   1 NFV=    2 NFG=   38 F=-.521D+01 C=0.196D+01 G=0.257D+00
     NIC=  0 NIT=   2 NFV=    3 NFG=   57 F=0.296D+02 C=0.153D+01 G=0.196D+00
     NIC=  0 NIT=   3 NFV=    4 NFG=   76 F=0.123D+02 C=0.120D+01 G=0.156D+00
     NIC=  0 NIT=   4 NFV=    5 NFG=   95 F=0.571D+01 C=0.912D+00 G=0.125D+00
     NIC=  0 NIT=   5 NFV=    6 NFG=  114 F=0.255D+01 C=0.651D+00 G=0.915D-01
     NIC=  0 NIT=   6 NFV=    7 NFG=  133 F=0.102D+01 C=0.406D+00 G=0.574D-01
     NIC=  0 NIT=   7 NFV=    8 NFG=  152 F=0.352D+00 C=0.187D+00 G=0.240D-01
     NIC=  0 NIT=   8 NFV=    9 NFG=  171 F=0.730D-01 C=0.533D-01 G=0.100D-01
     NIC=  0 NIT=   9 NFV=   10 NFG=  190 F=0.392D-02 C=0.108D-01 G=0.378D-01
     NIC=  0 NIT=  10 NFV=   11 NFG=  209 F=0.534D-03 C=0.306D-03 G=0.899D-02
     NIC=  0 NIT=  11 NFV=   12 NFG=  228 F=0.283D-04 C=0.147D-05 G=0.403D-03
     NIC=  0 NIT=  12 NFV=   13 NFG=  247 F=0.149D-05 C=0.562D-03 G=0.215D-04
     NIC=  0 NIT=  13 NFV=   14 NFG=  266 F=0.783D-07 C=0.113D-07 G=0.106D-05
     NIC=  0 NIT=  14 NFV=   15 NFG=  285 F=0.412D-08 C=0.776D-06 G=0.538D-07
     NIC=  0 NIT=  15 NFV=   16 NFG=  304 F=0.216D-09 C=0.752D-06 G=0.269D-08
     NIC=  0 NIT=  16 NFV=   17 NFG=  323 F=0.113D-10 C=0.549D-06 G=0.134D-09
   0 NIC=  0 NIT=  17 NFV=   17 NFG=  323 F=0.984D-13 C=0.549D-06 G=0.134D-09
 TIME= 0:00:00.17
```

The additional output file P.SIF has the following form:

```
 Problem name: BRITGAS
 The objective function uses      1 nonlinear group

 There are    360 nonlinear equality constraints

 There are    426 variables bounded only from below
 There are     24 variables bounded from below and above
```

If the sparsity pattern contains a relatively great number of nonzero elements, then default dimensions (e.g. $M, $MA, $MAH, $MHA, $MC, $MCH, $MHC) might be too small and therefore they must be specified in the input batch file.

7 Application of the UFO system (examples)

Before the solution of a given problem, the input file containing the problem description and other specifications for the macroprocessor must be usually prepared. This input file can contain only the macroinstruction $STANDARD (input file `STANDARD.UFO`). Then, a full dialogue is processed. However, a more advantageous possibility is to prepare an input file containing the problem description while a method selection is left to the dialogue. Moreover, since a method selection can be made automatically by using knowledge bases coded in UFO templates, the batch mode is recommended.

When writing input file instructions, we have to observe some conventions. Since the UFO source program contains a great number of common variables we recommend using variables beginning with the letter 'W' for the problem description to avoid their double use. Real variables of this type should be declared at the beginning of the UFO source program by the statement $FLOAT (for example $FLOAT W,W1,W2). Simple integers I, J, K, L need not be declared. We recommend using statement numbers smaller than 10000 for the problem description to avoid their double use.

The basic implementation of the UFO system is in a double precision arithmetic. Therefore, usually $FLOAT='REAL*8' and $P='D'. We recommend writing real constants always in the form of $P or D specification (for example 1.0$P 2, 4.0$P-1 or 1.0D2, 4.0D-1) since the conversions from a single precision, which depend on a compiler, can be incorrect. Instead of constants 0.0D0, 0.5D0, 1.0D0, 2.0D0, 3.0D0, 4.0D0, 5.0D0, 1.0D1, we can use the common variables ZERO, HALF, ONE, TWO, THREE, FOUR, FIVE, TEN, which contain corresponding values.

In the following text, we demonstrate the application of the UFO system to 36 typical problems. Every example consists of the problem description, the problem specification (input file), comments on the problem specification and the problem solution (basic screen output) obtained on a PC computer. All input files contain the necessary data and can be used in the batch mode. These input files are included into the UFO system as the demo-files `PROB01.UFO`,... ,`PROB36.UFO`.

7.1 Optimization with simple bounds

a) Problem description:

Suppose we have to find a local maximum of the objective function

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

with simple bounds $0 \le x_i \le i$ for $1 \le i \le n$, where $n = 5$. The starting point is $x_i = 2$ for $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE BOX CONSTRAINED MAXIMIZATION                          +
$REM +-------------------------------------------------------------+
$FLOAT W
$SET(INPUT)
  DO 1 I=1,NF
  X(I)=2.0D0; XL(I)=0.0D0; XU(I)=DBLE(I); IX(I)=3
1 CONTINUE
$ENDSET
$SET(FGMODELF)
  W=1.0D0
  DO 2 I=1,NF
  W=W*X(I)/DBLE(I)
2 CONTINUE
  FF=W-2.0D0
```

```
  DO 3 I=1,NF
  GF(I)=W/X(I)
3 CONTINUE
$ENDSET
$IEXT=1
$NF=5
$NX=5
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values and the simple bounds for variables. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the model function. Because we look for a maximum, we set $IEXT=1.

d) Problem solution (basic screen output):

```
 CLASS = VM - LG1   UPDATE = B   MODEL = FF   HESF = D     NF =      5
     NIT=    0 NFV=    1 NFG=    1  F=  1.866666667     G=0.667D-01
     NIT=    1 NFV=    4 NFG=    4  F=  1.550000000     G=0.150D+00
     NIT=    2 NFV=    7 NFG=    7  F=  1.200000000     G=0.200D+00
     NIT=    3 NFV=    9 NFG=    9  F=  1.000000000     G=0.000D+00
   0 NIT=    3 NFV=    9 NFG=    9  GRAD TOL  F=  1.000000000     G=0.000D+00
 FF = -0.1000000000D+01
 X  =  0.1000000000D+01  0.2000000000D+01  0.3000000000D+01  0.4000000000D+01
       0.5000000000D+01
 TIME= 0:00:00.00
```

7.2 Unconstrained least squares optimization

a) Problem description:

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

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

where $m = 20$, $t_i = i/10$ and $y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}$ for $1 \le i \le m$. The starting point is $x_1 = 1,\ x_2 = 2,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1,\ x_6 = 1$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE UNCONSTRAINED LEAST SQUARES                           +
$REM +-------------------------------------------------------------+
$FLOAT W,WA,WB,WC
$SET(INPUT)
  X(1)=1.0D0; X(2)=2.0D0; X(3)=1.0D0
  X(4)=1.0D0; X(5)=1.0D0; X(6)=1.0D0
  DO 1 KA=1,NA
    W=0.1D0*DBLE(KA)
    AM(KA)=EXP(-W)-5.0D0*EXP(-1.0D1*W)+3.0D0*EXP(-4.0D0*W)
  1 CONTINUE
$ENDSET
```

```
$SET(FMODELA)
  W=0.1D0*DBLE(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
$MODEL='AQ'
$KBA=1
$MOS1=1
$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 $y_i, 1 \leq i \leq m$. By using the macrovariable \$FMODELA we specify analytically the values of the approximating function. The gradients of the approximating functions are computed numerically. For the sum of squares minimization we set \$MODEL='AQ'. The specification \$KBA=1 indicates that the vector AM is used.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = N   MODEL = AQ   HESF = D     NF =      6
     NIT=    0 NFV=    7 NFG=    0  F= 0.4652437783     G=0.675D+00
     NIT=    1 NFV=   14 NFG=    0  F= 0.1755707488     G=0.172D+00
     NIT=    2 NFV=   22 NFG=    0  F= 0.1310718542     G=0.103D+00
     NIT=    3 NFV=   30 NFG=    0  F= 0.1014547191     G=0.987D-01
     NIT=    4 NFV=   37 NFG=    0  F= 0.8325110963E-01 G=0.315D+00
     NIT=    5 NFV=   44 NFG=    0  F= 0.5149743622E-01 G=0.225D+00
     NIT=    6 NFV=   52 NFG=    0  F= 0.1248663022E-01 G=0.964D-01
     NIT=    7 NFV=   59 NFG=    0  F= 0.4692732603E-02 G=0.479D-01
     NIT=    8 NFV=   66 NFG=    0  F= 0.2350712826E-02 G=0.292D-01
     NIT=    9 NFV=   73 NFG=    0  F= 0.1276233060E-02 G=0.420D-01
     NIT=   10 NFV=   80 NFG=    0  F= 0.7512378805E-03 G=0.482D-01
     NIT=   11 NFV=   87 NFG=    0  F= 0.3065852206E-03 G=0.395D-01
     NIT=   12 NFV=   95 NFG=    0  F= 0.3583746062E-04 G=0.409D-02
     NIT=   13 NFV=  103 NFG=    0  F= 0.2339698105E-04 G=0.374D-03
     NIT=   14 NFV=  110 NFG=    0  F= 0.1221490232E-04 G=0.272D-02
     NIT=   15 NFV=  117 NFG=    0  F= 0.7308108434E-05 G=0.287D-02
     NIT=   16 NFV=  124 NFG=    0  F= 0.3876069070E-05 G=0.202D-02
     NIT=   17 NFV=  131 NFG=    0  F= 0.2023584146E-05 G=0.190D-02
     NIT=   18 NFV=  138 NFG=    0  F= 0.1029329293E-05 G=0.193D-02
     NIT=   19 NFV=  145 NFG=    0  F= 0.6760021848E-06 G=0.196D-02
     NIT=   20 NFV=  152 NFG=    0  F= 0.5022053254E-09 G=0.553D-04
     NIT=   21 NFV=  159 NFG=    0  F= 0.4846290353E-19 G=0.610D-09
   0 NIT=   21 NFV=  159 NFG=    0  FV BOUND  F= 0.4846290353E-19 G=0.610D-09
 F  =  0.4846290353D-19
 X  =  0.1000000000D+01  0.1000000000D+02  0.4000000000D+01  0.1000000000D+01
       0.5000000000D+01  0.3000000000D+01
 TIME= 0:00:00.00
```

7.3 Unconstrained least powers optimization

a) Problem description:

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

$$F(x) = \frac{1}{4}\sum_{i=1}^{m}\left(x_4 e^{-x_1 t_i} - x_5 e^{-x_2 t_i} + x_6 e^{-x_3 t_i} - y_i\right)^4,$$

where $m = 20$, $t_i = i/10$ and $y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}$ for $1 \le i \le m$. The starting point is $x_1 = 1,\ x_2 = 2,\ x_3 = 1,\ x_4 = 1,\ x_5 = 1,\ x_6 = 1$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE UNCONSTRAINED LEAST POWERS                            +
$REM +-------------------------------------------------------------+
$FLOAT W,WA,WB,WC
$SET(INPUT)
  X(1)=1.0D0; X(2)=2.0D0; X(3)=1.0D0
  X(4)=1.0D0; X(5)=1.0D0; X(6)=1.0D0
  DO 1 KA=1,NA
    W=0.1D0*DBLE(KA)
    AM(KA)=EXP(-W)-5.0D0*EXP(-1.0D1*W)+3.0D0*EXP(-4.0D0*W)
  1 CONTINUE
$ENDSET
$SET(FMODELA)
  W=0.1D0*DBLE(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
$MODEL='AP'
$REXP='4.0D0'
$KBA=1
$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 $y_i, 1 \le i \le m$. By using the macrovariable \$FMODELA we specify analytically the values of the approximating function. The gradients of the approximating functions are computed numerically. For the sum of fourth powers minimization we set \$MODEL='AP' and \$REXP='4.0D0' (macrovariable \$REXP defines a corresponding power). The specification \$KBA=1 indicates that the vector AM is used.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = N   MODEL = AP   HESF = D     NF =      6
     NIT=    0 NFV=    7 NFG=    0  F= 0.3430916991E-01 G=0.194D+00
     NIT=    1 NFV=   14 NFG=    0  F= 0.1419887081E-01 G=0.665D-01
     NIT=    2 NFV=   21 NFG=    0  F= 0.4944922455E-02 G=0.111D-01
```

```
    NIT=    3 NFV=   28 NFG=    0  F= 0.3678302435E-02 G=0.124D-02
    NIT=    4 NFV=   35 NFG=    0  F= 0.2424810434E-02 G=0.624D-02
    NIT=    5 NFV=   42 NFG=    0  F= 0.9409619052E-03 G=0.294D-02
    NIT=    6 NFV=   50 NFG=    0  F= 0.6512073068E-03 G=0.111D-02
    NIT=    7 NFV=   57 NFG=    0  F= 0.4137478404E-03 G=0.615D-03
    NIT=    8 NFV=   64 NFG=    0  F= 0.2198851619E-03 G=0.505D-03
    NIT=    9 NFV=   72 NFG=    0  F= 0.1728982452E-03 G=0.122D-03
    NIT=   10 NFV=   79 NFG=    0  F= 0.1213304154E-03 G=0.272D-03
    NIT=   11 NFV=   86 NFG=    0  F= 0.6351299654E-04 G=0.679D-03
    NIT=   12 NFV=   93 NFG=    0  F= 0.1567466646E-04 G=0.685D-04
    NIT=   13 NFV=  100 NFG=    0  F= 0.4367175375E-05 G=0.121D-03
    NIT=   14 NFV=  107 NFG=    0  F= 0.9045737965E-06 G=0.307D-04
    NIT=   15 NFV=  114 NFG=    0  F= 0.2000673033E-06 G=0.119D-04
    NIT=   16 NFV=  121 NFG=    0  F= 0.4446122812E-07 G=0.463D-05
    NIT=   17 NFV=  128 NFG=    0  F= 0.9700186794E-08 G=0.167D-05
    NIT=   18 NFV=  135 NFG=    0  F= 0.2066473108E-08 G=0.561D-06
  0 NIT=   18 NFV=  135 NFG=    0  GRAD TOL  F= 0.2066473108E-08 G=0.561D-06
 F  =  0.2066473108D-08
 X  =  0.1014994274D+01  0.9976207912D+01  0.3907664874D+01  0.1011751654D+01
       0.4777298750D+01  0.2851199655D+01
 TIME= 0:00:00.00
```

7.4 Unconstrained minimax optimization

a) Problem description:

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

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

where $m = 21,\ t_i = (i-1)/10 - 1$ and $y_i = e^{t_i}$ for $1 \le i \le m$. The starting point is $x_1 = 0.5,\ x_2 = 0,\ x_3 = 0,\ x_4 = 0,\ x_5 = 0$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + DENSE UNCONSTRAINED MINIMAX OPTIMIZATION                      +
$REM +---------------------------------------------------------------+
$FLOAT W
$SET(INPUT)
  X(1)=0.5D0; X(2)=0.0D0; X(3)=0.0D0; X(4)=0.0D0; X(5)=0.0D0
$ENDSET
$SET(FMODELA)
  W=0.1D0*DBLE(KA-1)-1.0D0
  FA=(X(1)+W*X(2))/(1.0D0+W*(X(3)+W*(X(4)+W*X(5))))-EXP(W)
$ENDSET
$MODEL='AM'
$NF=5
$NA=21
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FMODELA we specify analytically the values of the approximating functions. The gradients of the approximating functions are computed numerically. For minimax approximation we set $MODEL='AM'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = AM   HESF = D     NF =      5
     NIT=    0 NFV=    6 NFG=    0  F=  2.218281828     G=0.100+121
     NIT=    1 NFV=   13 NFG=    0  F= 0.4253776661     G=0.783D+00
     NIT=    2 NFV=   19 NFG=    0  F= 0.8271278131E-01 G=0.223D+00
     NIT=    3 NFV=   25 NFG=    0  F= 0.1256835525E-01 G=0.114D+00
     NIT=    4 NFV=   31 NFG=    0  F= 0.6693249048E-02 G=0.290D-01
     NIT=    5 NFV=   37 NFG=    0  F= 0.6121578248E-02 G=0.140D-01
     NIT=    6 NFV=   43 NFG=    0  F= 0.1771793241E-02 G=0.550D-01
     NIT=    7 NFV=   49 NFG=    0  F= 0.2044251553E-03 G=0.208D-01
     NIT=    8 NFV=   55 NFG=    0  F= 0.1225222546E-03 G=0.833D-04
     NIT=    9 NFV=   61 NFG=    0  F= 0.1223712525E-03 G=0.240D-07
   0 NIT=    9 NFV=   61 NFG=    0  GRAD TOL  F= 0.1223712525E-03 G=0.240D-07
 F  =  0.1223712525D-03
 X  =  0.9998776287D+00  0.2535884404D+00 -0.7466075717D+00  0.2452015019D+00
      -0.3749029100D-01
 TIME= 0:00:00.00
```

7.5 Unconstrained nonsmooth optimization

a) Problem description:

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

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

The starting point is $x_1 = -1$, $x_2 = -1$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + DENSE UNCONSTRAINED NONSMOOTH OPTIMIZATION                    +
$REM +---------------------------------------------------------------+
$FLOAT W
$SET(INPUT)
  X(1)=-1.0D0
  X(2)=-1.0D0
$ENDSET
$SET(FGMODELF)
  W=X(1)**2+X(2)**2-1.0D0
  FF=-X(1)+2.0D0*W+1.75D0*ABS(W)
  W=SIGN(3.5D0,W)+4.0D0
  GF(1)=W*X(1)-1.0D0
  GF(2)=W*X(2)
$ENDSET
$NF=2
$KSF=3
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the objective function. Since the objective function is nonsmooth, we set $KSF=3.

d) Problem solution (basic screen output):

```
 CLASS = BM - L11   UPDATE = N   MODEL = FF   HESF = N     NF =      2
     NIT=    0 NFV=    1 NFG=    1  F=  4.750000000     G=0.100+121
     NIT=    1 NFV=    3 NFG=    3  F=-0.3788888889     G=0.850D+01
     NIT=    2 NFV=    4 NFG=    4  F=-0.6061514369     G=0.933D+00
     NIT=    3 NFV=    5 NFG=    5  F=  9.250000000     G=0.802D+00
     NIT=    4 NFV=    6 NFG=    6  F=-0.7284826639     G=0.802D+00
     NIT=    5 NFV=    7 NFG=    7  F= 0.8708184712     G=0.348D+00
     NIT=    6 NFV=    8 NFG=    8  F=-0.8275709577     G=0.722D+00
     NIT=    7 NFV=    9 NFG=    9  F=-0.8436035754     G=0.162D+00
     NIT=    8 NFV=   10 NFG=   10  F=-0.9986081259     G=0.984D-01
     NIT=    9 NFV=   11 NFG=   11  F=-0.9980886265     G=0.114D+00
     NIT=   10 NFV=   12 NFG=   12  F=-0.9992851884     G=0.501D+00
     NIT=   11 NFV=   13 NFG=   13  F=-0.9999999867     G=0.530D-01
     NIT=   12 NFV=   14 NFG=   14  F=-0.9999922205     G=0.454D-05
     NIT=   13 NFV=   15 NFG=   15  F= -1.000000000     G=0.986D-07
   0 NIT=   13 NFV=   15 NFG=   15  GRAD TOL  F= -1.000000000     G=0.986D-07
 FF = -0.1000000000D+01
 X  =  0.1000000000D+01  0.0000000000D+00
 TIME= 0:00:00.00
```

7.6 Optimization with linear constraints

a) Problem specification:

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

$$F(x) = (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$$

over the set given by the linear constraints

$$x_1 + x_2 + x_3 + 4x_4 = 7,$$

$$x_3 + 5x_5 = 6.$$

The starting point is $x_1 = 10,\ x_2 = 7,\ x_3 = 2,\ x_4 = -3,\ x_5 = 0.8$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE MINIMIZATION WITH LINEAR CONSTRAINTS                   +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  X(1)= 1.0D1; X(2)=7.0D0; X(3)= 2.0D0
  X(4)=-3.0D0; X(5)=0.8D0
  IC(1)=5; CL(1)=7.0D0
  CG(1)=1.0D0; CG(2)=1.0D0; CG(3)=1.0D0
  CG(4)=4.0D0; CG(5)=0.0D0
  IC(2)=5; CL(2)=6.0D0
```

```
  CG(6)=0.0D0; CG(7)=0.0D0; CG(8)=1.0D0
  CG(9)=0.0D0; CG(10)=5.0D0
$ENDSET
$SET(FMODELF)
  FF=(X(1)-X(2))**2+(X(3)-1.0D0)**2+(X(4)-1.0D0)**4+(X(5)-1.0D0)**6
$ENDSET
$SET(GMODELF)
  GF(1)= 2.0D0*(X(1)-X(2))
  GF(2)=-2.0D0*(X(1)-X(2))
  GF(3)= 2.0D0*(X(3)-1.0D0)
  GF(4)= 4.0D0*(X(4)-1.0D0)**3
  GF(5)= 6.0D0*(X(5)-1.0D0)**5
$ENDSET
$NF=5
$NC=2
$NCL=2
$FMIN='0.0D0'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables and the types and values of the linear constraints. By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. The specification $FMIN='0.0D0' is used, since the objective function value cannot be smaller then zero.

d) Problem solution (basic screen output):

```
 CLASS = VM - LG1   UPDATE = B   MODEL = FF   HESF = D     NF =      5
     NIT=    0 NFV=    1 NFG=    1  F=  266.0000640     G=0.853D+02
     NIT=    1 NFV=    2 NFG=    2  F=  23.36590202     G=0.911D+01
     NIT=    2 NFV=    3 NFG=    3  F=  5.067974228     G=0.427D+01
     NIT=    3 NFV=    4 NFG=    4  F= 0.5961395170     G=0.936D+00
     NIT=    4 NFV=    5 NFG=    5  F= 0.2531886542     G=0.439D+00
     NIT=    5 NFV=    6 NFG=    6  F= 0.1507495393     G=0.284D+00
     NIT=    6 NFV=    7 NFG=    7  F= 0.6239641368E-01 G=0.383D+00
     NIT=    7 NFV=    8 NFG=    8  F= 0.1334510618E-01 G=0.516D-01
     NIT=    8 NFV=    9 NFG=    9  F= 0.6238174366E-02 G=0.619D-01
     NIT=    9 NFV=   10 NFG=   10  F= 0.2151107202E-02 G=0.810D-01
     NIT=   10 NFV=   11 NFG=   11  F= 0.4055819437E-03 G=0.130D-01
     NIT=   11 NFV=   12 NFG=   12  F= 0.1934974160E-03 G=0.430D-02
     NIT=   12 NFV=   13 NFG=   13  F= 0.4675914285E-04 G=0.724D-02
     NIT=   13 NFV=   14 NFG=   14  F= 0.1498471817E-04 G=0.640D-02
     NIT=   14 NFV=   15 NFG=   15  F= 0.5645336646E-05 G=0.927D-03
     NIT=   15 NFV=   16 NFG=   16  F= 0.2238555954E-05 G=0.919D-03
     NIT=   16 NFV=   17 NFG=   17  F= 0.7049245994E-06 G=0.124D-02
     NIT=   17 NFV=   18 NFG=   18  F= 0.1667236198E-06 G=0.118D-03
     NIT=   18 NFV=   19 NFG=   19  F= 0.6896628295E-07 G=0.117D-03
     NIT=   19 NFV=   20 NFG=   20  F= 0.1898543896E-07 G=0.120D-03
     NIT=   20 NFV=   21 NFG=   21  F= 0.5414841480E-08 G=0.871D-04
     NIT=   21 NFV=   22 NFG=   22  F= 0.1679481945E-08 G=0.217D-04
     NIT=   22 NFV=   23 NFG=   23  F= 0.6418583139E-09 G=0.303D-04
```

```
    NIT=   23 NFV=   24 NFG=   24  F= 0.2757671243E-09 G=0.231D-04
    NIT=   24 NFV=   25 NFG=   25  F= 0.7610163021E-10 G=0.472D-05
    NIT=   25 NFV=   26 NFG=   26  F= 0.3874868646E-10 G=0.797D-07
  0 NIT=   25 NFV=   26 NFG=   26  GRAD TOL  F= 0.3874868646E-10 G=0.797D-07
 FF =  0.3874868646D-10
 X  =  0.1004989839D+01  0.1004989867D+01  0.9999999525D+00  0.9975050854D+00
       0.1000000010D+01
 TIME= 0:00:00.00
```

7.7 Least squares optimization with linear constraints

a) Problem description:

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

$$F(x) = \sum_{i=1}^{163} \left(\frac{1}{15} + \frac{2}{15} \sum_{j=1}^{7} \cos(2\pi x_j \sin \vartheta_i) \right)^2 ,$$

where $\vartheta_i = \pi(8.5 + i0.5)/180$ for $1 \leq i \leq 163$, over set given by the simple constraints $x_1 \geq 0.4$, $x_7 = 3.5$ and by the linear constraints

$$\begin{aligned} -x_1 + x_2 &\geq 0.4, \\ -x_2 + x_3 &\geq 0.4, \\ -x_3 + x_4 &\geq 0.4, \\ -x_4 + x_5 &\geq 0.4, \\ -x_5 + x_6 &\geq 0.4, \\ -x_6 + x_7 &\geq 0.4, \\ -x_4 + x_6 &= 1.0. \end{aligned}$$

The starting point is $x_1 = 0.5,\ x_2 = 1.0,\ x_3 = 1.5,\ x_4 = 2.0,\ x_5 = 2.5,\ x_6 = 3.0.\ x_7 = 3.5.$

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + DENSE LEAST SQUARES WITH LINEAR CONSTRAINTS                    +
$REM +---------------------------------------------------------------+
$FLOAT W(163),WA,WPI
$SET(INPUT)
  WPI=3.1415926535897932D0
  DO 1 I=1,NF
    X(I)=DBLE(I)*0.5D0; IX(I)=0
1 CONTINUE
  XL(1)=0.4D0; IX(1)=1; IX(7)=5
  DO 2 I=1,NA
    W(I)=2.0D0*WPI*SIN(WPI*(8.5D0+DBLE(I)*0.5D0)/1.8D2)
2 CONTINUE
  DO 3 I=1,6
    CL(I)=0.4D0; IC(I)=1
3 CONTINUE
  CL(7)=1.0D0; CU(7)=1.0D0; IC(7)=5
  DO 4 I=1,NF*NC
    CG(I)=0.0D0
```

```
4 CONTINUE
  K=0
  DO 5 I=1,6
    CG(K+I)=-1.0D0; CG(K+I+1)=1.0D0
    K=K+NF
5 CONTINUE
  CG(46)=-1.0D0; CG(48)= 1.0D0
$ENDSET
$SET(FMODELA)
  WA=0.0D0
  DO 6 I=1,NF
    WA=WA+COS(W(KA)*X(I))
6 CONTINUE
  FA=(1.0D0+2.0D0*WA)/1.5D1
$ENDSET
$MODEL='AQ'
$NF=7
$NX=7
$NA=163
$NC=7
$NCL=7
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, types and values of the simple bounds and types and values of the linear constraints. By using the macrovariable $FMODELA 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'.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = N   MODEL = AQ   HESF = D     NF =      7
     NIT=    0 NFV=    7 NFG=    0  F= 0.4638565295     G=0.114D+01
     NIT=    1 NFV=   14 NFG=    0  F= 0.4461088576     G=0.327D+00
     NIT=    2 NFV=   21 NFG=    0  F= 0.4348755804     G=0.383D+00
     NIT=    3 NFV=   28 NFG=    0  F= 0.4117773441     G=0.557D+00
     NIT=    4 NFV=   35 NFG=    0  F= 0.3640897673     G=0.831D+00
     NIT=    5 NFV=   42 NFG=    0  F= 0.2855278593     G=0.118D+01
     NIT=    6 NFV=   49 NFG=    0  F= 0.2459276580     G=0.145D+01
     NIT=    7 NFV=   56 NFG=    0  F= 0.2428262783     G=0.805D+00
     NIT=    8 NFV=   63 NFG=    0  F= 0.2346115275     G=0.263D-01
     NIT=    9 NFV=   70 NFG=    0  F= 0.2345064740     G=0.623D-02
     NIT=   10 NFV=   77 NFG=    0  F= 0.2345039665     G=0.752D-03
     NIT=   11 NFV=   84 NFG=    0  F= 0.2345039533     G=0.837D-04
     NIT=   12 NFV=   91 NFG=    0  F= 0.2345039531     G=0.966D-05
     NIT=   13 NFV=   98 NFG=    0  F= 0.2345039531     G=0.111D-05
     NIT=   14 NFV=  105 NFG=    0  F= 0.2345039531     G=0.128D-06
   0 NIT=   14 NFV=  105 NFG=    0  GRAD TOL  F= 0.2345039531     G=0.128D-06

 F  =  0.2345039531D+00
 X  =  0.4000000000D+00  0.8399686447D+00  0.1239968645D+01  0.1761071079D+01
       0.2161071079D+01  0.2761071079D+01  0.3500000000D+01
```

```
 TIME= 0:00:00.00
```

7.8 Minimax optimization with linear constraints

a) Problem description:

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

$$F(x) = \max(-\exp(x_1 - x_2),\ \sinh(x_1 - 1) - 1,\ -\log(x_2) - 1)$$

over the set given by the simple constraint $x_2 \geq 1/100$ and by the linear constraint

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

The starting point is $x_1 = -1,\ x_2 = 1/100$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE MINIMAX OPTIMIZATION WITH LINEAR CONSTRAINTS          +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  X(1)=-1.0D 0;                IX(1)=0
  X(2)= 1.0D-2; XL(2)= 1.0D-2; IX(2)=1
                CL(1)=-5.0D-1; IC(1)=1
  CG(1)=5.0D-2; CG(2)=-1.0D 0
$ENDSET
$SET(FMODELA)
  IF (KA.EQ.1) FA=-EXP(X(1)-X(2))
  IF (KA.EQ.2) FA= SINH(X(1)-1.0D0)-1.0D0
  IF (KA.EQ.3) FA=-LOG(X(2))-1.0D0
$ENDSET
$MODEL='AM'
$IEXT=-1
$NF=2
$NX=2
$NA=3
$NC=1
$NCL=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, types and values of the simple bounds and types and values of the linear constraints. By using the macrovariable $FMODELA we specify analytically the values of the approximating functions. The gradients of the approximating functions are computed numerically. For minimax approximation we set $MODEL='AM' and $IEXT=-1.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = AM   HESF = D     NF =      2
     NIT=    0 NFV=    3 NFG=    0  F=  3.605170186     G=0.100+121
     NIT=    1 NFV=    6 NFG=    0  F=  1.978554385     G=0.363D+00
     NIT=    2 NFV=    9 NFG=    0  F= 0.6817245960     G=0.487D+00
     NIT=    3 NFV=   12 NFG=    0  F=-0.2766371994     G=0.595D+00
     NIT=    4 NFV=   15 NFG=    0  F=-0.3838143331     G=0.344D+00
     NIT=    5 NFV=   18 NFG=    0  F=-0.4394007715     G=0.281D-01
     NIT=    6 NFV=   21 NFG=    0  F=-0.4488931954     G=0.124D-02
     NIT=    7 NFV=   24 NFG=    0  F=-0.4489107860     G=0.215D-05
   0 NIT=    8 NFV=   24 NFG=    0  GRAD TOL  F=-0.4489107860     G=0.737D-11
 F  = -0.4489107860D+00
 X  =  0.1526434615D+01  0.5763217308D+00
 TIME= 0:00:00.00
```

7.9 Nonsmooth optimization with linear constraints

a) Problem description:

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

$$F(x) = \max\left(-\sqrt{(x_1-x_3)^2+(x_2-x_4)^2}, -\sqrt{(x_3-x_5)^2+(x_4-x_6)^2}, -\sqrt{(x_5-x_1)^2+(x_6-x_2)^2}\right)$$

over the set given by the linear constraints

$$x_1\cos(2\pi i/5) + x_2\sin(2\pi i/5) \le 1, \quad 0 \le j \le 4,$$

$$x_3\cos(2\pi i/5) + x_4\sin(2\pi i/5) \le 1, \quad 0 \le j \le 4,$$

$$x_5\cos(2\pi i/5) + x_5\sin(2\pi i/5) \le 1, \quad 0 \le j \le 4.$$

The starting point is $x_1 = -1,\ x_2 = 0,\ x_3 = 0,\ x_4 = -1,\ x_5 = 1,\ x_6 = 1.$

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + DENSE NONSMOOTH OPTIMIZATION WITH LINEAR CONSTRAINTS          +
$REM +---------------------------------------------------------------+
$FLOAT W1,W2,W3,WA,WPI
$SET(INPUT)
  WPI=3.1415926535897932D0
  X(1)=-1.0D0; X(2)= 0.0D0; X(3)= 0.0D0
  X(4)=-1.0D0; X(5)= 1.0D0; X(6)= 1.0D0
  DO 1 I=1,NC
  CU(I)=1.0D0; IC(I)=2
1 CONTINUE
  DO 2 I=1,NF*NC
    CG(I)=0.0D0
2 CONTINUE
  K=1
  DO 4 I=1,3
    L=2*(I-1)
```

```
    DO 3 J=1,5
      CG(K+L)=SIN(2.0D0*WPI*DBLE(J-1)/5.0D0)
      CG(K+L+1)=COS(2.0D0*WPI*DBLE(J-1)/5.0D0)
      K=K+NF
3   CONTINUE
4 CONTINUE
$ENDSET
$SET(FMODELF)
  W1=-SQRT((X(1)-X(3))**2+(X(2)-X(4))**2)
  W2=-SQRT((X(3)-X(5))**2+(X(4)-X(6))**2)
  W3=-SQRT((X(5)-X(1))**2+(X(6)-X(2))**2)
  FF=MAX(W1,W2,W3)
$ENDSET
$NF=6
$NX=6
$NC=15
$NCL=15
$KSF=3
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables and the types and values of the linear constraints. By using the macrovariable $FMODELF we specify analytically the value of the objective function. The subgradients of the objective function is computed numerically. Since the objective function is nonsmooth, we set $KSF=3.

d) Problem solution (basic screen output):

```
 CLASS = BM - L11   UPDATE = N   MODEL = FF   HESF = N     NF =      6
     NIT=    0 NFV=    7 NFG=    0  F= -1.414213562     G=0.100+121
     NIT=    1 NFV=   14 NFG=    0  F= -1.723937907     G=0.447D+00
     NIT=    2 NFV=   21 NFG=    0  F= -1.769198225     G=0.800D-01
     NIT=    3 NFV=   28 NFG=    0  F= -1.819486899     G=0.106D+00
     NIT=    4 NFV=   35 NFG=    0  F= -1.831224488     G=0.807D-01
     NIT=    5 NFV=   42 NFG=    0  F= -1.833047757     G=0.802D-01
     NIT=    6 NFV=   49 NFG=    0  F= -1.725011873     G=0.801D-01
     NIT=    7 NFV=   56 NFG=    0  F= -1.843442148     G=0.791D-01
     NIT=    8 NFV=   63 NFG=    0  F= -1.852803685     G=0.609D-01
     NIT=    9 NFV=   70 NFG=    0  F= -1.827414255     G=0.609D-01
     NIT=   10 NFV=   77 NFG=    0  F= -1.857698751     G=0.324D-01
     NIT=   11 NFV=   84 NFG=    0  F= -1.857554478     G=0.310D-01
     NIT=   12 NFV=   91 NFG=    0  F= -1.859265289     G=0.134D+00
     NIT=   13 NFV=   98 NFG=    0  F= -1.859548882     G=0.173D-01
     NIT=   14 NFV=  105 NFG=    0  F= -1.859596171     G=0.135D-01
     NIT=   15 NFV=  112 NFG=    0  F= -1.859614932     G=0.166D-03
     NIT=   16 NFV=  119 NFG=    0  F= -1.859617277     G=0.980D-05
     NIT=   17 NFV=  126 NFG=    0  F= -1.859615967     G=0.206D-04
     NIT=   18 NFV=  133 NFG=    0  F= -1.859618401     G=0.158D-04
     NIT=   19 NFV=  140 NFG=    0  F= -1.859618685     G=0.573D-05
     NIT=   20 NFV=  147 NFG=    0  F= -1.859618696     G=0.220D-07
   0 NIT=   20 NFV=  147 NFG=    0  GRAD TOL  F= -1.859618696     G=0.220D-07
```

```
 FF = -0.1859618696D+01
 X  = -0.9723036844D+00  0.2436249348D+00  0.5321594418D+00 -0.8494315113D+00
       0.7265425280D+00  0.1000000000D+01
 TIME= 0:00:00.00
```

7.10 Optimization with nonlinear constraints (nonlinear programming)

a) Problem description:

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

$$F(x) = x_1 x_3$$

over the set given by the simple bounds $x_1 \geq 0,\ x_3 \geq 0,\ x_5 \geq 1,\ x_7 \geq 1$ 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 $x_1 = 3.0,\ x_2 = 0.0,\ x_3 = 2.0,\ x_4 = -1.5,\ x_5 = 1.5,\ x_6 = 5.0,\ x_7 = 0.0.$

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE MINIMIZATION WITH NONLINEAR CONSTRAINTS                +
$REM +-------------------------------------------------------------+
$FLOAT W
$SET(INPUT)
  X(1)= 3.0D0; XL(1)= 0.0D0; IX(1)= 1
  X(2)= 0.0D0
  X(3)= 2.0D0; XL(3)= 0.0D0; IX(3)= 1
  X(4)=-1.5D0
  X(5)= 1.5D0; XL(5)= 1.0D0; IX(5)= 1
  X(6)= 5.0D0
  X(7)= 0.0D0; XL(7)= 1.0D0; IX(7)= 1
  CL(1)=4.0D0; IC(1)= 1
  CL(2)=1.0D0; IC(2)= 1
  CL(3)=1.0D0; IC(3)= 1
  CL(4)=1.0D0; IC(4)= 1
  CL(5)=1.0D0; IC(5)= 1
$ENDSET
$SET(FMODELF)
  FF=X(1)*X(3)
$ENDSET
```

```
$SET(FMODELC)
  IF(KC.LE.0)THEN
  ELSEIF(KC.EQ.1)THEN
    FC=(X(4)-X(6))**2+(X(5)-X(7))**2
  ELSEIF(KC.EQ.2)THEN
    W=SQRT(X(2)**2+X(3)**2)
    FC=(X(3)*X(4)-X(2)*X(5))/W
  ELSEIF(KC.EQ.3)THEN
    W=SQRT(X(2)**2+X(3)**2)
    FC=(X(3)*X(6)-X(2)*X(7))/W
  ELSEIF(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
  ELSEIF(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
$IADF=1
$IADC=1
$NF=7
$NX=7
$NC=5
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of the variables, types and values of the simple bounds and types and values of the nonlinear constraints. By using the macrovariable $FMODELF we specify the value of the model function analytically. The gradient of the model function is computed by automatic differentiation, since $IADF=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = FF   HESF = D     NF =      7
     NIC=  0 NIT=   0 NFV=    1 NFG=    1 F=0.600D+01 C=0.294D+01 G=0.000D+00
     NIC=  0 NIT=   1 NFV=    3 NFG=    3 F=0.340D+02 C=0.961D+00 G=0.267D+01
     NIC=  0 NIT=   2 NFV=    5 NFG=    5 F=0.292D+02 C=0.540D-01 G=0.139D+01
     NIC=  0 NIT=   3 NFV=    7 NFG=    7 F=0.247D+02 C=0.156D-01 G=0.105D+01
     NIC=  0 NIT=   4 NFV=    9 NFG=    9 F=0.236D+02 C=0.356D-01 G=0.691D+00
     NIC=  0 NIT=   5 NFV=   11 NFG=   11 F=0.235D+02 C=0.277D-01 G=0.797D+00
     NIC=  0 NIT=   6 NFV=   13 NFG=   13 F=0.233D+02 C=0.171D-02 G=0.127D+00
     NIC=  0 NIT=   7 NFV=   15 NFG=   15 F=0.233D+02 C=0.122D-03 G=0.107D+00
     NIC=  0 NIT=   8 NFV=   17 NFG=   17 F=0.233D+02 C=0.345D-03 G=0.182D+00
     NIC=  0 NIT=   9 NFV=   19 NFG=   19 F=0.233D+02 C=0.122D-04 G=0.101D+00
     NIC=  0 NIT=  10 NFV=   21 NFG=   21 F=0.233D+02 C=0.188D-04 G=0.308D-01
     NIC=  0 NIT=  11 NFV=   23 NFG=   23 F=0.233D+02 C=0.604D-08 G=0.457D-03
     NIC=  0 NIT=  12 NFV=   25 NFG=   25 F=0.233D+02 C=0.701D-10 G=0.707D-04
     NIC=  0 NIT=  13 NFV=   27 NFG=   27 F=0.233D+02 C=0.611D-13 G=0.171D-05
   0 NIC=  0 NIT=  14 NFV=   27 NFG=   27 F=0.233D+02 C=0.611D-13 G=0.228D-09
```

```
 FF =  0.2331370850D+02
 X  =  0.4828427125D+01 -0.4592416296D-10  0.4828427125D+01  0.1000000000D+01
       0.2414213562D+01  0.2414213562D+01  0.1000000000D+01
 TIME= 0:00:00.00
```

7.11 Least squares optimization with nonlinear constraints

a) Problem description:

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

$$F(x) = (x_1 - x_4)^2 + (x_2 - x_5)^2 + (x_3 - x_6)^2$$

over the set given by the simple bound $4 \leq x_6 \leq 8$ and by the nonlinear constraints

$$x_1^2 + x_2^2 + x_3^2 \leq 5,$$

$$x_5^2 + (x_4 - 3)^2 \leq 1.$$

The starting point is $x_1 = 1,\ x_2 = 1\ , x_3 = 1,\ x_4 = 3,\ x_5 = 0,\ x_6 = 0.5$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE LEAST SQUARES WITH NONLINEAR CONSTRAINTS              +
$REM +-------------------------------------------------------------+

$SET(INPUT)
  X(1)=1.0D0; X(2)=1.0D0; X(3)=1.0D0
  X(4)=3.0D0; X(5)=0.0D0; X(6)=0.5D0
  IX(6)=3; XL(6)=4.0D0; XU(6)=8.0D0
  IC(1)=2; CU(1)=5.0D0
  IC(2)=2; CU(2)=1.0D0
$ENDSET
$SET(FMODELA)
  IF(KA.EQ.1)THEN
    FA=X(1)-X(4)
  ELSEIF(KA.EQ.2)THEN
    FA=X(2)-X(5)
  ELSE
    FA=X(3)-X(6)
  ENDIF
$ENDSET
$SET(FMODELC)
  IF(KC.EQ.1)THEN
    FC=X(1)**2+X(2)**2+X(3)**2
  ELSE
    FC=X(5)**2+(X(4)-3.0D0)**2
  ENDIF
$ENDSET
$IADA=1
$IADC=1
$NF=6
$NX=6
$NA=3
```

```
$NC=2
$MODEL='AQ'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, types and values of the simple bounds and types and values of the nonlinear constraints. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum of squares minimization we set $MODEL='AQ'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = AQ   HESF = D     NF =      6
     NIC=  0 NIT=   0 NFV=    1 NFG=    1 F=0.700D+01 C=0.000D+00 G=0.000D+00
     NIC=  0 NIT=   1 NFV=    4 NFG=    4 F=0.363D+01 C=0.125D+01 G=0.200D+01
     NIC=  0 NIT=   2 NFV=    6 NFG=    6 F=0.266D+01 C=0.158D+00 G=0.111D+01
     NIC=  0 NIT=   3 NFV=    8 NFG=    8 F=0.254D+01 C=0.315D-01 G=0.357D+00
     NIC=  0 NIT=   4 NFV=   10 NFG=   10 F=0.250D+01 C=0.320D-02 G=0.130D+00
     NIC=  0 NIT=   5 NFV=   12 NFG=   12 F=0.250D+01 C=0.230D-02 G=0.533D-01
     NIC=  0 NIT=   6 NFV=   14 NFG=   14 F=0.250D+01 C=0.798D-05 G=0.529D-02
     NIC=  0 NIT=   7 NFV=   16 NFG=   16 F=0.250D+01 C=0.876D-07 G=0.431D-03
     NIC=  0 NIT=   8 NFV=   18 NFG=   18 F=0.250D+01 C=0.934D-10 G=0.200D-04
   0 NIC=  0 NIT=   9 NFV=   18 NFG=   18 F=0.250D+01 C=0.934D-10 G=0.864D-07
 F  =  0.2500000000D+01
 X  =  0.1000000020D+01 -0.6092964516D-08  0.1999999990D+01  0.2000000000D+01
       0.4014414082D-07  0.4000000000D+01
 TIME= 0:00:00.02
```

7.12 Minimax optimization with nonlinear constraints

a) Problem description:

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

$$F(x) = \max\left(x_1 - 14,\ x_1^2 - 4x_1 + x_2^2 - 4x_2 + x_3^2 - 4x_3,\ x_1^2 - 4x_1 - 8\right)$$

over the set given by the nonlinear constraints

$$6x_2 + 4x_3 - x_1^3 \geq 3,$$

$$8x_1 + 14x_2 + 7x_3 = 56,$$

$$6x_1^2 + x_2^2 + x_3^2 = 25,$$

The starting point is $x_1 = 2,\ x_2 = 3,\ x_3 = 2$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE MINIMAX OPTIMIZATION WITH NONLINEAR CONSTRAINTS       +
$REM +-------------------------------------------------------------+
```

```
$SET(INPUT)
  X(1)=2.0D0; X(2)=3.0D0; X(3)=2.0D0
  CL(1)=3.0D0; IC(1)=1
  CL(2)=5.6D1; IC(2)=5
  CL(3)=2.5D1; IC(3)=5
$ENDSET
$SET(FMODELA)
  IF(KA.EQ.1)THEN
    FA=X(1)-1.4D1
  ELSEIF(KA.EQ.2) THEN
    FA=X(1)**2-4.0D0*X(1)+X(2)**2-4.0D0*X(2)+X(3)**2-4.0D0*X(3)
  ELSE
    FA=X(1)**2-4.0D0*X(1)-8.0D0
  ENDIF
$ENDSET
$SET(FMODELC)
  IF(KC.EQ.1)THEN
    FC=6.0D0*X(2)+4.0D0*X(3)-X(1)**3
  ELSEIF(KC.EQ.2) THEN
    FC=8.0D0*X(1)+1.4D1*X(2)+7.0D0*X(3)
  ELSE
    FC=X(1)**2+X(2)**2+X(3)**2
  ENDIF
$ENDSET
$MODEL='AM'
$IEXT=-1
$IADA=1
$IADC=1
$NF=3
$NA=3
$NC=3
$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 nonlinear constraints. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For minimax approximation we set $MODEL='AM' and $IEXT=-1.

d) Problem solution (basic screen output):

```
 CLASS = VM - LQ1   UPDATE = B   MODEL = AM   HESF = D     NF =      4
     NIC=  0 NIT=   0 NFV=    1 NFG=    1 F=0.000D+00 C=0.160D+02 G=0.000D+00
     NIC=  0 NIT=   1 NFV=    2 NFG=    2 F=0.276D+03 C=0.280D+03 G=0.140D+02
     NIC=  0 NIT=   2 NFV=    3 NFG=    3 F=0.745D+02 C=0.664D+02 G=0.701D+01
     NIC=  0 NIT=   3 NFV=    4 NFG=    4 F=0.936D+01 C=0.136D+02 G=0.140D+01
     NIC=  0 NIT=   4 NFV=    5 NFG=    5 F=0.217D+01 C=0.236D+01 G=0.526D+00
     NIC=  0 NIT=   5 NFV=    6 NFG=    6 F=-.374D+01 C=0.853D-01 G=0.168D+00
```

```
     NIC=  0 NIT=   6 NFV=    7 NFG=    7 F=-.393D+01 C=0.125D-03 G=0.418D-02
     NIC=  0 NIT=   7 NFV=    8 NFG=    8 F=-.393D+01 C=0.262D-09 G=0.514D-05
   0 NIC=  0 NIT=   8 NFV=    8 NFG=    8 F=-.393D+01 C=0.262D-09 G=0.107D-10
 F  = -0.3934510577D+01
 X  =  0.2548204707D+01  0.4023431118D+00  0.4283079826D+01 -0.3934510577D+01
 TIME= 0:00:00.00
```

7.13 Global optimization

a) Problem description:

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

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

over the set given by the inequalities $-12 \leq x_1 \leq 10$ and $-12 \leq x_2 \leq 10$. The starting point is $x_1 = 0$, $x_2 = 0$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + DENSE UNCONSTRAINED GLOBAL MINIMIZATION                     +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  XL(1)=-1.2D1; XU(1)=1.0D1
  XL(2)=-1.2D1; XU(2)=1.0D1
$ENDSET
$SET(FMODELF)
  FF=((X(1)-3.0D0)*(X(1)+5.0D0))**2+((X(2)-2.0D0)*(X(2)+3.0D0))**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 the bounds defining the investigated region. By using the macrovariable $FMODELF we specify analytically the value of the model function. The gradient of the model function is computed numerically. Since we require to find the global minimum we set $EXTREM='G'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LI1   UPDATE = B   MODEL = FF   HESF = D     NF =      2
   0  NIT=    64  NFV=   709  NEX=   4  F=-0.806D+03

 EXTREM   1 :
 F = -0.8060772623D+03
 X = -0.7329989920D+01 -0.6447506446D+01
```

```
 EXTREM   2 :
 F = -0.3072281498D+03
 X = -0.6228926454D+01  0.4363683102D+01

 EXTREM   3 :
 F = -0.1504539067D+03
 X =  0.3836710556D+01 -0.4317610574D+01

 EXTREM   4 :
 F = -0.5795091449D+02
 X =  0.3368245192D+01  0.2827173202D+01
 TIME= 0:00:00.00
```

7.14 Large-scale nonlinear equations

a) Problem description:

Suppose we have to solve the system of the nonlinear equations

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

where $n = 100$. The starting point is $x_i = -1$ for $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + LARGE-SCALE NONLINEAR EQUATIONS                             +
$REM +-------------------------------------------------------------+
$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
$MODEL='NE'
$JACA='N'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FMODELA we specify analytically the values of functions in the nonlinear equations. For solving nonlinear equations we set $MODEL='NE'. Since the Jacobian matrix is not defined, we set $JACA='N'.

d) Problem solution (basic screen output):

```
 CLASS = TN - GE3   UPDATE = N   MODEL = AQ   HESF = N     NF =    100
     NIT=    0 NFV=    1 F=  205.0000000
     NIT=    1 NFV=    5 F=  5.266293104
     NIT=    2 NFV=    9 F= 0.1672522970E-01
     NIT=    3 NFV=   13 F= 0.1523614858E-05
     NIT=    4 NFV=   19 F= 0.9562244762E-11
     NIT=    5 NFV=   27 F= 0.3893958431E-18
   0 NIT=    5 NFV=   27 NDC=    0  FV BOUND  F= 0.3893958431E-18
 TIME= 0:00:00.00
```

7.15 Large scale unconstrained optimization (sparse Hessian matrix)

a) Problem description:

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

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

where $n = 100$. The starting point is $x_i = -1$ for $1 \leq i \leq n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE UNCONSTRAINED MINIMIZATION                           +
$REM +-------------------------------------------------------------+
$FLOAT W,WF(0:100)
INTEGER IAD_WF(0:100)
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
    J=2*(I-1)+1
    IH(I)=J
    JH(J)=I
    JH(J+1)=I+1
1 CONTINUE
  IH(NF+1)=2*NF
$ENDSET
$SET(FMODELF)
  WF(0)=0.0D0
  DO 2 J=1,NF
    IF(J.EQ.1)THEN
      W=(3.0D0-2.0D0*X(J))*X(J)+1.0D0-X(J+1)
    ELSEIF(J.EQ.NF)THEN
      W=(3.0D0-2.0D0*X(J))*X(J)+1.0D0-X(J-1)
    ELSE
      W=(3.0D0-2.0D0*X(J))*X(J)+1.0D0-X(J-1)-X(J+1)
    ENDIF
    WF(J)=WF(J-1)+W*W
2 CONTINUE
    FF=WF(NF)
$ENDSET
```

```
$IADF=1
$NF=100
$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. By using the macrovariable $FMODELF we specify the value of the model function analytically. The gradient of the model function is computed by automatic differentiation, since $IADF=1. Since values of the objective function are counted in the cycle, we have to use real array WF(0:100) and the corresponding integer array IAD_WF(0:100) (see Section 6.1) Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used.

d) Problem solution (basic screen output):

```
 CLASS = MN - GM3   UPDATE = N   MODEL = FF   HESF = S     NF =    100
     NIT=    0 NFV=    4 NFG=    4  F=  410.0000000     G=0.380D+02
     NIT=    1 NFV=    8 NFG=    8  F=  51.26265782     G=0.123D+02
     NIT=    2 NFV=   12 NFG=   12  F=  6.162928967     G=0.694D+01
     NIT=    3 NFV=   16 NFG=   16  F= 0.2405291876     G=0.141D+01
     NIT=    4 NFV=   20 NFG=   20  F= 0.7598786964E-03 G=0.122D+00
     NIT=    5 NFV=   24 NFG=   24  F= 0.2442230554E-05 G=0.636D-02
     NIT=    6 NFV=   28 NFG=   28  F= 0.3898601055E-07 G=0.822D-03
     NIT=    7 NFV=   32 NFG=   32  F= 0.1062176777E-09 G=0.481D-04
     NIT=    8 NFV=   36 NFG=   36  F= 0.5393508095E-12 G=0.401D-05
     NIT=    9 NFV=   40 NFG=   40  F= 0.3871096527E-14 G=0.271D-06
   0 NIT=    9 NFV=   40 NFG=   40  GRAD TOL  F= 0.3871096527E-14 G=0.271D-06
 TIME= 0:00:00.00
```

7.16 Large-scale unconstrained optimization (sparse Jacobian matrix)

a) Problem description:

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

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

where n=100 and

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

The starting point is $x_i = -1$ for $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE PARTIALLY SEPARABLE UNCONSTRAINED MINIMIZATION       +
$REM +-------------------------------------------------------------+
```

```
$FLOAT WA
$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)
  IF(KA.EQ.1)THEN
    WA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA+1)
  ELSEIF(KA.EQ.NA)THEN
    WA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA-1)
  ELSE
    WA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA-1)-X(KA+1)
  ENDIF
  FA=WA*WA
$ENDSET
$IADA=1
$NF=100
$NA=100
$MODEL='AF'
$JACA='S'
$FMIN='0.0D0'
$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. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum of values minimization we set $MODEL='AF'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LM3   UPDATE = B   MODEL = AF   HESF = S     NF =    100
     NIT=    0 NFV=    1 NFG=    1  F=  410.0000000     G=0.380D+02
     NIT=    1 NFV=    2 NFG=    2  F=  56.19847311     G=0.887D+01
```

```
     NIT=    2 NFV=    3 NFG=    3  F=  33.21669961     G=0.563D+01
     NIT=    3 NFV=    5 NFG=    5  F=  8.034550334     G=0.672D+01
     NIT=    4 NFV=    7 NFG=    7  F=  2.498266945     G=0.753D+01
     NIT=    5 NFV=    9 NFG=    9  F=  1.126771725     G=0.617D+01
     NIT=    6 NFV=   10 NFG=   10  F= 0.2724480160E-01 G=0.730D+00
     NIT=    7 NFV=   11 NFG=   11  F= 0.1670056108E-03 G=0.647D-01
     NIT=    8 NFV=   12 NFG=   12  F= 0.1033932683E-05 G=0.365D-02
     NIT=    9 NFV=   13 NFG=   13  F= 0.1781676012E-07 G=0.834D-03
     NIT=   10 NFV=   14 NFG=   14  F= 0.2260632676E-09 G=0.981D-04
     NIT=   11 NFV=   15 NFG=   15  F= 0.2888717918E-11 G=0.995D-05
     NIT=   12 NFV=   16 NFG=   16  F= 0.1281762413E-12 G=0.216D-05
     NIT=   13 NFV=   17 NFG=   17  F= 0.2194981848E-15 G=0.782D-07
   0 NIT=   13 NFV=   17 NFG=   17  GRAD TOL  F= 0.2194981848E-15 G=0.782D-07
 TIME= 0:00:00.00
```

7.17 Large scale unconstrained least squares optimization

a) Problem description:

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

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

where $n = 100$ and

$$\begin{aligned}
f_i^A(x) &= x_1 - 1 && , \quad i = 1, \\
f_i^A(x) &= 10(x_i^2 - x_{i+1}), && , \quad k > 1, \quad \mathrm{mod}(k,2) = 0, \\
f_i^A(x) &= 2\exp(-(x_i - x_{i+1})^2) + \exp(-2(x_{i+1} - x_{i+2})^2) && , \quad k > 1, \quad \mathrm{mod}(k,2) = 1,
\end{aligned}$$

The starting point is $x_i = -1.2$ for $\mathrm{mod}(i,2) = 1$ and $x_i = 1.0$ for $\mathrm{mod}(i,2) = 0$, where $1 \leq i \leq n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE UNCONSTRAINED LEAST SQUARES                          +
$REM +-------------------------------------------------------------+
$FLOAT W1,W2
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)= 1.0D0
    ENDIF
1 CONTINUE
  NA=2*(NF-1)
  MA=2
  IAG(1)=1
  IAG(2)=2
  JAG(1)=1
  DO 2 KA=1,NA-1
    I=(KA+1)/2
```

```
    IF (MOD(KA,2).EQ.1) THEN
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
    ELSE
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
      JAG(MA)=I+2
      MA=MA+1
    ENDIF
    IAG(KA+2)=MA
2 CONTINUE
  MA=MA-1
$ENDSET
$SET(FMODELA)
  I=KA/2
  IF(KA.EQ.1)THEN
    FA=X(KA)-1.0D0
  ELSEIF(MOD(KA,2).EQ.0)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=2.0D0*EXP(-(X(I)-X(I+1))**2)+EXP(-2.0D0*(X(I+1)-X(I+2))**2)
  ENDIF
$ENDSET
$IADA=1
$NF=100
$NA=198
$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. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum-of-squares minimization we set $MODEL='AQ'.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM3   UPDATE = M   MODEL = AQ   HESF = S     NF =    100
     NIT=    0 NFV=    1 NFG=    1  F=  12344.43235     G=0.396D+03
     NIT=    1 NFV=    3 NFG=    3  F=  1211.833625     G=0.149D+03
     NIT=    2 NFV=    5 NFG=    5  F=  556.3758005     G=0.663D+02
     NIT=    3 NFV=    7 NFG=    7  F=  466.7731338     G=0.324D+02
     NIT=    4 NFV=    9 NFG=    9  F=  447.6914295     G=0.127D+02
     NIT=    5 NFV=   11 NFG=   11  F=  443.4171082     G=0.480D+01
     NIT=    6 NFV=   13 NFG=   13  F=  442.2501148     G=0.182D+02
```

```
    NIT=    7 NFV=   15 NFG=   15  F=  439.5246294     G=0.283D+01
    NIT=    8 NFV=   18 NFG=   18  F=  438.8538272     G=0.285D+01
    NIT=    9 NFV=   20 NFG=   20  F=  438.3971433     G=0.325D+01
    NIT=   10 NFV=   22 NFG=   22  F=  438.1103835     G=0.101D+01
    ------------------------------------------------------------
    NIT=   32 NFV=   57 NFG=   57  F=  436.9702388     G=0.342D-03
    NIT=   33 NFV=   59 NFG=   59  F=  436.9702388     G=0.161D-03
    NIT=   34 NFV=   60 NFG=   60  F=  436.9702388     G=0.115D-03
    NIT=   35 NFV=   62 NFG=   62  F=  436.9702388     G=0.499D-04
    NIT=   36 NFV=   63 NFG=   63  F=  436.9702388     G=0.552D-04
    NIT=   37 NFV=   64 NFG=   64  F=  436.9702388     G=0.140D-04
    NIT=   38 NFV=   65 NFG=   65  F=  436.9702388     G=0.116D-04
    NIT=   39 NFV=   67 NFG=   67  F=  436.9702388     G=0.745D-05
    NIT=   40 NFV=   68 NFG=   68  F=  436.9702388     G=0.380D-05
    NIT=   41 NFV=   69 NFG=   69  F=  436.9702388     G=0.131D-05
    NIT=   42 NFV=   70 NFG=   70  F=  436.9702388     G=0.569D-06
  0 NIT=   42 NFV=   70 NFG=   70  GRAD TOL  F=  436.9702388     G=0.569D-06
 TIME= 0:00:00.03
```

7.18 Large-scale unconstrained l_1 optimization

a) Problem description:

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

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

where $n = 100$ and

$$\begin{aligned}
f_i^A(x) &= x_1 - 1 && , \quad i = 1, \\
f_i^A(x) &= 10(x_i^2 - x_{i+1}), && , \quad k > 1, \quad \mathrm{mod}(k,2) = 0, \\
f_i^A(x) &= 2\exp(-(x_i - x_{i+1})^2) + \exp(-2(x_{i+1} - x_{i+2})^2) && , \quad k > 1, \quad \mathrm{mod}(k,2) = 1,
\end{aligned}$$

The starting point is $x_i = -1.2$ for $\mathrm{mod}(i,2) = 1$ and $x_i = 1.0$ for $\mathrm{mod}(i,2) = 0$, where $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE UNCONSTRAINED L-1 OPTIMIZATION                       +
$REM +-------------------------------------------------------------+
$FLOAT W1,W2
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)= 1.0D0
    ENDIF
1 CONTINUE
  NA=2*(NF-1)
  MA=2
  IAG(1)=1
  IAG(2)=2
```

```
  JAG(1)=1
  DO 2 KA=1,NA-1
    I=(KA+1)/2
    IF (MOD(KA,2).EQ.1) THEN
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
    ELSE
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
      JAG(MA)=I+2
      MA=MA+1
    ENDIF
    IAG(KA+2)=MA
2 CONTINUE
  MA=MA-1
$ENDSET
$SET(FMODELA)
  I=KA/2
  IF(KA.EQ.1)THEN
    FA=X(KA)-1.0D0
  ELSEIF(MOD(KA,2).EQ.0)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=2.0D0*EXP(-(X(I)-X(I+1))**2)+EXP(-2.0D0*(X(I+1)-X(I+2))**2)
  ENDIF
$ENDSET
$IADA=1
$NF=100
$NA=198
$MODEL='AA'
$JACA='S'
$XMAX='3.0D0'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the Jacobian matrix and the maximum step-size $XMAX. The sparse Jacobian matrix, indicated by the statement $JACA='S', is tridiagonal. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum-of-absolute-values minimization we set $MODEL='AA'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LG1   UPDATE = N   MODEL = AA   HESF = S     NF =    100
     NIT=    0 NFV=    3 NFG=    3  F=  1157.710087     G=0.287D+02
     NIT=    1 NFV=    6 NFG=    6  F=  666.2461362     G=0.141D+02
```

```
    NIT=    2 NFV=   10 NFG=   10  F=  410.6587423     G=0.140D+02
    NIT=    3 NFV=   13 NFG=   13  F=  309.3768933     G=0.680D+01
    NIT=    4 NFV=   17 NFG=   17  F=  272.4065551     G=0.401D+01
    NIT=    5 NFV=   21 NFG=   21  F=  266.4111131     G=0.278D+01
    NIT=    6 NFV=   25 NFG=   25  F=  261.4510889     G=0.343D+01
    NIT=    7 NFV=   28 NFG=   28  F=  261.1807945     G=0.417D+01
    NIT=    8 NFV=   32 NFG=   32  F=  261.0895640     G=0.688D+01
    NIT=    9 NFV=   38 NFG=   38  F=  260.9593145     G=0.458D+01
    NIT=   10 NFV=   42 NFG=   42  F=  260.7236285     G=0.226D+01
    ------------------------------------------------------------
    NIT=   44 NFV=  231 NFG=  231  F=  293.3677930     G=0.265D+02
    NIT=   45 NFV=  240 NFG=  240  F=  293.3677908     G=0.258D+02
    NIT=   46 NFV=  248 NFG=  248  F=  293.3677902     G=0.136D+02
    NIT=   47 NFV=  256 NFG=  256  F=  293.3677899     G=0.131D+02
    NIT=   48 NFV=  262 NFG=  262  F=  293.3677899     G=0.839D+01
    NIT=   49 NFV=  268 NFG=  268  F=  293.3677898     G=0.648D+01
    NIT=   50 NFV=  271 NFG=  271  F=  293.3677898     G=0.164D+01
    NIT=   51 NFV=  275 NFG=  275  F=  293.3677898     G=0.402D-01
    NIT=   52 NFV=  278 NFG=  278  F=  293.3677898     G=0.381D-04
    NIT=   53 NFV=  291 NFG=  291  F=  293.3677898     G=0.281D-05
    NIT=   54 NFV=  303 NFG=  303  F=  293.3677898     G=0.841D-06
  0 NIT=   54 NFV=  303 NFG=  303  GRAD TOL  F=  293.3677353     G=0.841D-06
 TIME= 0:00:00.17
```

7.19 Large-scale unconstrained l_∞ (minimax) optimization

a) Problem description:

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

$$F(x) = \max_{1 \le i \le n} \left| f_i^A(x) \right|$$

where $n = 100$ and

$$\begin{aligned}
f_i^A(x) &= x_1 - 1 && , \quad i = 1, \\
f_i^A(x) &= 10(x_i^2 - x_{i+1}), && , \quad k > 1, \quad \mathrm{mod}(k,2) = 0, \\
f_i^A(x) &= 2\exp(-(x_i - x_{i+1})^2) + \exp(-2(x_{i+1} - x_{i+2})^2) , && \quad k > 1, \quad \mathrm{mod}(k,2) = 1,
\end{aligned}$$

The starting point is $x_i = -1.2$ for $\mathrm{mod}(i,2) = 1$ and $x_i = 1.0$ for $\mathrm{mod}(i,2) = 0$, where $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE UNCONSTRAINED L1-INFINITY (MINIMAX) OPTIMIZATION      +
$REM +-------------------------------------------------------------+
$FLOAT W1,W2
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)= 1.0D0
    ENDIF
```

```
1 CONTINUE
  NA=2*(NF-1)
  MA=2
  IAG(1)=1
  IAG(2)=2
  JAG(1)=1
  DO 2 KA=1,NA-1
    I=(KA+1)/2
    IF (MOD(KA,2).EQ.1) THEN
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
    ELSE
      JAG(MA)=I
      MA=MA+1
      JAG(MA)=I+1
      MA=MA+1
      JAG(MA)=I+2
      MA=MA+1
    ENDIF
    IAG(KA+2)=MA
2 CONTINUE
  MA=MA-1
$ENDSET
$SET(FMODELA)
  I=KA/2
  IF(KA.EQ.1)THEN
    FA=X(KA)-1.0D0
  ELSEIF(MOD(KA,2).EQ.0)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=2.0D0*EXP(-(X(I)-X(I+1))**2)+EXP(-2.0D0*(X(I+1)-X(I+2))**2)
  ENDIF
$ENDSET
$IADA=1
$NF=100
$NA=198
$MODEL='AM'
$JACA='S'
$CLASS='VM'
$XMAX='2.3D0'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the Jacobian matrix and the maximum step-size $XMAX. The sparse Jacobian matrix, indicated by the statement $JACA='S', is tridiagonal. Therefore we set $MA=300. Since we use the sparse Hessian matrix, we set $HESF='S' and specify the number of its nonzero elements $M=3000 (this is a reasonable upper bound). By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation,

since $IADA=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum-of-absolute-values minimization we set $MODEL='AA'.

d) Problem solution (basic screen output):

```
 CLASS = VM - LG1   UPDATE = B   MODEL = AM   HESF = S     NF =    100
     NIT=    0 NFV=    1 NFG=    1  F= -1972.482481     G=0.506D-02
     NIT=    1 NFV=    2 NFG=    2  F=  6.388665237     G=0.334D+00
     NIT=    2 NFV=    3 NFG=    3  F=  3.933293443     G=0.430D+00
     NIT=    3 NFV=    4 NFG=    4  F=  2.058362700     G=0.614D+00
     NIT=    4 NFV=    5 NFG=    5  F=  1.155433487     G=0.442D-01
     NIT=    5 NFV=    6 NFG=    6  F=  1.464701872     G=0.140D-01
     NIT=    6 NFV=    7 NFG=    7  F=  1.749590098     G=0.426D-02
     NIT=    7 NFV=    8 NFG=    8  F=  2.658151338     G=0.153D+01
     NIT=    8 NFV=   11 NFG=   11  F=  2.578627491     G=0.741D+01
     NIT=    9 NFV=   14 NFG=   14  F=  2.484660456     G=0.520D-01
     NIT=   10 NFV=   16 NFG=   16  F=  2.472074894     G=0.955D-01
     ------------------------------------------------------------
     NIT=   48 NFV=  125 NFG=  125  F=  2.354874070     G=0.644D+01
     NIT=   49 NFV=  128 NFG=  128  F=  2.354874042     G=0.199D+00
     NIT=   50 NFV=  129 NFG=  129  F=  2.354874042     G=0.832D-01
     NIT=   51 NFV=  130 NFG=  130  F=  2.354874041     G=0.365D-01
     NIT=   52 NFV=  131 NFG=  131  F=  2.354874041     G=0.140D-01
     NIT=   53 NFV=  132 NFG=  132  F=  2.354874041     G=0.334D-02
     NIT=   54 NFV=  133 NFG=  133  F=  2.354874041     G=0.234D-03
     NIT=   55 NFV=  135 NFG=  135  F=  2.354874041     G=0.229D-03
     NIT=   56 NFV=  137 NFG=  137  F=  2.354874041     G=0.234D-03
     NIT=   57 NFV=  138 NFG=  138  F=  2.354874041     G=0.128D-05
     NIT=   58 NFV=  141 NFG=  141  F=  2.354874041     G=0.644D-06
   0 NIT=   58 NFV=  141 NFG=  141  GRAD TOL  F=  2.354873722     G=0.644D-06
 TIME= 0:00:00.09
```

7.20 Large-scale unconstrained nonsmooth optimization

a) Problem description:

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

$$F(x) = \sum_{i=1}^{n-1} \max\left(x_i^2 + (x_{i+1}-1)^2 + x_{i+1} - 1,\ -x_i^2 - (x_{i+1}-1)^2 + x_{i+1} - 1\right),$$

where $n = 100$. The starting point is $x_i = -1.5$ for $\mathrm{mod}(i,2) = 1$ and $x_i = 2.0$ for $\mathrm{mod}(i,2) = 0$, where $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + LARGE SCALE UNCONSTRAINED NONSMOOTH OPTIMIZATION              +
$REM +---------------------------------------------------------------+
$FLOAT W1,W2
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.5D0
```

```
    ELSE
      X(I)= 2.0D0
    ENDIF
1 CONTINUE
$ENDSET
$SET(FGMODELF)
  FF=0.0D0
  GF(1)=0.0D0
  DO 2 I=1,NF-1
    W1= X(I)*X(I)+(X(I+1)-1.0D0)*(X(I+1)-1.0D0)+X(I+1)-1.0D0
    W2=-X(I)*X(I)-(X(I+1)-1.0D0)*(X(I+1)-1.0D0)+X(I+1)+1.0D0
    IF (W1.GE.W2) THEN
      FF=FF+W1
      GF(I)=GF(I)+2.0D0*X(I)
      GF(I+1)= 2.0D0*(X(I+1)-1.0D0)+1.0D0
    ELSE
      FF=FF+W2
      GF(I)=GF(I)-2.0D0*X(I)
      GF(I+1)=-2.0D0*(X(I+1)-1.0D0)+1.0D0
    END IF
2 CONTINUE
$ENDSET
$NF=100
$HESF='N'
$KSF=3
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the objective function. For nonsmooth optimization we set $KSF=3. Since the pattern of the Hessian matrix is not given, we set $HESF='N'.

d) Problem solution (basic screen output):

```
 CLASS = BL - LI1   UPDATE = N   MODEL = FF   HESF = N     NF =    100
     NIT=    0 NFV=    1 NFG=    1  F=  592.2500000     G=0.700D+01
     NIT=    1 NFV=    2 NFG=    2  F=  592.2500000     G=0.700D+01
     NIT=    2 NFV=    3 NFG=    3  F=  61.35842142     G=0.300D+01
     NIT=    3 NFV=    4 NFG=    4  F=  61.35842142     G=0.300D+01
     NIT=    4 NFV=    5 NFG=    5  F=  8.817561091     G=0.300D+01
     NIT=    5 NFV=    6 NFG=    6  F=  4.412558758     G=0.238D+01
     NIT=    6 NFV=    7 NFG=    7  F=  3.624278568     G=0.243D+01
     NIT=    7 NFV=    8 NFG=    8  F=  2.725409817     G=0.253D+01
     NIT=    8 NFV=    9 NFG=    9  F= 0.8653137918     G=0.288D+01
     NIT=    9 NFV=   10 NFG=   10  F= 0.8653137918     G=0.300D+01
     NIT=   10 NFV=   11 NFG=   11  F= 0.8653137918     G=0.224D+01
     ------------------------------------------------------------
     NIT=  211 NFV=  212 NFG=  212  F= 0.7717072027E-06 G=0.300D+01
     NIT=  212 NFV=  213 NFG=  213  F= 0.7717072027E-06 G=0.300D+01
     NIT=  213 NFV=  214 NFG=  214  F= 0.7717072027E-06 G=0.300D+01
     NIT=  214 NFV=  215 NFG=  215  F= 0.7717072027E-06 G=0.300D+01
```

```
     NIT=  215 NFV=  216 NFG=  216  F= 0.7717072027E-06 G=0.300D+01
     NIT=  216 NFV=  217 NFG=  217  F= 0.7717072027E-06 G=0.300D+01
     NIT=  217 NFV=  218 NFG=  218  F= 0.7717072027E-06 G=0.300D+01
     NIT=  218 NFV=  219 NFG=  219  F= 0.7717072027E-06 G=0.300D+01
   0 NIT=  219 NFV=  219 NFG=  219  GRAD TOL  F= 0.7717072027E-06 G=0.434D-06
 TIME= 0:00:00.02
```

7.21 Sparse linear programming

a) Problem description:

Suppose we have to find the global maximum of the linear function

$$F(x) = \sum_{i=1}^{n} (-1)^i x_i$$

with simple bounds $-20 \leq x_i \leq 20,\ 1 \leq i \leq n$, and linear constraints

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

where $n = 20$ and $n_C = 18$. The starting point is not given. The maximum value of the linear objective function is $F = 7.0$

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE LINEAR PROGRAMMING                                   +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  DO 1 I=1,NF
    IX(I)=3; XL(I)=-2.0D1; XU(I)=2.0D1
    GF(I)=DBLE((-1)**I)
1 CONTINUE
  DO 2 KC=1,NC
    IC(KC)=5; CL(KC)=DBLE(KC)
    $SETCG(KC,KC,-1.0D0)
    $SETCG(KC,KC+1, 1.0D0)
    $SETCG(KC,KC+2,-1.0D0)
2 CONTINUE
$ENDSET
$IEXT=1
$NF=20
$NX=20
$NC=18
$NCL=18
$MODEL='FL'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds for variables and the sparsity pattern with numerical values of the constraint Jacobian matrix. We use the procedure UKMCI1. The sparse Jacobian

matrix, indicated by the statement $JACC='S', is tridiagonal. The option $MODEL='FL' indicates the linear programming problem.

d) Problem solution (basic screen output):

```
 CLASS = LP - LN1   UPDATE = N   MODEL = FL   HESF = N     NF =     20
 NUMITR=    1 INEW=   20 IOLD=   15 KINP=   0 IU=   48  F=0.980D+04
 NUMITR=    2 INEW=   19 IOLD=   20 KINP=   0 IU=   49  F=0.208D+04
 NUMITR=    3 INEW=    0 IOLD=   20 KINP=   0 IU=   49  F=0.000D+00
 NUMITR=    3 NEL=     3 NREF=    1 KINP=   0 IU=   49  F=0.000D+00  ITERL= 1
 NUMITR=    1 INEW=   15 IOLD=   19 KINP=   0 IU=   49  F=0.900D+01
 NUMITR=    2 INEW=   20 IOLD=   18 KINP=   0 IU=   48  F=0.700D+01
 NUMITR=    3 INEW=    0 IOLD=   18 KINP=   0 IU=   48  F=0.700D+01
 NUMITR=    3 NEL=     3 NREF=    1 KINP=   0 IU=   48  F=0.700D+01  ITERL= 2
   0 NIT=   6 NFV=   2 NFG=    0 OPTIMUM  F=  7.00000     C=0.0D+00 G=0.0D+00
 FF = -0.7000000000D+01
 X  = -0.2000000000D+01  0.0000000000D+00  0.1000000000D+01 -0.1000000000D+01
      -0.5000000000D+01 -0.8000000000D+01 -0.8000000000D+01 -0.6000000000D+01
      -0.5000000000D+01 -0.7000000000D+01 -0.1100000000D+02 -0.1400000000D+02
      -0.1400000000D+02 -0.1200000000D+02 -0.1100000000D+02 -0.1300000000D+02
      -0.1700000000D+02 -0.2000000000D+02 -0.2000000000D+02 -0.1800000000D+02
 TIME= 0:00:00.00
```

7.22 Sparse quadratic programming

a) Problem description:

Suppose we have to find the global minimum of the quadratic function

$$F(x) = \frac{1}{2}(2x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 2x_1x_3 + 2x_3x_4) - x_1 - 3x_2 + x_3 - x_4$$

with simple bounds $x_i \geq 0$ for $1 \leq i \leq 4$ and linear constraints

$$\begin{aligned} x_1 + 2x_2 + x_3 + x_4 &\leq 5, \\ 3x_1 + x_2 + 2x_3 - x_4 &\leq 4, \\ 2x_2 + 8x_3 &\geq 3. \end{aligned}$$

The starting point is $x_i = 1/2$ for $1 \leq i \leq 4$. The minimum value of the quadratic objective function is $F = -4.681818$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SPARSE QUADRATIC PROGRAMMING                                  +
$REM +---------------------------------------------------------------+
$SET(INPUT)
  DO 1 I=1,NF
  X(I)= 0.5D0; XL(I)=0.0D0; IX(I)=1
1 CONTINUE
  GF(1)=-1.0D0; GF(2)=-3.0D0; GF(3)= 1.0D0; GF(4)=-1.0D0
  IH(1)= 1; IH(2)= 3; IH(3)= 4; IH(4)= 6; IH( 5)= 7
  JH(1)= 1; JH(2)= 3; JH(3)= 2; JH(4)= 3; JH( 5)= 4; JH(6)= 4
  HF(1)= 2.0D0; HF(2)=-1.0D0; HF(3)= 1.0D0
```

```
  HF(4)= 2.0D0; HF(5)= 1.0D0; HF(6)= 1.0D0
  IC(1)=2; IC(2)=2; IC(3)=1
  CU(1)= 5.0D0; CU(2)= 4.0D0; CL(3)= 3.0D0
  ICG(1)=1; ICG(2)=5; ICG(3)=9; ICG(4)=11
  JCG(1)=1; JCG(2)=2; JCG(3)=3; JCG(4)=4; JCG( 5)=1
  JCG(6)=2; JCG(7)=3; JCG(8)=4; JCG(9)=2; JCG(10)=3
  CG(1)= 1.0D0; CG(2)= 2.0D0; CG(3)= 1.0D0; CG(4)= 1.0D0; CG( 5)= 3.0D0
  CG(6)= 1.0D0; CG(7)= 2.0D0; CG(8)=-1.0D0; CG(9)= 2.0D0; CG(10)= 8.0D0
$ENDSET
$NF=4
$NX=4
$NC=3
$NCL=3
$MODEL='FQ'
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds for variables, the sparsity pattern with numerical values of the model Hessian matrix, and the sparsity pattern with numerical values of the constraint Jacobian matrix. The sparse Hessian matrix is indicated by the statement $HESF='S'. The sparse Jacobian matrix is indicated by the statement $JACC='S'. The option $MODEL='FQ' indicates the quadratic programming problem.

d) Problem solution (basic screen output):

```
 CLASS = QP - LN2   UPDATE = N   MODEL = FQ   HESF = S     NF =      4
 MODE =   1   NRED =   0   N =   4   IOLD =   0   INEW =   0
 MODE =   1   NRED =   1   N =   3   IOLD =   0   INEW =   3   ADDITION
 MODE =   1   NRED =   2   N =   2   IOLD =   0   INEW =  -3   ADDITION
 MODE =   1   NRED =   3   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   4   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   4   N =   3   IOLD =   1   INEW =   0   DELETION
 MODE =   1   NRED =   5   N =   2   IOLD =   0   INEW =   1   ADDITION
 MODE =   1   NRED =   6   N =   2   IOLD =   0   INEW =   0
 MODE =   1   NRED =   7   N =   2   IOLD =   0   INEW =   0
   0 NIT=   7 NFV=   2 NFG=    0 OPTIMUM  F= -4.68182     C=0.0D+00 G=0.0D+00
 FF = -0.4681818182D+01
 X  =  0.2727272727D+00  0.2090909091D+01  0.0000000000D+00  0.5454545455D+00
 TIME= 0:00:00.00
```

7.23 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.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE MINIMIZATION WITH LINEAR CONSTRAINTS                  +
```

```
$REM +-------------------------------------------------------------+
$FLOAT WI,WJ

$SET(INPUT)
  DO 1 I=1,NF
    X(I)=10.0D0; XL(I)=0.0D0; XU(I)=5.0D0; IX(I)=3
1 CONTINUE
  IH( 1)= 1; IH( 2)= 6; IH( 3)=10; IH( 4)=15; IH( 5)=19
  IH( 6)=24; IH( 7)=27; IH( 8)=30; IH( 9)=33; IH(10)=36
  IH(11)=38; IH(12)=40; IH(13)=42; IH(14)=44; IH(15)=45
  IH(16)=46; IH(17)=47;
  JH( 1)= 1; JH( 2)= 4; JH( 3)= 7; JH( 4)= 8; JH( 5)=16
  JH( 6)= 2; JH( 7)= 3; JH( 8)= 7; JH( 9)=10;
  JH(10)= 3; JH(11)= 7; JH(12)= 9; JH(13)=10; JH(14)=14
  JH(15)= 4; JH(16)= 7; JH(17)=11; JH(18)=15;
  JH(19)= 5; JH(20)= 6; JH(21)=10; JH(22)=12; JH(23)=16
  JH(24)= 6; JH(25)= 8; JH(26)=15;
  JH(27)= 7; JH(28)=11; JH(29)=13;
  JH(30)= 8; JH(31)=10; JH(32)=15;
  JH(33)= 9; JH(34)=12; JH(35)=16;
  JH(36)=10; JH(37)=14;
  JH(38)=11; JH(39)=13;
  JH(40)=12; JH(41)=14;
  JH(42)=13; JH(43)=14;
  JH(44)=14;
  JH(45)=15;
  JH(46)=16;
  DO 2 I=1,NC
    IC(I)=5
2 CONTINUE
  CL(1)= 2.5D0; CL(2)= 1.1D0; CL(3)=-3.1D0; CL(4)=-3.5D0
  CL(5)= 1.3D0; CL(6)= 2.1D0; CL(7)= 2.3D0; CL(8)=-1.5D0
  $SETCG(1, 1, 0.22D0); $SETCG(1, 2, 0.20D0); $SETCG(1, 3, 0.19D0)
  $SETCG(1, 4, 0.25D0); $SETCG(1, 5, 0.15D0); $SETCG(1, 6, 0.11D0)
  $SETCG(1, 7, 0.12D0); $SETCG(1, 8, 0.13D0); $SETCG(1, 9, 1.00D0)
  $SETCG(2, 1,-1.46D0); $SETCG(2, 3,-1.30D0); $SETCG(2, 4, 1.82D0)
  $SETCG(2, 5,-1.15D0); $SETCG(2, 7, 0.80D0); $SETCG(2,10, 1.00D0)
  $SETCG(3, 1, 1.29D0); $SETCG(3, 2,-0.89D0); $SETCG(3, 5,-1.16D0)
  $SETCG(3, 6,-0.96D0); $SETCG(3, 8,-0.49D0); $SETCG(3,11, 1.00D0)
  $SETCG(4, 1,-1.10D0); $SETCG(4, 2,-1.06D0); $SETCG(4, 3, 0.95D0)
  $SETCG(4, 4,-0.54D0); $SETCG(4, 6,-1.78D0); $SETCG(4, 7,-0.41D0)
  $SETCG(4,12, 1.00D0); $SETCG(5, 4,-1.43D0); $SETCG(5, 5, 1.51D0)
  $SETCG(5, 6, 0.59D0); $SETCG(5, 7,-0.33D0); $SETCG(5, 8,-0.43D0)
  $SETCG(5,13, 1.00D0); $SETCG(6, 2,-1.72D0); $SETCG(6, 3,-0.33D0)
  $SETCG(6, 5, 1.62D0); $SETCG(6, 6, 1.24D0); $SETCG(6, 7, 0.21D0)
  $SETCG(6, 8,-0.26D0); $SETCG(6,14, 1.00D0); $SETCG(7, 1, 1.12D0)
  $SETCG(7, 4, 0.31D0); $SETCG(7, 7, 1.12D0); $SETCG(7, 9,-0.36D0)
  $SETCG(7,15, 1.00D0); $SETCG(8, 2, 0.45D0); $SETCG(8, 3, 0.26D0)
  $SETCG(8, 4,-1.10D0); $SETCG(8, 5, 0.58D0); $SETCG(8, 7,-1.03D0)
  $SETCG(8, 8, 0.10D0); $SETCG(8,16, 1.00D0)
$ENDSET
$SET(FGMODELF)
```

```
  FF=0.0D0
  DO 3 I=1,NF
    GF(I)=0.0D0
3 CONTINUE
  DO 5 I=1,NF
    WI=X(I)*(X(I)+1.0D0)+1.0D0
    K1=IH(I)
    K2=IH(I+1)-1
    DO 4 K=K1,K2
      J=JH(K)
      WJ=X(J)*(X(J)+1.0D0)+1.0D0
      FF=FF+WI*WJ
      GF(I)=GF(I)+(2.0D0*X(I)+1.0D0)*WJ
      GF(J)=GF(J)+WI*(2.0D0*X(J)+1.0D0)
  4 CONTINUE
5 CONTINUE
$ENDSET
$NF=16
$NX=16
$NC=8
$NCL=8
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the bounds for variables, the sparsity pattern with numerical values of the model Hessian matrix, and the sparsity pattern with numerical values of the constraint Jacobian matrix. We use the procedure UKMCI1. The sparse Hessian matrix is indicated by the statement $HESF='S'. The sparse Jacobian matrix is indicated by the statement $JACC='S'. The option $MODEL='FF' indicates a general objective function. By using the macrovariable $FGMODELF we specify analytically the value and the gradient of the model function.

d) Problem solution (basic screen output):

```
 CLASS = VM - LM3   UPDATE = M   MODEL = FF   HESF = S     NF =     16
     NIT=    0 NFV=    2 NFG=    2  F=  4528.573861     G=0.100D+01
     NIT=    1 NFV=    3 NFG=    3  F=  325.0955478     G=0.000D+00
     NIT=    2 NFV=    4 NFG=    4  F=  252.2712344     G=0.000D+00
     NIT=    3 NFV=    5 NFG=    5  F=  246.0828245     G=0.185D+02
     NIT=    4 NFV=    6 NFG=    6  F=  245.0236003     G=0.887D+01
     NIT=    5 NFV=    7 NFG=    7  F=  244.9100700     G=0.161D+01
     NIT=    6 NFV=    8 NFG=    8  F=  244.8997060     G=0.351D-01
     NIT=    7 NFV=    9 NFG=    9  F=  244.8996978     G=0.130D-01
     NIT=    8 NFV=   10 NFG=   10  F=  244.8996975     G=0.124D-02
     NIT=    9 NFV=   11 NFG=   11  F=  244.8996975     G=0.159D-03
     NIT=   10 NFV=   12 NFG=   12  F=  244.8996975     G=0.322D-06
   0 NIT=   10 NFV=   12 NFG=   12  GRAD TOL  F=  244.8996975     G=0.322D-06
 FF =  0.2448996975D+03
 X  =  0.3984735060D-01  0.7919831554D+00  0.2028703315D+00  0.8443579157D+00
       0.1269906451D+01  0.9347387094D+00  0.1681961969D+01  0.1553008761D+00
       0.1567870334D+01  0.0000000000D+00  0.0000000000D+00  0.0000000000D+00
```

```
      0.6602040658D+00  0.0000000000D+00  0.6742559281D+00  0.0000000000D+00
 TIME= 0:00:00.00
```

7.24 Large-scale optimization with nonlinear equality constraints

a) Problem description:

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

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

where $n = 100$ and

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

over the set given by the nonlinear equality constraints

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

The starting point is $x_i = -1, \ 1 \le i \le n$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SPARSE MINIMIZATION WITH NONLINEAR EQUALITY CONSTRAINTS       +
$REM +---------------------------------------------------------------+
$FLOAT WA,WB
$SET(INPUT)
  DO 1 I=1,NF
  X(I)=-1.0D0
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)=0.0D0
4 CONTINUE
$ENDSET
$SET(FMODELF)
  FF=0.0D0
  DO 5 J=1,NF
  WA=(3.0D0-2.0D0*X(J))*X(J)+1.0D0
  IF (J.GT. 1) WA=WA-X(J-1)
  IF (J.LT.NF) WA=WA-X(J+1)
  FF=FF+WA**2
5 CONTINUE
$ENDSET
$SET(GMODELF)
  DO 6 J=1,NF
  GF(J)=0.0D0
6 CONTINUE
  DO 7 J=1,NF
  WA=(3.0D0-2.0D0*X(J))*X(J)+1.0D0
  IF (J.GT. 1) WA=WA-X(J-1)
  IF (J.LT.NF) WA=WA-X(J+1)
  WB=2.0D0*WA
  GF(J)=GF(J)+WB*(3.0D0-4.0D0*X(J))
  IF (J.GT. 1) GF(J-1)=GF(J-1)-WB
  IF (J.LT.NF) GF(J+1)=GF(J+1)-WB
7 CONTINUE
$ENDSET
$SET(FMODELC)
  K=KC+2
  FC=8.0D0*X(K)*(X(K)**2-X(K-1))-2.0D0*(1.0D0-X(K))&
+4.0D0*(X(K)-X(K+1)**2)+X(K-1)**2-X(K-2)+X(K+1)-X(K+2)**2
$ENDSET
$SET(GMODELC)
  K=KC+2
  GC(K-2)=-1.0D0
  GC(K-1)=-8.0D0*X(K)+2.0D0*X(K-1)
  GC(K)=2.4D1*X(K)**2-8.0D0*X(K-1)+6.0D0
  GC(K+1)=-8.0D0*X(K+1)+1.0D0
  GC(K+2)=-2.0D0*X(K+2)
$ENDSET
$NF=100
```

```
$NC=96
$NCE=NC
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Hessian matrix, the sparsity pattern of the constraint Jacobian matrix, and the constraint specifications. The sparse Hessian matrix, indicated by the statement $HESF='S', is tridiagonal. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. The statement $NCE=NC specifies that all constraints are equalities (thus the special method for equality constraints can be used). By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. By using the macrovariable $FMODELC we specify analytically the values of the constraint functions. By using the macrovariable $GMODELC we specify analytically the gradients of the constraint functions.

d) Problem solution (basic screen output):

```
 CLASS = MN - LK3   UPDATE = N   MODEL = FF   HESF = S     NF =    100
     NIC=  0 NIT=   0 NFV=    1 NFG=   10 F=0.410D+03 C=0.280D+02 G=0.380D+02
     NIC=  0 NIT=   1 NFV=    2 NFG=   20 F=0.374D+04 C=0.880D+01 G=0.126D+02
     NIC=  0 NIT=   2 NFV=    3 NFG=   30 F=0.587D+03 C=0.285D+01 G=0.489D+01
     NIC=  0 NIT=   3 NFV=    4 NFG=   40 F=0.269D+03 C=0.127D+01 G=0.640D+01
     NIC=  0 NIT=   4 NFV=    5 NFG=   50 F=0.966D+02 C=0.851D+00 G=0.890D+01
     NIC=  0 NIT=   5 NFV=    9 NFG=   60 F=-.557D+02 C=0.731D+00 G=0.776D+01
     NIC=  0 NIT=   6 NFV=   11 NFG=   70 F=0.295D+01 C=0.311D+00 G=0.385D+01
     NIC=  0 NIT=   7 NFV=   12 NFG=   80 F=0.533D+01 C=0.490D-01 G=0.850D+00
     NIC=  0 NIT=   8 NFV=   13 NFG=   90 F=0.530D+01 C=0.252D-02 G=0.189D+00
     NIC=  0 NIT=   9 NFV=   14 NFG=  100 F=0.529D+01 C=0.927D-03 G=0.514D-01
     NIC=  0 NIT=  10 NFV=   15 NFG=  110 F=0.529D+01 C=0.190D-03 G=0.111D-01
     NIC=  0 NIT=  11 NFV=   16 NFG=  120 F=0.529D+01 C=0.238D-04 G=0.117D-02
     NIC=  0 NIT=  12 NFV=   17 NFG=  130 F=0.529D+01 C=0.447D-06 G=0.220D-04
     NIC=  0 NIT=  13 NFV=   18 NFG=  140 F=0.529D+01 C=0.168D-09 G=0.824D-08
   0 NIC=  0 NIT=  14 NFV=   18 NFG=  140 F=0.529D+01 C=0.168D-09 G=0.824D-08
 TIME= 0:00:00.00
```

7.25 Large scale least squares optimization with nonlinear equality constraints

a) Problem description:

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

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

where $n = 100$ and

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

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 $x_i = -1,\ 1 \leq i \leq n$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SPARSE LEAST SQUARES WITH NONLINEAR EQUALITY CONSTRAINTS      +
$REM +---------------------------------------------------------------+
$SET(INPUT)
  DO 1 I=1,NF
    X(I)=-1.0D0
1 CONTINUE
  MA=0
  IAG(1)=1
  DO 2 I=1,NA
    IF (I.GT.1) THEN
    MA=MA+1
    JAG(MA)=I-1
    ENDIF
    MA=MA+1
    JAG(MA)=I
    IF (I.LT.NF) THEN
    MA=MA+1
    JAG(MA)=I+1
    ENDIF
    IAG(I+1)=MA+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)=0.0D0
4 CONTINUE
$ENDSET
$SET(FMODELA)
  IF(KA.EQ.1)THEN
  FA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA+1)
```

```
  ELSEIF(KA.EQ.NF)THEN
  FA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA-1)
  ELSE
  FA=(3.0D0-2.0D0*X(KA))*X(KA)+1.0D0-X(KA-1)-X(KA+1)
  ENDIF
$ENDSET
$SET(FMODELC)
  K=KC+2
  FC=8.0D0*X(K)*(X(K)**2-X(K-1))-2.0D0*(1.0D0-X(K))&
+4.0D0*(X(K)-X(K+1)**2)+X(K-1)**2-X(K-2)+X(K+1)-X(K+2)**2
$ENDSET
$IADA=1
$IADC=1
$NF=100
$NA=100
$NC=96
$NCE=NC
$MODEL='AQ'
$JACA='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian, matrix the sparsity pattern of the constraint Jacobian matrix, and the constraint specifications. The sparse Jacobian matrix, indicated by the statement $JACA='S' is tridiagonal. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. The statement $NCE=NC specifies that all constraints are equalities (thus the special method for equality constraints can be used). By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum-of-squares minimization we set $MODEL='AQ'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LK3   UPDATE = N   MODEL = AQ   HESF = S     NF =    100
     NIC=  0 NIT=   0 NFV=   19 NFG=   19 F=0.205D+03 C=0.280D+02 G=0.190D+02
     NIC=  0 NIT=   1 NFV=   39 NFG=   39 F=0.363D+04 C=0.880D+01 G=0.631D+01
     NIC=  0 NIT=   2 NFV=   59 NFG=   59 F=0.484D+03 C=0.285D+01 G=0.244D+01
     NIC=  0 NIT=   3 NFV=   79 NFG=   79 F=0.169D+03 C=0.127D+01 G=0.320D+01
     NIC=  0 NIT=   4 NFV=   99 NFG=   99 F=0.641D+02 C=0.851D+00 G=0.445D+01
     NIC=  0 NIT=   5 NFV=  121 NFG=  121 F=-.181D+02 C=0.593D+00 G=0.330D+01
     NIC=  0 NIT=   6 NFV=  141 NFG=  141 F=0.465D+01 C=0.307D+00 G=0.765D+00
     NIC=  0 NIT=   7 NFV=  161 NFG=  161 F=0.267D+01 C=0.157D-01 G=0.231D+00
     NIC=  0 NIT=   8 NFV=  181 NFG=  181 F=0.265D+01 C=0.278D-02 G=0.889D-01
     NIC=  0 NIT=   9 NFV=  201 NFG=  201 F=0.265D+01 C=0.107D-02 G=0.257D-01
     NIC=  0 NIT=  10 NFV=  221 NFG=  221 F=0.265D+01 C=0.198D-03 G=0.516D-02
     NIC=  0 NIT=  11 NFV=  241 NFG=  241 F=0.265D+01 C=0.214D-04 G=0.534D-03
     NIC=  0 NIT=  12 NFV=  261 NFG=  261 F=0.265D+01 C=0.373D-06 G=0.918D-05
     NIC=  0 NIT=  13 NFV=  281 NFG=  281 F=0.265D+01 C=0.117D-09 G=0.287D-08
```

```
  0 NIC=  0 NIT=  14 NFV=  281 NFG=  281 F=0.265D+01 C=0.117D-09 G=0.287D-08
 TIME= 0:00:00.24
```

7.26 Large-scale nonlinear programming (sparse Hessian matrix)

a) Problem description:

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

$$F(x) = \sum_{i=1}^{n-3} \big(x_i^2 + x_{i+1}^2 + 2x_{i+2}^2 + x_{i+3}^2 - 5x_i - 5x_{i+1} - 21x_{i+2} + 7x_{i+3}\big),$$

where $n = 100$, over the set given by the nonlinear constraints

$$\begin{aligned} x_{j-1}^2 + x_j^2 + x_{j+1}^2 + x_{j+2}^2 + x_{j-1} - x_j + x_{j+1} - x_{j+2} \le 8, \qquad & \mathrm{mod}(k,3) = 0 \\ x_{j-1}^2 + 2x_j^2 + x_{j+1}^2 + 2x_{j+2}^2 - x_{j-1} - x_{j+2} \le 10, \qquad & \mathrm{mod}(k,3) = 1 \\ 2x_{j-1}^2 + x_j^2 + x_{j+1}^2 + 2x_{j-1} - x_j - x_{j+2} \le 5, \qquad & \mathrm{mod}(k,3) = 2 \end{aligned}$$

where $j = 2(\mathrm{div}(k-1,3)+1)$, $1 \le k \le 3(n-2)/2$ (div(k,l) is the integer division and $\mathrm{mod}(k,l)$ is the remainder after integer division). The starting point is $x_i = 0$, $1 \le i \le n$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SPARSE OPTIMIZATION WITH NONLINEAR CONSTRAINTS                 +
$REM +---------------------------------------------------------------+
$SET(INPUT)
   DO 1 I=1,NF
   X(I)=0.0D0
 1 CONTINUE
   DO 2 I=1,NF
   IH(I)=I
   JH(I)=I
 2 CONTINUE
   IH(NF+1)=NF+1
   MC=0
   KC=0
   DO 5 J=1,NC/3
   CU(KC+1)=8.0D0
   CU(KC+2)=1.0D1
   CU(KC+3)=5.0D0
   I=2*(J-1)
   DO 4 K=1,3
   IC(KC+K)=2
   ICG(KC+K)=MC+1
   DO 3 L=1,4
   JCG(MC+L)=I+L
 3 CONTINUE
   MC=MC+4
 4 CONTINUE
   KC=KC+3
 5 CONTINUE
   ICG(NC+1)=MC+1
```

```
$ENDSET
$SET(FMODELF)
   FF=0.0D0
   DO 11 I=1,NF-3
   FF=FF+X(I)**2+X(I+1)**2+2.0D0*X(I+2)**2+X(I+3)**2&
-5.0D0*X(I)-5.0D0*X(I+1)-2.1D1*X(I+2)+7.0D0*X(I+3)
11 CONTINUE
$ENDSET
$SET(GMODELF)
   DO 12 I=1,NF
   GF(I)=0.0D0
12 CONTINUE
   DO 13 I=1,NF-3
   GF(I)=GF(I)+2.0D0*X(I)-5.0D0
   GF(I+1)=GF(I+1)+2.0D0*X(I+1)-5.0D0
   GF(I+2)=GF(I+2)+4.0D0*X(I+2)-2.1D1
   GF(I+3)=GF(I+3)+2.0D0*X(I+3)+7.0D0
13 CONTINUE
$ENDSET
$SET(FMODELC)
   J=2*((KC-1)/3+1)
   L=MOD(KC,3)
   GO TO (21,22,23), L+1
21 FC=X(J-1)**2+X(J)**2+X(J+1)**2+X(J+2)**2+X(J-1)-X(J)+X(J+1)-X(J+2)
   GO TO 24
22 FC=X(J-1)**2+2.0D0*X(J)**2+X(J+1)**2+2.0D0*X(J+2)**2-X(J-1)-X(J+2)
   GO TO 24
23 FC=2.0D0*X(J-1)**2+X(J)**2+X(J+1)**2+2.0D0*X(J-1)-X(J)-X(J+2)
24 CONTINUE
$ENDSET
$SET(GMODELC)
   J=2*((KC-1)/3+1)
   L=MOD(KC,3)
   GO TO (25,26,27), L+1
25 GC(J-1)=2.0D0*X(J-1)+1.0D0
   GC(J)=2.0D0*X(J)-1.0D0
   GC(J+1)=2.0D0*X(J+1)+1.0D0
   GC(J+2)=2.0D0*X(J+2)-1.0D0
   GO TO 28
26 GC(J-1)=2.0D0*X(J-1)-1.0D0
   GC(J)=4.0D0*X(J)
   GC(J+1)=2.0D0*X(J+1)
   GC(J+2)=4.0D0*X(J+2)-1.0D0
   GO TO 28
27 GC(J-1)=4.0D0*X(J-1)+2.0D0
   GC(J)=2.0D0*X(J)-1.0D0
   GC(J+1)=2.0D0*X(J+1)
   GC(J+2)=-1.0D0
28 CONTINUE
$ENDSET
$NF=100
$NC=147
```

```
$JACC='S'
$HESF='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Hessian matrix, the sparsity pattern of the constraint Jacobian matrix, and the constraint specifications. The sparse Hessian matrix, indicated by the statement $HESF='S', is diagonal. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. By using the macrovariable $FMODELC we specify analytically the values of the constraint functions. By using the macrovariable $GMODELC we specify analytically the gradients of the constraint functions.

d) Problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = FF   HESF = S     NF =    100
     NIC=  0 NIT=   0 NFV=    1 NFG=    7 F=0.000D+00 C=0.000D+00 G=0.307D+02
     NIC=  0 NIT=   1 NFV=    2 NFG=   14 F=-.273D+04 C=0.177D+02 G=0.580D+01
     NIC=  0 NIT=   2 NFV=    3 NFG=   21 F=-.238D+04 C=0.107D+02 G=0.402D+01
     NIC=  0 NIT=   3 NFV=    4 NFG=   28 F=-.203D+04 C=0.227D+01 G=0.199D+02
     NIC=  0 NIT=   4 NFV=    5 NFG=   35 F=-.208D+04 C=0.227D+02 G=0.175D+02
     NIC=  0 NIT=   5 NFV=    6 NFG=   42 F=-.207D+04 C=0.573D+01 G=0.206D+02
     NIC=  0 NIT=   6 NFV=    7 NFG=   49 F=-.207D+04 C=0.179D+01 G=0.105D+01
     NIC=  0 NIT=   7 NFV=    8 NFG=   56 F=-.207D+04 C=0.613D+00 G=0.131D+01
     NIC=  0 NIT=   8 NFV=    9 NFG=   63 F=-.207D+04 C=0.189D+00 G=0.169D+01
     NIC=  0 NIT=   9 NFV=   10 NFG=   70 F=-.207D+04 C=0.150D-01 G=0.912D-01
     NIC=  0 NIT=  10 NFV=   11 NFG=   77 F=-.207D+04 C=0.307D-03 G=0.859D-03
     NIC=  0 NIT=  11 NFV=   12 NFG=   84 F=-.207D+04 C=0.178D-07 G=0.123D-05
     NIC=  0 NIT=  12 NFV=   13 NFG=   91 F=-.207D+04 C=0.000D+00 G=0.629D-07
     NIC=  0 NIT=  13 NFV=   14 NFG=   98 F=-.207D+04 C=0.000D+00 G=0.314D-08
     NIC=  0 NIT=  14 NFV=   15 NFG=  105 F=-.207D+04 C=0.888D-15 G=0.157D-09
   0 NIC=  0 NIT=  15 NFV=   15 NFG=  105 F=-.207D+04 C=0.888D-15 G=0.157D-09
 TIME= 0:00:00.02
```

7.27 Large-scale nonlinear programming (sparse Jacobian matrix)

a) Problem description:

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

$$F(x) = \sum_{1=1}^{n-3} f_i^A(x),$$

where

$$f_i^A(x) = (x_i^2 + x_{i+1}^2 + 2x_{i+2}^2 + x_{i+3}^2 - 5x_i - 5x_{i+1} - 21x_{i+2} + 7x_{i+3}, \quad 1 \le i \le n-3,$$

and $n = 100$, over the set given by the nonlinear constraints

$$\begin{aligned} x_{j-1}^2 + x_j^2 + x_{j+1}^2 + x_{j+2}^2 + x_{j-1} - x_j + x_{j+1} - x_{j+2} \le 8, \qquad & \mathrm{mod}(k,3) = 0 \\ x_{j-1}^2 + 2x_j^2 + x_{j+1}^2 + 2x_{j+2}^2 - x_{j-1} - x_{j+2} \le 10, \qquad & \mathrm{mod}(k,3) = 1 \\ 2x_{j-1}^2 + x_j^2 + x_{j+1}^2 + 2x_{j-1} - x_j - x_{j+2} \le 5, \qquad & \mathrm{mod}(k,3) = 2 \end{aligned}$$

where $j = 2(\mathrm{div}(k-1,3)+1)$, $1 \le k \le 3(n-2)/2$ (div(k,l) is the integer division and $\mathrm{mod}(k,l)$ is the remainder after integer division). The starting point is $x_i = 0,\ 1 \le i \le n$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + PARTIALLY SEPARABLE OPTIMIZATION WITH NONLINEAR CONSTRAINTS   +
$REM +-------------------------------------------------------------+
$SET(INPUT)
   DO 1 I=1,NF
   X(I)=0.0D0
 1 CONTINUE
   MA=0
   IAG(1)=1
   DO 8 KA=1,NA
   DO 9 K=1,4
   JAG(MA+K)=KA+K-1
 9 CONTINUE
   MA=MA+4
   IAG(KA+1)=MA+1
 8 CONTINUE
   MC=0
   KC=0
   DO 5 J=1,NC/3
   CU(KC+1)=8.0D0
   CU(KC+2)=1.0D1
   CU(KC+3)=5.0D0
   I=2*(J-1)
   DO 4 K=1,3
   IC(KC+K)=2
   ICG(KC+K)=MC+1
   DO 3 L=1,4
   JCG(MC+L)=I+L
 3 CONTINUE
   MC=MC+4
 4 CONTINUE
   KC=KC+3
 5 CONTINUE
   ICG(NC+1)=MC+1
$ENDSET
$SET(FMODELA)
   FA=X(KA)**2+X(KA+1)**2+2.0D0*X(KA+2)**2+X(KA+3)**2&
-5.0D0*X(KA)-5.0D0*X(KA+1)-2.1D1*X(KA+2)+7.0D0*X(KA+3)
$ENDSET
$SET(FMODELC)
   J=2*((KC-1)/3+1)
   L=MOD(KC,3)
   GO TO (21,22,23), L+1
21 FC=X(J-1)**2+X(J)**2+X(J+1)**2+X(J+2)**2+X(J-1)-X(J)+X(J+1)-X(J+2)
   GO TO 24
22 FC=X(J-1)**2+2.0D0*X(J)**2+X(J+1)**2+2.0D0*X(J+2)**2-X(J-1)-X(J+2)
   GO TO 24
23 FC=2.0D0*X(J-1)**2+X(J)**2+X(J+1)**2+2.0D0*X(J-1)-X(J)-X(J+2)
24 CONTINUE
$ENDSET
```

```
$IADA=1
$IADC=1
$NF=100
$NA=97
$NC=147
$MODEL='AF'
$JACA='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian matrix, the sparsity pattern of the constraint Jacobian matrix and the constraint specifications. The sparse objective Jacobian matrix is indicated by the statement $JACA='S'. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum of values minimization we set $MODEL='AF'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = AF   HESF = S     NF =    100
     NIC=  0 NIT=   0 NFV=    8 NFG=    8 F=0.000D+00 C=0.000D+00 G=0.304D+02
     NIC=  0 NIT=   1 NFV=   17 NFG=   17 F=-.271D+04 C=0.202D+02 G=0.595D+01
     NIC=  0 NIT=   2 NFV=   26 NFG=   26 F=-.247D+04 C=0.899D+01 G=0.876D+01
     NIC=  0 NIT=   3 NFV=   35 NFG=   35 F=-.218D+04 C=0.239D+01 G=0.224D+02
     NIC=  0 NIT=   4 NFV=   44 NFG=   44 F=-.231D+04 C=0.308D+02 G=0.250D+01
     NIC=  0 NIT=   5 NFV=   53 NFG=   53 F=-.227D+04 C=0.115D+02 G=0.185D+02
     NIC=  0 NIT=   6 NFV=   62 NFG=   62 F=-.228D+04 C=0.296D+01 G=0.497D+02
     NIC=  0 NIT=   7 NFV=   71 NFG=   71 F=-.229D+04 C=0.192D+00 G=0.342D+01
     NIC=  0 NIT=   8 NFV=   80 NFG=   80 F=-.229D+04 C=0.229D-01 G=0.185D+00
     NIC=  0 NIT=   9 NFV=   89 NFG=   89 F=-.229D+04 C=0.240D-02 G=0.147D-02
     NIC=  0 NIT=  10 NFV=   98 NFG=   98 F=-.229D+04 C=0.606D-06 G=0.237D-06
     NIC=  0 NIT=  11 NFV=  107 NFG=  107 F=-.229D+04 C=0.000D+00 G=0.524D-08
     NIC=  0 NIT=  12 NFV=  116 NFG=  116 F=-.229D+04 C=0.000D+00 G=0.262D-09
     NIC=  0 NIT=  13 NFV=  125 NFG=  125 F=-.229D+04 C=0.000D+00 G=0.131D-10
   0 NIC=  0 NIT=  14 NFV=  125 NFG=  125 F=-.229D+04 C=0.000D+00 G=0.131D-10
 TIME= 0:00:00.17
```

7.28 Large scale least squares optimization with nonlinear constraints

a) Problem description:

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

$$F(x) = \frac{1}{2}\sum_{i=1}^{n-1}\left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right),$$

where $n = 100$, with constraints

$$3x_{i+1}^3 + 2x_{i+2} - 5 + \sin(x_{i+1} - x_{i+2})\sin(x_{i+1} + x_{i+2}) + 4x_{i+1} - x_i \exp(x_i - x_{i+1}) - 3 \geq 2$$

for $1 \leq i \leq n-2$. The starting point is $x_i = -1.2$, $\mathrm{mod}(i,2) = 1$, $x_i = 1.0$, $\mathrm{mod}(i,2) = 0$. The minimum value of the objective function is $F = 0.0$ (This problem is equivalent to the previous one).

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE LEAST SQUARES WITH NONLINEAR CONSTRAINTS              +
$REM +-------------------------------------------------------------+
$FLOAT W1,W2,W3
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)=1.0D0
    ENDIF
1 CONTINUE
  DO 2 I=1,NF-1
    J=3*(I-1)+1
    JAG(J)=I
    JAG(J+1)=I+1
    JAG(J+2)=I
    K=2*(I-1)+1
    IAG(K)=J
    IAG(K+1)=J+2
2 CONTINUE
  IAG(K+2)=J+3
  NA=2*(NF-1)
  MA=3*(NF-1)
  NC=NF-2
  MC=0
  ICG(1)=1
  DO 3 I=2,NF-1
    MC=MC+1
    JCG(MC)=I-1
    MC=MC+1
    JCG(MC)=I
    MC=MC+1
    JCG(MC)=I+1
    ICG(I)=MC+1
3 CONTINUE
  CALL UXVINS(NC,1,IC)
  CALL UXVSET(NC,2.0D0,CL)
$ENDSET
$SET(FMODELA)
  I=(KA+1)/2
  IF(MOD(KA,2).EQ.1)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=X(I)-1.0D0
  ENDIF
$ENDSET
```

```
$SET(FMODELC)
  K=KC+1
  FC=3.0D0*X(K)**3+2.0D0*X(K+1)-5.0D0+SIN(X(K)-X(K+1))*SIN(X(K)&
+X(K+1))+4.0D0*X(K)-X(K-1)*EXP(X(K-1)-X(K))-3.0D0
$ENDSET
$IADA=1
$IADC=1
$NF=100
$NA=198
$NC=98
$MODEL='AQ'
$JACA='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian matrix, the sparsity pattern of the constraint Jacobian matrix and the constraint specifications. The sparse objective Jacobian matrix is indicated by the statement $JACA='S'. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the sum-of-squares minimization we set $MODEL='AQ'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = AQ   HESF = S     NF =    100
     NIC=   0 NIT=   0 NFV=   11 NFG=   11 F=0.126D+05 C=0.268D+02 G=0.396D+03
     NIC=   0 NIT=   1 NFV=   23 NFG=   23 F=0.162D+05 C=0.125D+02 G=0.181D+03
     NIC=   0 NIT=   2 NFV=   35 NFG=   35 F=-.136D+05 C=0.594D+01 G=0.170D+04
     NIC=   0 NIT=   3 NFV=   47 NFG=   47 F=-.694D+03 C=0.379D+00 G=0.196D+03
     NIC=   0 NIT=   4 NFV=   59 NFG=   59 F=0.107D+03 C=0.142D-01 G=0.192D+02
     NIC=   0 NIT=   5 NFV=   71 NFG=   71 F=0.108D+03 C=0.331D-01 G=0.473D+01
     NIC=   0 NIT=   6 NFV=   83 NFG=   83 F=0.108D+03 C=0.000D+00 G=0.134D+02
     NIC=   0 NIT=   7 NFV=   95 NFG=   95 F=0.107D+03 C=0.138D-03 G=0.144D+02
     NIC=   0 NIT=   8 NFV=  107 NFG=  107 F=0.107D+03 C=0.113D-05 G=0.664D+01
     NIC=   0 NIT=   9 NFV=  119 NFG=  119 F=0.106D+03 C=0.431D-09 G=0.889D+01
     NIC=   0 NIT=  10 NFV=  131 NFG=  131 F=0.106D+03 C=0.000D+00 G=0.127D+02
     NIC=   0 NIT=  11 NFV=  143 NFG=  143 F=0.106D+03 C=0.116D-03 G=0.619D+00
     NIC=   0 NIT=  12 NFV=  155 NFG=  155 F=0.106D+03 C=0.000D+00 G=0.717D+01
     NIC=   0 NIT=  13 NFV=  167 NFG=  167 F=0.106D+03 C=0.733D-05 G=0.268D+00
     NIC=   0 NIT=  14 NFV=  179 NFG=  179 F=0.106D+03 C=0.000D+00 G=0.835D-01
     NIC=   0 NIT=  15 NFV=  191 NFG=  191 F=0.106D+03 C=0.433D-07 G=0.296D-02
     NIC=   0 NIT=  16 NFV=  203 NFG=  203 F=0.106D+03 C=0.178D-14 G=0.154D-04
     NIC=   0 NIT=  17 NFV=  215 NFG=  215 F=0.106D+03 C=0.178D-14 G=0.130D-11
   0 NIC=   0 NIT=  18 NFV=  215 NFG=  215 F=0.106D+03 C=0.178D-14 G=0.130D-11
 TIME= 0:00:00.22
```

7.29 Large-scale l_1 optimization with nonlinear constrains

a) Problem description:

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

$$F(x) = \sum_{i=1}^{n-1} \left(10|x_i^2 - x_{i+1}| + |x_i - 1|\right),$$

where $n = 100$, with constraints

$$3x_{i+1}^3 + 2x_{i+2} - 5 + \sin(x_{i+1} - x_{i+2})\sin(x_{i+1} + x_{i+2}) + 4x_{i+1} - x_i \exp(x_i - x_{i+1}) - 3 \geq 2$$

for $1 \leq i \leq n-2$. The starting point is $x_i = -1.2$, $\mathrm{mod}(i,2) = 1$, $x_i = 1.0$, $\mathrm{mod}(i,2) = 0$.

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + SPARSE L-1 OPTIMIZATION WITH NONLINEAR CONSTRAINTS          +
$REM +-------------------------------------------------------------+
$FLOAT W1,W2,W3
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)=1.0D0
    ENDIF
1 CONTINUE
  DO 2 I=1,NF-1
    J=3*(I-1)+1
    JAG(J)=I
    JAG(J+1)=I+1
    JAG(J+2)=I
    K=2*(I-1)+1
    IAG(K)=J
    IAG(K+1)=J+2
2 CONTINUE
  IAG(K+2)=J+3
  NA=2*(NF-1)
  MA=3*(NF-1)
  NC=NF-2
  MC=0
  ICG(1)=1
  DO 3 I=2,NF-1
    MC=MC+1
    JCG(MC)=I-1
    MC=MC+1
    JCG(MC)=I
    MC=MC+1
    JCG(MC)=I+1
    ICG(I)=MC+1
3 CONTINUE
  CALL UXVINS(NC,1,IC)
```

```
  CALL UXVSET(NC,2.0D0,CL)
$ENDSET
$SET(FMODELA)
  I=(KA+1)/2
  IF(MOD(KA,2).EQ.1)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=X(I)-1.0D0
  ENDIF
$ENDSET
$SET(FMODELC)
  K=KC+1
  FC=3.0D0*X(K)**3+2.0D0*X(K+1)-5.0D0+SIN(X(K)-X(K+1))*SIN(X(K)&
+X(K+1))+4.0D0*X(K)-X(K-1)*EXP(X(K-1)-X(K))-3.0D0
$ENDSET
$IADA=1
$IADC=1
$NF=100
$NA=198
$NC=98
$M=10000
$MODEL='AA'
$JACA='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian matrix, the sparsity pattern of the constraint Jacobian matrix and the constraint specifications. The sparse objective Jacobian matrix is indicated by the statement $JACA='S'. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the l_1 (sum of absolute values) minimization we set $MODEL='AA'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = AA   HESF = S     NF =    298
     NIC=  0 NIT=   0 NFV=    1 NFG=    7 F=0.302D+03 C=0.268D+02 G=0.280D+01
     NIC=  0 NIT=   1 NFV=    2 NFG=   14 F=0.908D+04 C=0.110D+02 G=0.302D+03
     NIC=  0 NIT=   2 NFV=    3 NFG=   21 F=0.955D+04 C=0.149D+02 G=0.503D+03
     NIC=  0 NIT=   3 NFV=    4 NFG=   28 F=0.332D+04 C=0.136D+02 G=0.274D+03
     NIC=  0 NIT=   4 NFV=    5 NFG=   35 F=0.204D+03 C=0.783D+02 G=0.157D+04
     NIC=  0 NIT=   5 NFV=    6 NFG=   42 F=0.288D+04 C=0.262D+02 G=0.116D+04
     NIC=  0 NIT=   6 NFV=    7 NFG=   49 F=0.120D+03 C=0.160D+02 G=0.207D+03
     NIC=  0 NIT=   7 NFV=   11 NFG=   56 F=0.151D+03 C=0.141D+02 G=0.181D+03
     NIC=  0 NIT=   8 NFV=   15 NFG=   63 F=0.164D+03 C=0.123D+02 G=0.158D+03
     NIC=  0 NIT=   9 NFV=   16 NFG=   70 F=0.175D+03 C=0.994D+00 G=0.127D+03
     NIC=  0 NIT=  10 NFV=   17 NFG=   77 F=0.156D+03 C=0.186D+01 G=0.143D+02
```

```
     NIC=  0 NIT=  11 NFV=   18 NFG=   84 F=0.156D+03 C=0.350D+00 G=0.632D+01
     NIC=  0 NIT=  12 NFV=   19 NFG=   91 F=0.155D+03 C=0.173D+01 G=0.830D+01
     NIC=  0 NIT=  13 NFV=   21 NFG=   98 F=0.325D+03 C=0.210D+03 G=0.626D+04
     NIC=  0 NIT=  14 NFV=   22 NFG=  105 F=0.175D+03 C=0.412D+03 G=0.312D+03
     NIC=  0 NIT=  15 NFV=   23 NFG=  112 F=0.155D+03 C=0.304D+02 G=0.530D+01
     NIC=  0 NIT=  16 NFV=   24 NFG=  119 F=0.155D+03 C=0.132D-01 G=0.167D+01
     NIC=  0 NIT=  17 NFV=   25 NFG=  126 F=0.155D+03 C=0.176D+01 G=0.652D+00
     NIC=  0 NIT=  18 NFV=   26 NFG=  133 F=0.155D+03 C=0.951D+00 G=0.115D-01
     NIC=  0 NIT=  19 NFV=   27 NFG=  140 F=0.155D+03 C=0.258D-03 G=0.289D-04
     NIC=  0 NIT=  20 NFV=   28 NFG=  147 F=0.155D+03 C=0.899D-07 G=0.110D-07
     NIC=  0 NIT=  21 NFV=   29 NFG=  154 F=0.155D+03 C=0.888D-15 G=0.449D-09
   0 NIC=  0 NIT=  22 NFV=   29 NFG=  154 F=0.155D+03 C=0.888D-15 G=0.449D-09
 TIME= 0:00:00.42
```

7.30 Large-scale l_∞ (minimax) optimization with nonlinear constraints

a) Problem description:

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

$$F(x) = \max_{1 \le i \le n-1} \left(\max(10|x_i^2 - x_{i+1}|, |x_i - 1|) \right),$$

where $n = 100$, with constraints

$$3x_{i+1}^3 + 2x_{i+2} - 5 + \sin(x_{i+1} - x_{i+2}) \sin(x_{i+1} + x_{i+2}) + 4x_{i+1} - x_i \exp(x_i - x_{i+1}) - 3 \ge 2$$

for $1 \le i \le n-2$. The starting point is $x_i = -1.2$, $\mathrm{mod}(i,2) = 1$, $x_i = 1.0$, $\mathrm{mod}(i,2) = 0$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SPARSE MINIMAX OPTIMIZATION WITH NONLINEAR CONSTRAINTS       +
$REM +---------------------------------------------------------------+
$FLOAT W1,W2,W3
$SET(INPUT)
  DO 1 I=1,NF
    IF(MOD(I,2).EQ.1) THEN
      X(I)=-1.2D0
    ELSE
      X(I)=1.0D0
    ENDIF
1 CONTINUE
  DO 2 I=1,NF-1
    J=3*(I-1)+1
    JAG(J)=I
    JAG(J+1)=I+1
    JAG(J+2)=I
    K=2*(I-1)+1
    IAG(K)=J
    IAG(K+1)=J+2
2 CONTINUE
  IAG(K+2)=J+3
  NA=2*(NF-1)
  MA=3*(NF-1)
```

```
  NC=NF-2
  MC=0
  ICG(1)=1
  DO 3 I=2,NF-1
    MC=MC+1
    JCG(MC)=I-1
    MC=MC+1
    JCG(MC)=I
    MC=MC+1
    JCG(MC)=I+1
    ICG(I)=MC+1
3 CONTINUE
  CALL UXVINS(NC,1,IC)
  CALL UXVSET(NC,2.0D0,CL)
$ENDSET
$SET(FMODELA)
  I=(KA+1)/2
  IF(MOD(KA,2).EQ.1)THEN
    FA=1.0D1*(X(I)**2-X(I+1))
  ELSE
    FA=X(I)-1.0D0
  ENDIF
$ENDSET
$SET(FMODELC)
  K=KC+1
  FC=3.0D0*X(K)**3+2.0D0*X(K+1)-5.0D0+SIN(X(K)-X(K+1))*SIN(X(K)&
+X(K+1))+4.0D0*X(K)-X(K-1)*EXP(X(K-1)-X(K))-3.0D0
$ENDSET
$IADA=1
$IADC=1
$NF=100
$NA=198
$NC=98
$M=10000
$MODEL='AM'
$JACA='S'
$JACC='S'
$EPS6='1.0D-2'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian matrix, the sparsity pattern of the constraint Jacobian matrix and the constraint specifications. The sparse objective Jacobian matrix is indicated by the statement $JACA='S'. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. Note that statements IF-THEN and ELSEIF-THEN have not to contain blanks if the automatic differentiation is used. For the l_∞ minimization (minimax) we set $MODEL='AM'.

d) Problem solution (basic screen output):

```
 CLASS = MN - LI3   UPDATE = N   MODEL = AM   HESF = S     NF =    101
     NIC=  0 NIT=   0 NFV=    1 NFG=    7 F=0.302D+03 C=0.268D+02 G=0.386D+02
     NIC=  0 NIT=   1 NFV=    2 NFG=   14 F=0.924D+04 C=0.118D+02 G=0.475D+03
     NIC=  0 NIT=   2 NFV=    3 NFG=   21 F=0.386D+04 C=0.157D+02 G=0.794D+03
     NIC=  0 NIT=   3 NFV=    4 NFG=   28 F=-.193D+05 C=0.201D+02 G=0.262D+03
     NIC=  0 NIT=   4 NFV=    5 NFG=   35 F=-.387D+03 C=0.711D+00 G=0.633D+02
     NIC=  0 NIT=   5 NFV=    6 NFG=   42 F=-.443D+02 C=0.280D+01 G=0.171D+02
     NIC=  0 NIT=   6 NFV=    7 NFG=   49 F=0.861D+01 C=0.330D+01 G=0.194D+01
     NIC=  0 NIT=   7 NFV=    8 NFG=   56 F=0.220D+01 C=0.291D+01 G=0.357D+00
     NIC=  0 NIT=   8 NFV=    9 NFG=   63 F=0.116D+01 C=0.182D+01 G=0.395D-01
     NIC=  0 NIT=   9 NFV=   10 NFG=   70 F=0.224D+01 C=0.143D+01 G=0.167D+01
     NIC=  0 NIT=  10 NFV=   11 NFG=   77 F=0.695D+00 C=0.831D+01 G=0.262D+01
     ------------------------------------------------------------------------
     NIC=  0 NIT=  61 NFV=  303 NFG=  434 F=0.192D+01 C=0.907D-02 G=0.344D-04
     NIC=  0 NIT=  62 NFV=  310 NFG=  441 F=0.192D+01 C=0.893D-02 G=0.338D-04
     NIC=  0 NIT=  63 NFV=  315 NFG=  448 F=0.192D+01 C=0.832D-02 G=0.312D-04
     NIC=  0 NIT=  64 NFV=  316 NFG=  455 F=0.192D+01 C=0.000D+00 G=0.308D-07
     NIC=  0 NIT=  65 NFV=  323 NFG=  462 F=0.192D+01 C=0.000D+00 G=0.303D-07
     NIC=  0 NIT=  66 NFV=  329 NFG=  469 F=0.192D+01 C=0.000D+00 G=0.288D-07
     NIC=  0 NIT=  67 NFV=  331 NFG=  476 F=0.192D+01 C=0.000D+00 G=0.143D-07
     NIC=  0 NIT=  68 NFV=  338 NFG=  483 F=0.192D+01 C=0.000D+00 G=0.141D-07
     NIC=  0 NIT=  69 NFV=  345 NFG=  490 F=0.192D+01 C=0.000D+00 G=0.427D-07
   0 NIC=  0 NIT=  70 NFV=  345 NFG=  490 F=0.192D+01 C=0.000D+00 G=0.427D-07
 TIME= 0:00:01.80
```

7.31 Optimization of dynamical systems - general integral criterion

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \frac{1}{2}\int_0^T (y_1^2(t) + y_2^2(t))dt + \frac{1}{2}(y_1^2(T) + y_2^2(T))$$

where $T = 1.5$ and where

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

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

b) Problem specification (input file):

```
$REM +-------------------------------------------------------------+
$REM + OPTIMIZATION OF DYNAMICAL SYSTEMS - GENERAL CRITERION       +
$REM +-------------------------------------------------------------+
$SET(INPUT)
  X(1)=0.0D0
  TA=0.0D0
  TAMAX=1.5D0
$ENDSET
```

```
$SET(FMODELF)
  FF=0.5D0*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODELF)
  DF(1)=YA(1)
  DF(2)=YA(2)
$ENDSET
$SET(FMODELA)
  FA=HALF*(YA(1)**2+YA(2)**2)
$ENDSET
$SET(DMODELA)
  DA(1)=YA(1)
  DA(2)=YA(2)
$ENDSET
$SET(FMODELE)
  GO TO (1,2) KE
1 FE=YA(2)
  GO TO 3
2 FE=-YA(1)+(ONE-YA(1)**2)*YA(2)
3 CONTINUE
$ENDSET
$SET(DMODELE)
  GO TO (4,5) KE
4 DE(1)=ZERO
  DE(2)=ONE
  GO TO 6
5 DE(1)=-ONE-TWO*YA(1)*YA(2)
  DE(2)=ONE-YA(1)**2
6 CONTINUE
$ENDSET
$SET(FMODELY)
  GO TO (7,8) KE
7 FE=X(1)
  GO TO 9
8 FE=ONE
9 CONTINUE
$ENDSET
$SET(GMODELY)
   GO TO (10,11) KE
10 GE(1)=ONE
   GO TO 12
11 GE(1)=ZERO
12 CONTINUE
$ENDSET
$NF=1
$NE=2
$MODEL='DF'
$TOLR='1.0D-9'
$TOLA='1.0D-9'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial value of the variable x_1 as well as the initial and terminal times 0 and T, respectively. By using the macrovariables $FMODELA and $DMODELA we specify the subintegral function and by using the macrovariables $FMODELF and $DMODELF we specify the terminal function. The right hand sides of the differential equations are specified by using the macrovariables $FMODELE and $DMODELE, while the initial values and their derivatives are given by using the macrovariables $FMODELY and $GMODELY. The option $MODEL='DF' indicates a general integral criterion.

d) Problem solution (basic screen output):

```
 CLASS = VM - LI1   UPDATE = B   MODEL = DF   HESF = D     NF =      1
     NIT=    0 NFV=    1 NFG=    0  F=  2.763393900     G=0.242D+01
     NIT=    1 NFV=    3 NFG=    0  F=  1.974913643     G=0.513D+00
     NIT=    2 NFV=    4 NFG=    0  F=  1.944408382     G=0.398D-01
     NIT=    3 NFV=    5 NFG=    0  F=  1.944233193     G=0.478D-03
     NIT=    4 NFV=    6 NFG=    0  F=  1.944233168     G=0.338D-06
   0 NIT=    4 NFV=    6 NFG=    0  GRAD TOL  F=  1.944233168     G=0.338D-06
 FF =  0.7671653593D+00
 X  =  0.6169839176D+00
 TIME= 0:00:00.00
```

7.32 Optimization of dynamical systems - special integral criterion

a) Problem description:

Suppose we have to find a minimum of the objective function

$$F(x) = \frac{1}{2}\int_0^T (y_1(t) - 1/(1+t))^2 dt$$

where $T = 1$ and where

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

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + OPTIMIZATION OF DYNAMICAL SYSTEMS - QUADRATIC CRITERION       +
$REM +---------------------------------------------------------------+
$SET(INPUT)
  X(1)=2.0D0
  X(2)=0.0D0
  TA=0.0D0
  TAMAX=1.0D0
$ENDSET
$SET(FMODELE)
  FE=-X(1)*YA(1)**2
  YE=1.0D0/(1.0D0+TA)
  WE=1.0D0
$ENDSET
$SET(GDMODELE)
  GE(1)=-YA(1)**2
  GE(2)= 0.0D0
```

```
  DE(1)=-2.0D0*X(1)*YA(1)
$ENDSET
$SET(FMODELY)
  FE=X(2)
$ENDSET
$SET(GMODELY)
  GE(1)=0.0D0
  GE(2)=1.0D0
$ENDSET
$MODELA='Y'
$NF=2
$NE=1
$MODEL='DQ'
$TOLR='1.0D-9'
$TOLA='1.0D-9'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of the variables x_1 and x_2 as well as the initial and terminal times 0 and T, respectively. The right hand side of the differential equation is specified by using the macrovariables $FMODELE and $GDMODELE, while the initial values and their derivatives are given by using the macrovariables $FMODELY and $GMODELY. The option $MODEL='DQ' together with $MODELA='Y' indicate a special integral criterion.

d) Problem solution (basic screen output):

```
 CLASS = GN - GM7   UPDATE = N   MODEL = DQ   HESF = D     NF =      2
     NIT=    0 NFV=    1 NFG=    1  F= 0.2500000000     G=0.693D+00
     NIT=    1 NFV=    3 NFG=    2  F= 0.3379696559E-01 G=0.114D+00
     NIT=    2 NFV=    5 NFG=    3  F= 0.1598937577E-02 G=0.613D-02
     NIT=    3 NFV=    7 NFG=    4  F= 0.1195750953E-04 G=0.225D-02
     NIT=    4 NFV=    9 NFG=    5  F= 0.1909017677E-08 G=0.300D-04
     NIT=    5 NFV=   11 NFG=    6  F= 0.2793082948E-15 G=0.200D-08
   0 NIT=    5 NFV=   11 NFG=    6  GRAD TOL  F= 0.2793082948E-15 G=0.200D-08
 F  =  0.2793082948D-15
 X  =  0.9999999725D+00  0.9999999990D+00
 TIME= 0:00:00.00
```

7.33 Non-stiff initial value problem for ordinary differential equations

a) Problem description:

Suppose we have to find a solution of a three-body problem

$$\frac{dy_1(t)}{dt} = y_3(t), \qquad y_1(0) = 0.9$$

$$\frac{dy_2(t)}{dt} = y_4(t), \qquad y_2(0) = 0$$

$$\frac{dy_3(t)}{dt} = y_1(t) + 2y_4(t) - \mu' \frac{y_1(t)+\mu}{D_1(t)} - \mu \frac{y_1(t)-\mu'}{D_2(t)}, \quad y_3(0) = 0$$

$$\frac{dy_4(t)}{dt} = y_2(t) - 2y_3(t) - \mu' \frac{y_2(t)}{D_1(t)} - \mu \frac{y_2(t)}{D_2(t)}, \qquad y_3(0) = -2$$

with $\mu = 0.012$, $\mu' = 1 - \mu$,

$$D1 = \left((y_1(t) + \mu)^2 + y_2(t)^2\right)^{3/2},$$

$$D2 = \left((y_1(t) - \mu')^2 + y_2(t)^2\right)^{3/2},$$

in the interval $0 \leq t \leq T$, where $T = 18$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SOLUTION OF NONSTIFF ORDINARY DIFFERENTIAL EQUATIONS          +
$REM +---------------------------------------------------------------+
$FLOAT W,W1,W2
$SET(INPUT)
  TA=0.0D0
  YA(1)= 0.9D0
  YA(2)= 0.0D0
  YA(3)= 0.0D0
  YA(4)=-2.0D0
  TAMAX= 1.8D1
$ENDSET
$SET(FMODELE)
  W=1.2D-2
  W1=(YA(1)+W)**2+YA(2)**2
  W1=W1*SQRT(W1)
  W2=(YA(1)+W-1.D0)**2+YA(2)**2
  W2=W2*SQRT(W2)
  GO TO (1,2,3,4) KE
1 FE=YA(3)
  GO TO 5
2 FE=YA(4)
  GO TO 5
3 FE=YA(1)+2*YA(4)+(W-1.D0)*(YA(1)+W)/W1-W*(YA(1)+W-1.D0)/W2
  GO TO 5
4 FE=YA(2)-2*YA(3)+(W-1.D0)*YA(2)/W1-W*YA(2)/W2
5 CONTINUE
$ENDSET
$NA=19
$NE=4
$MODEL='DE'
$MED=2
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable \$INPUT we specify the initial values of the variables y_1, y_2, y_3, y_4 as well as the initial and terminal times 0 and T, respectively. The right hand sides of the differential equations are specified by using the macrovariable \$FMODELE. The option \$MODEL='DE' indicates integration of a system of ordinary differential equations. Macrovariable \$NA denotes the number of time points (nodes) in which the results are printed.

d) Problem solution (basic screen output):

```
 CLASS = DE - DP8   UPDATE = N   MODEL = NO   HESF = N     NF =      0
    0  NSTP=   129  NACC=    99  NREJ=    29  NEV =  6296  NEG =     0
    1  AT=  0.000000000D+00
       AY=  0.900000000D+00  0.000000000D+00  0.000000000D+00 -0.200000000D+01
    2  AT=  0.100000000D+01
       AY= -0.472474065D+00 -0.831552587D+00 -0.169925866D+01  0.933246989D+00
    3  AT=  0.200000000D+01
       AY= -0.522790604D+00  0.777581853D+00  0.164553511D+01  0.102899578D+01
    4  AT=  0.300000000D+01
       AY=  0.860391993D+00 -0.437352811D-02 -0.262361582D-01 -0.198574759D+01
    5  AT=  0.400000000D+01
       AY= -0.545402744D+00 -0.728281933D+00 -0.158624165D+01  0.112488972D+01
    6  AT=  0.500000000D+01
       AY= -0.361844730D+00  0.857723938D+00  0.179231945D+01  0.743627310D+00
    7  AT=  0.600000000D+01
       AY=  0.900662074D+00 -0.136996194D+00 -0.190925673D+00 -0.195779058D+01
    8  AT=  0.700000000D+01
       AY= -0.560957109D+00 -0.823081310D+00 -0.162128701D+01  0.106071764D+01
    9  AT=  0.800000000D+01
       AY= -0.539311035D+00  0.804263515D+00  0.165764912D+01  0.100452367D+01
   10  AT=  0.900000000D+01
       AY=  0.852371731D+00  0.255993603D-01 -0.165348326D-01 -0.198405038D+01
   11  AT=  0.100000000D+02
       AY= -0.552637953D+00 -0.670747471D+00 -0.153518712D+01  0.120708594D+01
   12  AT=  0.110000000D+02
       AY= -0.236500690D+00  0.877334516D+00  0.186864292D+01  0.531341150D+00
   13  AT=  0.120000000D+02
       AY=  0.910238785D+00 -0.267656880D+00 -0.360816991D+00 -0.191426158D+01
   14  AT=  0.130000000D+02
       AY= -0.609930562D+00 -0.895015297D+00 -0.163634493D+01  0.104639816D+01
   15  AT=  0.140000000D+02
       AY= -0.713359518D+00  0.770343601D+00  0.150308593D+01  0.122606580D+01
   16  AT=  0.150000000D+02
       AY=  0.822862768D+00  0.266689821D+00  0.396871774D+00 -0.191636216D+01
   17  AT=  0.160000000D+02
       AY= -0.414633818D+00 -0.671205422D+00 -0.169537478D+01  0.101918630D+01
   18  AT=  0.170000000D+02
       AY= -0.172307075D+00  0.856357518D+00  0.188430339D+01  0.497722001D+00
   19  AT=  0.180000000D+02
       AY=  0.101278207D+01 -0.321748683D+00 -0.191710973D-01 -0.194769624D+01
 TIME= 0:00:00.02
```

7.34 Stiff initial value problem for ordinary differential equations

a) Problem description:

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

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

$$\frac{dy_1(t)}{dt} = (1 - y_1^2(t))y_2(t) - y_1(t), \quad y_2(0) = 0$$

in the interval $0 \leq t \leq T$ where $T = 20$.

b) Problem specification (input file):

```
$REM +---------------------------------------------------------------+
$REM + SOLUTION OF STIFF ORDINARY DIFFERENTIAL EQUATIONS             +
$REM +---------------------------------------------------------------+
$FLOAT W1,W2,W3,W4
$SET(INPUT)
  TA=0.0D0
  YA(1)=2.0D0
  YA(2)=-6.6D-1
  TAMAX=2.0D0
$ENDSET
$SET(FMODELE)
  W1=1.0D-6
  GO TO (1,2) KE
1 FE=YA(2)
  GO TO 3
2 FE=((1.0D0-YA(1)**2)*YA(2)-YA(1))/W1
3 CONTINUE
$ENDSET
$SET(DMODELE)
  W1=1.0D-6
  GO TO (4,5) KE
4 DE(1)=0.0D0
  DE(2)=1.0D0
  GO TO 6
5 DE(1)=(-2.0D0*YA(1)*YA(2)-1.0D0)/W1
  DE(2)=(1.0D0-YA(1)**2)/W1
6 CONTINUE
$ENDSET
$NA=301
$NE=2
$MODEL='DE'
$ODE='STIFF'
$TOLR='1.0D-4'
$TOLA='1.0D-8'
$MED=1
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of the variables y_1 and y_2 as well as the initial and terminal times 0 and T, respectively. The right hand sides of the differential equations are specified by using the macrovariable $FMODELE. The option $MODEL='DE' indicates integration of a system of ordinary differential equations. Macrovariable $NA denotes the number of time points (nodes) in which the results are printed.

d) Problem solution (basic screen output):

```
 CLASS = DE - RD5   UPDATE = N   MODEL = NO   HESF = N     NF =      0
    0  NSTP=   345  NACC=   290  NREJ=    38  NEV =  5532  NEG =   374
    1  AT=  0.000000000D+00
       AY=  0.200000000D+01 -0.660000000D+00
    2  AT=  0.100000000D-06
       AY=  0.199999993D+01 -0.661727883D+00
    3  AT=  0.200000000D-06
       AY=  0.199999987D+01 -0.663007940D+00
    4  AT=  0.656175225D-06
       AY=  0.199999956D+01 -0.665734985D+00
    5  AT=  0.126092836D-05
       AY=  0.199999916D+01 -0.666514522D+00
    6  AT=  0.218147783D-05
       AY=  0.199999855D+01 -0.666657189D+00
    7  AT=  0.375735133D-05
       AY=  0.199999750D+01 -0.666667678D+00
    8  AT=  0.804409324D-05
       AY=  0.199999464D+01 -0.666669509D+00
    9  AT=  0.292949166D-04
       AY=  0.199998047D+01 -0.666677397D+00
   10  AT=  0.199301503D-03
       AY=  0.199986713D+01 -0.666740383D+00
   ----------------------------------------
  281  AT=  0.161429359D+01
       AY=  0.200007205D+01 -0.666537135D+00
  282  AT=  0.161429463D+01
       AY=  0.200007136D+01 -0.666622471D+00
  283  AT=  0.161429652D+01
       AY=  0.200007010D+01 -0.666627481D+00
  284  AT=  0.161430239D+01
       AY=  0.200006618D+01 -0.666629772D+00
  285  AT=  0.161433549D+01
       AY=  0.200004412D+01 -0.666642040D+00
  286  AT=  0.161460030D+01
       AY=  0.199986757D+01 -0.666740131D+00
  287  AT=  0.161671877D+01
       AY=  0.199845427D+01 -0.667526465D+00
  288  AT=  0.163366649D+01
       AY=  0.198708726D+01 -0.673925875D+00
  289  AT=  0.173214063D+01
       AY=  0.191874076D+01 -0.715528198D+00
  290  AT=  0.193035891D+01
       AY=  0.176619569D+01 -0.833326803D+00
  291  AT=  0.200000000D+01
       AY=  0.170616796D+01 -0.892808294D+00
 TIME= 0:00:00.03
```

7.35 Minimization with complementarity constraints (sparse Hessian matrix)

a) Problem description:

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

$$F(x) = x_1^2 - 2x_1 + x_2^2 - 2x_2 + x_5^2 + x_6^2,$$

over the set given by the simple constraints $x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, $x_4 \geq 0$, nonlinear constraints

$$\begin{aligned} 2x_5 - 2x_1 + 2(x_5 - 1)x_3 &= 0 \\ 2x_6 - 2x_2 + 2(x_6 - 1)x_4 &= 0 \\ 0.25 - (x_5 - 1)^2 &\geq 0 \\ 0.25 - (x_6 - 1)^2 &\geq 0, \end{aligned}$$

and complementrity constraints

$$\begin{aligned} x_3(0.25 - (x_5 - 1)^2) &= 0 \\ x_4(0.25 - (x_6 - 1)^2) &= 0. \end{aligned}$$

The starting point is $x_i = 0,\ 1 \leq i \leq n$.

b) Problem specification (input file):

```
$REM +-----------------------------------------------------------+
$REM + SPARSE MINIMIZATION WITH COMPLEMENTARITY CONSTRAINTS       +
$REM +-----------------------------------------------------------+
$ADD(INTEGER,'\I')
$SET(INPUT)
  IX(1)=1; XL(1)=0.0D0
  IX(2)=1; XL(2)=0.0D0
  IX(3)=1; XL(3)=0.0D0
  IX(4)=1; XL(4)=0.0D0
  DO 1 I=1,NF
1 CONTINUE
  IC(1)=5; CL(1)=0.0D0
  IC(2)=5; CL(2)=0.0D0
  IC(3)=1; CL(3)=0.0D0
  IC(4)=1; CL(4)=0.0D0
  ICC(1)= 3;
  ICC(2)= 4
  ICC(3)=-3;
  ICC(4)=-4
  DO 2 I=1,NF
    IH(I)=I; JH(I)=I
2 CONTINUE
  IH(7)=7
  ICG(1)=1
  ICG(2)=4
  ICG(3)=7
  ICG(4)=8
  ICG(5)=9
  JCG(1)=1
  JCG(2)=3
```

```
  JCG(3)=5
  JCG(4)=2
  JCG(5)=4
  JCG(6)=6
  JCG(7)=5
  JCG(8)=6
  MC=8
$ENDSET
$SET(FMODELF)
  FF=X(1)**2-2.0D0*X(1)+X(2)**2-2.0D0*X(2)+X(5)**2+X(6)**2
$ENDSET
$SET(GMODELF)
  GF(1)=2.0D0*X(1)-2.0D0
  GF(2)=2.0D0*X(2)-2.0D0
  GF(5)=2.0D0*X(5)
  GF(6)=2.0D0*X(6)
$ENDSET
$SET(FMODELC)
  IF (KC.EQ.1) THEN
    FC=2.0D0*X(5)-2.0D0*X(1)+2.0D0*(X(5)-1.0D0)*X(3)
  ELSEIF (KC.EQ.2) THEN
    FC=2.0D0*X(6)-2.0D0*X(2)+2.0D0*(X(6)-1.0D0)*X(4)
  ELSEIF (KC.EQ.3) THEN
    FC=2.5D-1-(X(5)-1.0D0)**2
  ELSEIF (KC.EQ.4) THEN
    FC=2.5D-1-(X(6)-1.0D0)**2
  ENDIF
$ENDSET
$SET(GMODELC)
  IF (KC.EQ.1) THEN
    GC(1)=-2.0D0
    GC(3)=2.0D0*(X(5)-1.0D0)
    GC(5)=2.0D0+2.0D0*X(3)
  ELSEIF (KC.EQ.2) THEN
    GC(2)=-2.0D0
    GC(4)=2.0D0*(X(6)-1.0D0)
    GC(6)=2.0D0+2.0D0*X(4)
  ELSEIF (KC.EQ.3) THEN
    GC(5)=-2.0D0*(X(5)-1.0D0)
  ELSEIF (KC.EQ.4) THEN
    GC(6)=-2.0D0*(X(6)-1.0D0)
  ENDIF
$ENDSET
$NF=6
$NX=4
$NC=4
$NCC=2
$HESF='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Hessian matrix, the sparsity pattern of the constraint Jacobian matrix, and the specification of standard and complementarity constraints. The sparse Hessian matrix, indicated by the statement $HESF='S', is diagonal. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELF we specify analytically the value of the model function. By using the macrovariable $GMODELF we specify analytically the gradient of the model function. By using the macrovariable $FMODELC we specify analytically the values of the constraint functions. By using the macrovariable $GMODELC we specify analytically the gradients of the constraint functions. Macrovariable $NCC specifies the number of complementarity constraints.

d) Problem solution (basic screen output):

```
 CLASS = MN - LC3   UPDATE = N   MODEL = FF   HESF = S     NF =      6
     NIC=  0 NIT=   0 NFV=    1 NFG=    4 F=0.210D+00 C=0.750D+00 G=0.490D+01
     NIC=  0 NIT=   1 NFV=    3 NFG=    8 F=0.152D+02 C=0.926D+00 G=0.337D+01
     NIC=  0 NIT=   2 NFV=    4 NFG=   12 F=0.823D+01 C=0.562D+00 G=0.100D+01
     NIC=  0 NIT=   3 NFV=    5 NFG=   16 F=-.372D+00 C=0.122D+00 G=0.389D+00
     NIC=  0 NIT=   4 NFV=    6 NFG=   20 F=-.946D+00 C=0.198D+00 G=0.163D+00
     NIC=  0 NIT=   5 NFV=    7 NFG=   24 F=-.981D+00 C=0.702D-02 G=0.276D-01
     NIC=  0 NIT=   6 NFV=    8 NFG=   28 F=-.994D+00 C=0.903D-04 G=0.463D-02
     NIC=  0 NIT=   7 NFV=    9 NFG=   32 F=-.999D+00 C=0.696D-05 G=0.133D-02
     NIC=  0 NIT=   8 NFV=   10 NFG=   36 F=-.998D+00 C=0.978D-06 G=0.284D-03
     NIC=  0 NIT=   9 NFV=   11 NFG=   40 F=-.100D+01 C=0.607D-06 G=0.966D-04
     NIC=  0 NIT=  10 NFV=   12 NFG=   44 F=-.100D+01 C=0.185D-08 G=0.240D-04
     NIC=  0 NIT=  11 NFV=   13 NFG=   48 F=-.100D+01 C=0.244D-08 G=0.511D-05
     NIC=  0 NIT=  12 NFV=   14 NFG=   52 F=-.100D+01 C=0.156D-08 G=0.175D-05
     NIC=  0 NIT=  13 NFV=   15 NFG=   56 F=-.100D+01 C=0.100D-11 G=0.438D-06
     NIC=  0 NIT=  14 NFV=   16 NFG=   60 F=-.100D+01 C=0.576D-11 G=0.942D-07
     NIC=  0 NIT=  15 NFV=   17 NFG=   64 F=-.100D+01 C=0.360D-11 G=0.316D-07
     NIC=  0 NIT=  16 NFV=   18 NFG=   68 F=-.100D+01 C=0.155D-14 G=0.788D-08
     NIC=  0 NIT=  17 NFV=   19 NFG=   72 F=-.100D+01 C=0.179D-13 G=0.162D-08
     NIC=  0 NIT=  18 NFV=   20 NFG=   76 F=-.100D+01 C=0.117D-13 G=0.592D-09
     NIC=  0 NIT=  19 NFV=   21 NFG=   80 F=-.100D+01 C=0.129D-15 G=0.148D-09
     NIC=  0 NIT=  20 NFV=   22 NFG=   84 F=-.100D+01 C=0.104D-15 G=0.332D-10
   0 NIT=  21 NFV=   22 NFG=   84 F=-0.10000D+01 C=0.1D-15 P=0.2D-16 G=0.3D-10
 FF = -0.1000000000D+01
 X  =  0.5000022653D+00  0.5000020861D+00  0.7132758014D-12  0.7846083046D-11
       0.5000022653D+00  0.5000020861D+00
 TIME= 0:00:00.00
```

7.36 Large-scale least squares optimization with complementarity constraints

a) Problem description:

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

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

where

$$f_i^A = (2+5x_i^2)x_i + 1 + \sum_{j=\max(1,i-5)}^{i+1} x_j(1+x_j), \quad 1 \le i \le n-1,$$

and $n = 101$, over the set given by the nonlinear constraints

$$f_i^C(x) = 4x_{2i} - (x_{2i-1} - x_{2i+1}) \exp(x_{2i-1} - x_{2i} - x_{2i+1}) - 3 \geq 0,$$

$1 \leq i \leq n_C$, where $n_C = 50$, and the complementarity constraints

$$f_i^C(x) f_{i+n_c/2}^C(x) = 0, \quad 1 \leq i \leq n_C/2.$$

The starting point is $x_i = 3$ for $1 \leq i \leq n$.

b) Problem specification (input file):

```
$REM +-----------------------------------------------------------------+
$REM + SPARSE SUM OF SQUARES WITH COMPLEMENTARITY CONSTRAINTS         +
$REM +-----------------------------------------------------------------+
$ADD(INTEGER,'\I\K')
$SET(INPUT)
  DO 1 I=1,NF
   X(I)=3.0D0
1 CONTINUE
  MA=1
  DO 3 KA=1,NA
  IAG(KA)=MA
  DO 2 I=5,1,-1
  IF (KA.GT.I) THEN
  JAG(MA)=KA-I
  MA=MA+1
  ENDIF
2 CONTINUE
  JAG(MA)=KA
  JAG(MA+1)=KA+1
  MA=MA+2
3 CONTINUE
  IAG(NA+1)=MA
  MA=MA-1
  MC=1
  DO 4 KC=1,NC
  IC(KC)=1; CL(KC)=0.0D0
  ICC(KC)=KC
  ICG(KC)=MC
  I=2*KC
  JCG(MC)=I-1
  JCG(MC+1)=I
  JCG(MC+2)=I+1
  MC=MC+3
4 CONTINUE
  ICG(NC+1)=MC
  MC=MC-1
$ENDSET
$SET(FMODELA)
  FA=(2.0D0+5.0D0*X(KA)**2)*X(KA)+1.0D0
  DO 5 I=MAX(1,KA-5),MIN(NF,KA+1)
  FA=FA+X(I)*(1.0D0+X(I))
5 CONTINUE
```

```
$ENDSET
$SET(FMODELC)
  K=2*KC
  FC=4.0$P 0*X(K)-(X(K-1)-X(K+1))*EXP(X(K-1)-X(K)-X(K+1))-3.0D0
$ENDSET
$IADA=1
$IADC=1
$NF=101
$NA=100
$NC=50
$NCC=25
$MA=800
$MC=800
$MODEL='AQ'
$JACA='S'
$JACC='S'
$BATCH
$STANDARD
```

c) Comments on the problem specification:

By using the macrovariable $INPUT we specify the initial values of variables, the sparsity pattern of the objective Jacobian matrix, the sparsity pattern of the constraint Jacobian matrix and the specification of standard and complementarity constraints. The sparse objective Jacobian matrix is indicated by the statement $JACA='S'. The sparse constraint Jacobian matrix is indicated by the statement $JACC='S'. By using the macrovariable $FMODELA we specify values of the approximating function analytically. The gradients of the approximating function are computed by automatic differentiation, since $IADA=1. By using the macrovariable $FMODELC we specify the values of the constraint functions analytically. The gradients of the constraint functions are computed by automatic differentiation, since $IADC=1. For the sum-of-squares minimization we set $MODEL='AQ'. Macrovariable $NCC specifies the number of complementarity constraints.

d) Problem solution (basic screen output):

```
 CLASS = MN - LC3   UPDATE = N   MODEL = AQ   HESF = S     NF =    101
     NIC=  0 NIT=   0 NFV=   27 NFG=   27 F=0.252D+07 C=0.000D+00 G=0.420D+05
     NIC=  0 NIT=   1 NFV=   55 NFG=   55 F=0.674D+06 C=0.000D+00 G=0.139D+05
     NIC=  0 NIT=   2 NFV=   83 NFG=   83 F=0.182D+06 C=0.000D+00 G=0.457D+04
     NIC=  0 NIT=   3 NFV=  111 NFG=  111 F=0.500D+05 C=0.000D+00 G=0.151D+04
     NIC=  0 NIT=   4 NFV=  139 NFG=  139 F=0.139D+05 C=0.000D+00 G=0.505D+03
     NIC=  0 NIT=   5 NFV=  167 NFG=  167 F=0.519D+04 C=0.346D-01 G=0.178D+03
     NIC=  0 NIT=   6 NFV=  195 NFG=  195 F=0.279D+04 C=0.000D+00 G=0.674D+02
     NIC=  0 NIT=   7 NFV=  223 NFG=  223 F=0.184D+04 C=0.384D-02 G=0.262D+02
     NIC=  0 NIT=   8 NFV=  251 NFG=  251 F=0.163D+04 C=0.191D-01 G=0.168D+02
     NIC=  0 NIT=   9 NFV=  279 NFG=  279 F=0.159D+04 C=0.523D-02 G=0.255D+01
     NIC=  0 NIT=  10 NFV=  307 NFG=  307 F=0.159D+04 C=0.125D-03 G=0.142D+00
     NIC=  0 NIT=  11 NFV=  335 NFG=  335 F=0.159D+04 C=0.683D-04 G=0.126D-01
     NIC=  0 NIT=  12 NFV=  363 NFG=  363 F=0.159D+04 C=0.374D-06 G=0.349D-03
     NIC=  0 NIT=  13 NFV=  391 NFG=  391 F=0.159D+04 C=0.486D-09 G=0.166D-06
     NIC=  0 NIT=  14 NFV=  419 NFG=  419 F=0.159D+04 C=0.000D+00 G=0.853D-13
     NIC=  0 NIT=  15 NFV=  447 NFG=  447 F=0.159D+04 C=0.000D+00 G=0.568D-13
   0 NIT=  16 NFV=  447 NFG=  447 F= 0.15927D+04 C=0.0D+00 P=0.2D-28 G=0.6D-13
```

8 Model examples for demonstration of graphic outputs

Here we introduce several problem specifications (input files) which demonstrate the application of the graphic screen output. The graphic screen output can be used only on PC computers under the MS DOS system. This possibility is not allowed on the UNIX workstations.

The input files are included into the UFO system as demo-files `PROC01.UFO`,... ,`PROC08.UFO`. Corresponding graphic pictures are included in the Appendix D. The data recommended for graphic pictures are introduced in lines which begin by the directive $REM.

8.1 Nonlinear regression

```
$SET(INPUT)
  LDIM=5
  X(1)=7.0D20
  X(2)=1.0D4
  X(3)=2.2D0
  X(4)=1.01D0
  X(5)=7.0D17
  X(6)=7.0D3
  X(7)=1.6D0
  X(8)=1.01D0
  X(9)=1.0D16
  X(10)=4.0D3
  X(11)=1.5D0
  X(12)=1.01D0
  X(13)=2.0D15
  X(14)=4.0D3
  X(15)=1.3D0
  X(16)=1.01D0
  X(17)=1.0D16
  X(18)=5.0D2
  X(19)=1.2D0
  X(20)=1.01D0
  BETA=5.95D0
  CALL BIUD01(NF,LDIM,NA,X,XL,XU,IX,AT,AM)
$ENDSET
$SET(FMODELA)
  CALL BAFU01(NF,LDIM,KA,NA,X,AT,FA,BETA)
$ENDSET
$SET(GMODELA)
  CALL BAGU01(NF,LDIM,KA,NA,X,AT,GA,BETA)
$ENDSET
$NF=30
$NA=500
$MIT=100
$MODEL='AQ'
$CLASS='GN'
$TYPE='G'
$DECOMP='M'
$NUMBER=7
$UPDATE='F'
$TOLX='1.0$P-16'
```

```
$TOLF='1.0$P-16'
$TOLB='1.0$P-16'
$TOLG='1.0$P-6'
$KBA=1
$KBF=2
$GRAPH='Y'
$SCAN='Y'
$BATCH
$ADD(REAL,'\BETA\AT($NA)')
$ADD(SUBROUTINES)
      SUBROUTINE BIUD01(N,L,NA,X,XL,XU,IX,AT,AM)
      INTEGER N,L,NA,IX(N),I,K
      REAL*8 X(N),XL(N),XU(N),AT(NA),AM(NA)
      N=4*L
      K=0
      DO 1 I=1,L
      X(K+1)=LOG(X(K+1))
      XL(K+1)=LOG(1.0D+0)
      XU(K+1)=LOG(1.0D+40)
      IX(K+1)=3
      X(K+2)=LOG(X(K+2))
      XL(K+2)=LOG(1.0D+0)
      XU(K+2)=LOG(1.0D+10)
      IX(K+2)=3
      XL(K+3)=1.0D-2
      XU(K+3)=1.0D+2
      IX(K+3)=3
      XL(K+4)=1.00001D0
      XU(K+4)=1.00000D1
      IX(K+4)=3
      K=K+4
    1 CONTINUE
      OPEN (11,FILE='PROC01.DAT',STATUS='OLD')
      NA=0
    2 NA=NA+1
      READ (11,'(2D14.6)',ERR=3) AT(NA),AM(NA)
      GO TO 2
    3 NA=NA-1
      RETURN
      END
      SUBROUTINE BAFU01(N,L,KA,NA,X,AT,FA,BETA)
      INTEGER N,L,KA,NA
      REAL*8 X(N),AT(NA),FA,Q(8),QD(8)
      REAL*8 ARG,POM,BK,B6INT,BETA
      INTEGER J,K
      COMMON /BCOM/ Q,QD
      DATA BK /8.617385D-5/
      FA=0.0D 0
      K=0
      DO 1 J=1,L
      ARG=X(K+3)/(BK*AT(KA))
      IF (KA.EQ.1) THEN
```

```
  Q(J)=B6INT(AT(KA),ARG)
  FA=FA+EXP(X(K+1)+X(K+2)-ARG)
  ELSE
  POM=X(K+4)-1.0D0
  FA=FA+EXP(X(K+1)+X(K+2)-ARG)*
 & (1.0D0+(POM/BETA)*EXP(X(K+1))*(B6INT(AT(KA),ARG)-
 & Q(J)))**(-X(K+4)/POM)
  ENDIF
  K=K+4
1 CONTINUE
  RETURN
  END
  SUBROUTINE BAGU01(N,L,KA,NA,X,AT,GA,BETA)
  INTEGER N,L,KA,NA
  REAL*8 X(N),AT(NA),GA(N)
  REAL*8 FAC,ARG,POM,POW,BK,B6INT,B6INTD,A,B,C,D,E,F,G
  REAL*8 Q(8),QD(8),QQ,QQD,BETA
  INTEGER J,K
  COMMON /BCOM/ Q,QD
  DATA BK /8.617385D-5/
  K=0
  DO 1 J=1,L
  FAC=1.0D0/(BK*AT(KA))
  ARG=FAC*X(K+3)
  IF (KA.EQ.1) THEN
  Q(J)=B6INT(AT(KA),ARG)
  QD(J)=FAC*B6INTD(AT(KA),ARG)
  QQ=0.0D0
  QQD=0.0D0
  ELSE
  QQ=B6INT(AT(KA),ARG)-Q(J)
  QQD=FAC*B6INTD(AT(KA),ARG)-QD(J)
  ENDIF
  POM=X(K+4)-1.0D0
  POW=-X(K+4)/POM
  A=EXP(X(K+1)+X(K+2)-ARG)
  B=EXP(X(K+1))
  G=B*QQ
  C=(1.0D0+(POM/BETA)*G)
  D=C**POW
  E=POW*D/C
  F=POM*POM
  GA(K+1)=A*(D+E*(POM/BETA)*G)
  GA(K+2)=A*D
  GA(K+3)=A*(-FAC*D+E*(POM/BETA)*B*QQD)
  GA(K+4)=A*D*(LOG(C)/F+POW*G/(C*BETA))
  K=K+4
1 CONTINUE
  RETURN
  END
```

```
      FUNCTION B6INT(T,X)
      REAL*8 T,X,B6INT
      REAL*8 A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6
      DATA A1,A2,A3,A4,A5,A6 /41.0D+0, 590.0D+0, 3648.0D+0,
     &                      9432.0D+0, 8028.0D+0, 720.0D+0/
      DATA B1,B2,B3,B4,B5,B6 /42.0D+0, 630.0D+0, 4200.0D+0,
     &                   12600.0D+0, 15120.0D+0, 5040.0D+0/
      B6INT=(1.0D0-(A6+X*(A5+X*(A4+X*(A3+X*(A2+X*(A1+X))))))/
     &     (B6+X*(B5+X*(B4+X*(B3+X*(B2+X*(B1+X)))))))*EXP(-X)*T
      RETURN
      END
      FUNCTION B6INTD(T,X)
      REAL*8 T,X,B6INTD
      REAL*8 A1,A2,A3,A4,A5,A6,B1,B2,B3,B4,B5,B6
      REAL*8 C1,C2,C3,C4,C5,D1,D2,D3,D4,D5,DIS,DEN,DISD,DEND
      DATA A1,A2,A3,A4,A5,A6 /41.0D+0, 590.0D+0, 3648.0D+0,
     &                      9432.0D+0, 8028.0D+0, 720.0D+0/
      DATA B1,B2,B3,B4,B5,B6 /42.0D+0, 630.0D+0, 4200.0D+0,
     &                   12600.0D+0, 15120.0D+0, 5040.0D+0/
      DATA C1,C2,C3,C4,C5 /205.0D+0, 2360.0D+0, 10944.0D+0,
     &                                18863.0D+0, 8028.0D+0/
      DATA D1,D2,D3,D4,D5 /210.0D+0, 2520.0D+0, 12600.0D+0,
     &                              25200.0D+0, 15120.0D+0/
      DIS=A6+X*(A5+X*(A4+X*(A3+X*(A2+X*(A1+X)))))
      DEN=B6+X*(B5+X*(B4+X*(B3+X*(B2+X*(B1+X)))))
      DISD=C5+X*(C4+X*(C3+X*(C2+X*(C1+6.0D0*X))))
      DEND=D5+X*(D4+X*(D3+X*(D2+X*(D1+6.0D0*X))))
      B6INTD=((DIS-DISD+DEND*DIS/DEN)/DEN-1.0D0)*EXP(-X)*T
      RETURN
      END
$ENDADD
$STANDARD
```

8.2 Nonlinear minimax optimization

```
$FLOAT W
$SET(INPUT)
  X(1)=0.5D0 ; X(2)=0.0D0 ; X(3)=0.0D0
  X(4)=0.0D0 ; X(5)=0.0D0
$ENDSET
$SET(FMODELA)
  W=0.1D0*DBLE(KA-1)-1.0D0
  FA=(X(1)+W*X(2))/(1.0D0+W*(X(3)+W*(X(4)+W*X(5))))-EXP(W)
$ENDSET
$MODEL='AM'
$NF=5
$NA=21
$NAL=0
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
```

```
$PATH='E'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=3, XL=-5, XU=5
```

8.3 Transformer network design

```
$SET(INPUT)
  NEXT=4
  CALL EIUD06(NF,NA,NAL,X,FMIN,XMAX,NEXT,IEXT,IERR)
$ENDSET
$SET(FMODELA)
  CALL EAFU06(NF,KA,X,FA,NEXT)
$ENDSET
$SET(GMODELA)
  CALL EAGU06(NF,KA,X,GA,NEXT)
$ENDSET
$NF=6
$NA=11
$NAL=0
$MOUT=1
$MODEL='AM'
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$PATH='E'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=3, XL=-5, XU=5
```

8.4 Global optimization

```
$SET(INPUT)
  NEXT=4
  CALL EIUD09(NF,XL,XU,NEXT,IERR)
$ENDSET
$SET(FMODELF)
  CALL EFFU09(NF,X,FF,NEXT)
$ENDSET
$NF=4
$MOUT=1
$GCLASS=1
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$EXTREM='G'
```

```
$BATCH
$STANDARD

$REM VAR=1, XL=-3.8, XU=3.8
$REM VAR=2, XL=-3.8, XU=3.8
```

8.5 Nonsmooth optimization

```
$SET(INPUT)
  NEXT=15
  CALL EIUD19(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
  MA=NF+3
$ENDSET
$SET(FMODELF)
  CALL EFFU19(NF,X,FF,NEXT)
$ENDSET
$SET(GMODELF)
  CALL EFGU19(NF,X,GF,NEXT)
$ENDSET
$KSF=3
$NF=30
$MOUT=-1
$MODEL='FF'
$GRAPH='Y'
$MAP='Y'
$HIL='Y'
$ISO='Y'
$PATH='Y'
$BATCH
$STANDARD

$REM VAR=1, XL=-5, XU=5
$REM VAR=4, XL=-5, XU=5
```

8.6 The Rosenbrock function

```
$SET(INPUT)
  X(1)=-1.2D0
  X(2)= 1.0D0
$ENDSET
$SET(FMODELF)
  FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
$ENDSET
$NF=2
$GRAPH='Y'
$MAP='Y'
$ISO='Y'
$PATH='Y'
$BATCH
$STANDARD
```

8.7 Ordinary differential equations

```
$FLOAT W1,W2,W3,W4
$SET(INPUT)
  TA=0.0D0
  YA(1)=0.994D0
  YA(2)=0.0D0
  YA(3)=0.0D0
  YA(4)=-2.00158510637908252240537862224D0
  TAMAX=17.06521656015796255889172062490D0
$ENDSET
$SET(FMODELE)
  W1=0.012277471D0
  W2=1.D0-W1
  W3=(YA(1)+W1)**2+YA(2)**2
  W3=W3*SQRT(W3)
  W4=(YA(1)-W2)**2+YA(2)**2
  W4=W4*SQRT(W4)
  GO TO (1,2,3,4) KE
1 FE=YA(3)
  GO TO 5
2 FE=YA(4)
  GO TO 5
3 FE=YA(1)+2*YA(4)-W2*(YA(1)+W1)/W3-W1*(YA(1)-W2)/W4
  GO TO 5
4 FE=YA(2)-2*YA(3)-W2*YA(2)/W3-W1*YA(2)/W4
5 CONTINUE
$ENDSET
$NE=4
$NA=2000
$MODEL='DE'
$SOLVER='DP5'
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$MED=1
$GRAPH='Y'
$BATCH
$STANDARD
```

8.8 The Lorenz attractor

```
$FLOAT W1,W2,W3
$SET(INPUT)
  W1=10.0D0
  W2=28.0D0
  W3=8.0D0/3.0D0
  TA=0.0D0
  YA(1)=-8.0D0
  YA(2)= 8.0D0
  YA(3)=W2-1.0D0
  TAMAX=50.0D0
$ENDSET
```

```
$SET(FMODELE)
  GO TO (1,2,3) KE
1 FE=-W1*YA(1)+W1*YA(2)
  GO TO 4
2 FE=-YA(1)*YA(3)+W2*YA(1)-YA(2)
  GO TO 4
3 FE=YA(1)*YA(2)-W3*YA(3)
4 CONTINUE
$ENDSET
$NE=3
$NA=2000
$MODEL='D'
$SOLVER='DP8'
$TOLR='1.0$P-9'
$TOLA='1.0$P-9'
$MED=1
$GRAPH='Y'
$BATCH
$STANDARD
```

References

[1] M.Al-Baali, R.Fletcher: Variational methods for nonlinear least squares. J. Optimizaton Theory and Applications 36 (1985) 405-421.

[2] M.Al-Baali: Descent property and global convergence of the Fletcher-Reeves method with inexact line search. IMA J. Numerical Analysis 5 (1985) 121-124.

[3] M.Altman: Generalized gradient methods of minimizing a functional. Bull. Acad. Polon. Sci., Ser. Sci. Math. Astronom. Phys. 14 (1966) 313-318.

[4] N.Andrei: Scaled conjugate gradient algorithms for unconstrained optimization. Computational Optimization and Applications 38 (2007) 401-416.

[5] N.Andrei: Another hybrid conjugate gradient algorithm for unconstrained optimization. Numerical Algorithms 47 (2008) 143-156.

[6] N.Andrei: A Dai-Yuan conjugate gradient algorithm with sufficient descent and conjugacy conditions for unconstrained optimization. Applied Mathematics Letters 21 (2008) 165-171.

[7] N.Andrei: Unconstained optimization test functions. Research Institute for Informatics, Center for Advanced Modelling and Optimization, Bucharest, Romania. (2008) 143-156.

[8] M.Argaez, R.A.Tapia, L.Velasquez: Numerical comparison of path-following strategies for a basic interior-point method for nonlinear programming. Report No. CRPC-TR9777-S, Center for Research on Parallel Computation, Rice University, Houston 1998.

[9] P.Armand: Modification of the Wolfe line search rule to satisfy the descent conditions in the Polak-Ribiere-Polyak conjugate gradient method. J. Optimization Theory and Applications 132 (2007) 287-305.

[10] L.Armijo: Minimization of functions having continuous partial derivatives. Pacific J. Math. 16 (1966) 1-3.

[11] E.M.L.Beale: A derivative of conjugate gradients. In: Numerical Methods for Nonlinear Optimization (F.A.Lootsma, ed.), Academic Press, London, 1972 39-43.

[12] M.Bertocchi, E.Spedicato: Computational experience with conjugate gradient algorithms. Estratto da Calcolo 16 (1979) 255-269.

[13] 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.

[14] M.C.Biggs: Minimization algorithms making use of nonquadratic properties of the objective function. J. Inst. math. Appl. 8 (1971) 315-327.

[15] M.C.Biggs:

[16] E.G.Birgin, J.M.Martinez: A spectral conjugate gradient method for unconstrained optimization. Applied Mathematics and Optimization 43 (2001) 117-128.

[17] C.G.Bischof: Incremental condition estimation. SIAM J. Matrix Analysis and Applications 11 (1990) 312-322.

[18] A.Björck: Numerical Methods in Matrix Computations. Springer International Publishing, Switzerland 2015.

[19] P.Bjorstadt, J.Nocedal: Analysis of a new algorithm for one-dimensional minimization. Computing 22 (1979) 93-100.

[20] C.G.E.Boender, A.H.G.Rinnoy Kan: Bayessian stopping rules for multistart global optimization methods. Math. Programming 37 (1987) 59-80.

[21] 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.

[22] P.T.Boggs, J.W.Tolle: A strategy for global convergence in a sequential quadratic programming algorithm. SIAM Journal on Numerical Analysis 26 (1989) 600-623.

[23] P.T.Boggs, J.W.Tolle: Sequential quadratic programming. Acta Numerica 4 (1995) 1-51.

[24] I.D.L.Bogle, J.D.Perkins: A New Sparsity Preserving Quasi-Newton Update for Solving Nonlinear Equations. SIAM Journal on Scientific and Statistical Computations 11 (1990) 621-630.

[25] I.Bongartz, A.R.Conn, N. Gould, P.L.Toint: CUTE: constrained and unconstrained testing environment. Report.

[26] J.M.Bennet: Triangular factors of modified matrices. Numerische Mathematik 7 (1965) 217-221.

[27] R.P.Brent: Some efficient algorithms for solving systems of nonlinear equations. SIAM Journal on Numerical Analysis 10 (1973) 327-344.

[28] C.G.Broyden: A class of methods for solving nonlinear simultaneous equations. Math. of Comput. 19 (1965) 577-593.

[29] 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.

[30] 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.

[31] A.G.Buckley: Extending the relationship between the conjugate gradient and BFGS algorithms. Mathematical Programming 15 (1978) 343-348.

[32] A.G.Buckley: A combined conjugate-gradient quasi-Newton minimization algorithm, Mathematical Programming 15 (1978) 200-210.

[33] A.G.Buckley: Conjugate gradient methods. In: Nonlinear Optimization 1981 (M.J.D.Powell, ed.), Academic Press, London, 1982 17-22.

[34] J.R.Bunch, B.N. Parlett: Direct methods for solving symmetric indefinite systems of linear equations. SIAM J. Numer. Anal. 8 (1971) 639-655.

[35] J.R.Bunch, Kaufmann: Some stable methods for calculating inertia and solving symmetric linear systems. Mathematics of Computation 137 (1977) 163-179.

[36] R.H.Byrd, J.C.Gilbert, J.Nocedal: A trust region method based on interior point techniques for nonlinear programming. Mathematical Programming 89 (2000) 149-185.

[37] R.H.Byrd, M.E.Hribar, J.Nocedal: An interior point algorithm for large scale nonlinear programming. SIAM J. Optimization 9 (1999) 877-900.

[38] R.H.Byrd, G.Liu, J.Nocedal: On the local behavior of an interior point method for nonlinear programming. In Numerical Analysis 1997 (D.F.Griffiths and D.J.Higham, eds.), pp.37-56. Addison Wesley Longman, 1998.

[39] R.H.Byrd, J.Nocedal, R.B.Schnabel: Representation of quasi-Newton matrices and their use in limited memory methods. Mathematical Programming 63 (1994) 129-156.

[40] R.H.Byrd, J.Nocedal, R.A.Waltz: Feasible interior methods using slacks for nonlinear optimization. Report No. OTC 2000/11, Optimization Technology Center, December 2000.

[41] 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.

[42] C.Cartis, N.I.M.Gould, P.L.Toint: Adaptive cubic regularisation methods for unconstrained optimization. Part I: motivation, convergence and numerical results. Mathematical Programming 127 (2011) 245295.

[43] C.Cartis, N.I.M.Gould, P.L.Toint: Adaptive cubic regularisation methods for unconstrained optimization. Part II: worst-case function and derivative evaluation complexity. Mathematical Programming 130 (2011) 295319.

[44] T.F.Chan: Rank revealing QR factorizations. Linear Algebra Appl. 88/89 (1987) 67-82.

[45] X.Chen, J.Sun: Global convergence of a two-parameter family of conjugate gradient methods without line-search. J. of Computational and Applied Mathematics 146 (2002) 37-45.

[46] W.Cheng: Spectral scaling BFGS method. J. Optimizaton Theory and Applications 146 (2010) 305-319.

[47] W.Cheng, Y.Xiao, Q.Hu: A family of derivative-free conjugate gradient methods for large scale nonlinear systems of equations. J. of Computational and Applied Mathematics 224 (2009) 11-19.

[48] A.Cohen: Rate of convergence of several conjugate gradient algorithms. SIAM J. Numerical Analysis 9 (1972) 248-259.

[49] T.F.Coleman: Large sparse numerical optimization. Springer-Verlag, Berlin, 1984.

[50] T.F.Coleman, B.S.Garbow, J.S.Moré: Software for estimation sparse Hessian matrices. ACM Trans. of Math. Software 11 (1985) 363-367.

[51] T.F.Coleman, B.S.Garbow, J.S.Moré: Software for estimating sparse Jacobian matrices. ACM Trans. of Math. Software 10 (1984) 329-345.

[52] A.R.Conn, N. Gould, P.L.Toint: LANCELOT. A Fortran Package for Large-Scale Nonlinear Optimization. Springer Verlag, Berlin 1992.

[53] 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.

[54] H.P.Crowder and P.Wolfe: Linear convergence of the conjugate gradient method. IBM J. Res. Dev. 16 (1969) 431-433.

[55] H.Curry: The method of steepest descent for nonlinear minimization problems. Quart. Appl. Math. 2 (1944) 258-261.

[56] Y.Dai: Analysis of conjugate gradient methods. Ph.D. thesis, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, 1997.

[57] Y.Dai: Further insight into the convergence of the Fletcher-Reeves method. Sci. China Ser. A 42 (1999) 905-916.

[58] Y.Dai: Convergence of nonlinear conjugate gradient methods. Journal of Computational Mathematics 19 (5), 539-548.

[59] Y.Dai: Convergence of Polak-Ribiere-Polyak conjugate gradient method with constant stepsizes. Research report AMSS-2001-040, Academy of Mathematics and Systems Sciences, Chinese Academy of Sciences 2001.

[60] Y.Dai: New properties of a nonlinear conjugate gradient method. Numerische Mathematik 89 (2001) 83-98.

[61] Y.Dai: A nonmonotone conjugate gradient algorithm for unconstrained optimization. J. Syst. Sci. Complex. 15 (2002) 139-145.

[62] Y.Dai: Nonlinear conjugate gradient methods. Preprint (2010) 1-36.

[63] Y.Dai: J.Han, G.Liu, D.Sun, H.Yin, Y.Yuan: Convergence properties of nonlinear conjugate gradient methods. SIAM J. Optimization 10 (1999) 345-358.

[64] Y.H.Dai, L.Z.Liao: New conjugacy conditions and related nonlinear conjugate gradient methods. Appliad Mathematics and Optimization 43 (2001) 87-101.

[65] Y.Dai, L.Liao, D.Li: On restart procedures for the conjugate gradient method. Numerical Algorithms 35 (2004) 249-260.

[66] Y.Dai, J.M.Martinez, Y.Yuan : An increasing-angle property of the conjugate gradient method and the implementation of large-scale minimization algorithms with line searches. Numerical Linear Algebra with Applications 10 (2003) 323-334.

[67] Y.Dai, Q.Ni: Testing different conjugate gradient methods for large-scale unconstrained optimization. J. of Computational Mathematics 21 (2003) 311-320.

[68] Y.Dai, Y.Yuan, Convergence properties of the Fletcher-Reeves method. IMA J. of Numerical Analysis 16 (1996) 155-164.

[69] Y.Dai, Y.Yuan: Convergence properties of the conjugate descent method. Adv. Math. (China) 26 (1996) 552-562.

[70] Y.Dai, Y.Yuan: Further studies on the Polak-Ribiere-Polyak method. Research report ICM-95-040, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, 1995.

[71] Y.H.Dai, Y.Yuan: A nonlinear conjugate gradient method with a strong global convergence property. SIAM J. Optimization 10 (1999) 177-182.

[72] Y.Dai, Y.Yuan: Global convergence of the method of shortest residuals. Numerische Mathematik 83 (1999) 581-598.

[73] Y.Dai, Y.Yuan: Convergence of the Fletcher-Reeves method under a generalized Wolfe search. J. of Computational Mathematics 2 (1996) 142-148.

[74] Y.Dai, Y.Yuan: Convergence properties of the Beale-Powell restart algorithm. Sci. China Ser. A 41 (1998) 1142-1150.

[75] Y.Dai, Y.Yuan: Some properties of a new conjugate gradient method. In: Advances in Nonlinear Programming (Y. Yuan ed.), Kluwer Publications, Boston, (1998) 251-262.

[76] Y.Dai, Y.Yuan: A note on the nonlinear conjugate gradient method. J. of Computational Mathematics 20 (2002) 575-582.

[77] Y.Dai, Y.Yuan: A class of globally convergent conjugate gradient methods. Research report ICM-98-030, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, 1998.

[78] Y.Dai, Y.Yuan:, Extension of a class of nonlinear conjugate gradient methods. Research report ICM-98-049, Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, 1998.

[79] Y.Dai, Y.Yuan:, A three-parameter family of hybrid conjugate gradient method. Mathematics of Computation 70 (2001) 1155-1167.

[80] Y.H.Dai, Y.Yuan: An efficient hybrid conjugate gradient method for unconstrained optimization. Annals of Operations Research, 2001.

[81] Y.Dai, Y.Yuan:, A class of globally convergent conjugate gradient methods. Sci. China Ser. A, 46 (2003) 251-261.

[82] J.W.Daniel, W.B.Gragg, L.Kaufman, G.W.Stewart: Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. Mathematics of Computation 136 (1976) 772-795.

[83] W.C.Davidon: Variable metric method for minimisation. A.E.C. Research and Development Report ANL-5990, 1959.

[84] W.C.Davidon: Optimally conditioned optimization algorithms without line searches. Mathematical Programming 9 (1975) 1-30.

[85] T.A.Davis, I.S.Duff: An unsymmetric pattern multifrontal method for sparse LU factorization. Report No. TR-93-018, CIS Department, University of Florida, Gainesville 1993.

[86] M.Dehghan, M.Hajarian: Convergence of the modified Dai-Yuan conjugate gradient method for unconstrained optimization. Preprint (2010) 1-8.

[87] N.Deng, Z.Li: Global convergence of three terms conjugate gradient methods. Optimization Methods and Software 4 (1995) 273-282.

[88] R.S.Dembo, T.Steihaug: Truncated-Newton algorithms for large-scale unconstrained minimization. Math. Programming 26 (1983) 190-212.

[89] N.Y.Deng, Y.Xiao, F.J.Zhou: Nonmonotonic trust region algorithm. J. Optimizaton Theory and Applications 76 (1993) 259-285.

[90] 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.

[91] J.E.Dennis, D.M.Gay, R.E.Welsch: NL2SOL. An adaptive nonlinear least-squares algorithm. Transactions on Mathematical Software 7 (1981) 369-383.

[92] 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.

[93] J.E.Dennis, R.B.Schnabel: Numerical methods for unconstrained optimization and nonlinear equations. Prentice-Hall, Englewood Cliffs, New Jersey 1983.

[94] J.E.Dennis, L.N.Vicente: On the convergence theory of trust-region-based algorithms for equality-constrained optimization. SIAM J. Optimization 7 (1997) pp. 927-950.

[95] J.E.Dennis, R.E.Welsch: Techniques for Nonlinear Least Squares and Robust Regression. Communications in Statistics B 7 (1978) 345-359.

[96] L.C.W.Dixon, P.G.Ducksbury, P.Singh: A new three term conjugate gradient method. Technical Report No. 130, Numerical Optimization Centre, The Hatfield Polytechnic, 1985.

[97] E.D.Dolan, J.J.Moré: Benchmarking optimization software with performance profiles. Mathematical Programming 91 (2002) 201-213.

[98] S.Du, Y.Chen: Global convergence of a modified spectral FR conjugate gradient method. Applied Mathematics and Computation 202 (2008) 766-770.

[99] I.S.Duff, J.K.Reid: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. of Math. Software 9 (1983) 302-325.

[100] I.S.Duff, J.K.Reid, N.Munksgaard, H.B.Neilsen: Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. Journal of the Institute of Mathematics and its Applications 23, (1979) 235-250.

[101] A.El-Bakry, R.Tapia, T.Tsuchiya, Y.Zhang: On the formulation and theory of Newton interior-point method for nonlinear programming. Journal of Optimization Theory and Applications, 89, 507-541, 1996.

[102] R.Fletcher: A general quadratic programming algorithm. J. Inst. Math. Appl. 7 (1971) 76-91.

[103] R.Fletcher: A modified Marquardt subroutine for nonlinear least squares. Report No. R-6799, Theoretical Physics Division, A.E.R.E. Harwell, 1971.

[104] R.Fletcher: A FORTRAN subroutine for minimization by the method of conjugate gradients. Atomic Energy Research Establishment, Harwell, Oxfordhire, England, Report No. R-7073, 1972.

[105] R.Fletcher: A new approach to variable metric algorithms. Computer J. 13 (1970) 317-322.

[106] R.Fletcher: Nonlinear programmming without a penalty function. Numerical analysis report NA/171, University of Dundee, 1997.

[107] R.Fletcher: Practical methods of optimization (Second edition). Wiley, New York, 1987.

[108] R.Fletcher: Second order corrections for nondifferentiable optimization. In: "Numerical analysis, Dundee 1981" (G.A.Watson ed.), Lecture Notes in Mathematics912, Springer-Verlag, Berlin 1982.

[109] R. Fletcher and S. Leyffer (1997). Nonlinear programming without a penalty function. Technical Report NA/171, Department of Mathematics, University of Dundee.

[110] R.Fletcher, M.J.D.Powell: A rapidly convergent descent method for minimization. Computer J. 6 (1963) 163-168.

[111] R.Fletcher, C.M.Reeves: Function minimization by conjugate gradients. Computer J. 7 (1964) 149-154.

[112] R.Fletcher, C.Xu: Hybrid methods for nonlinear least squares. IMA J. Numer. Anal. 7 (1987) 371-389.

[113] J.A.Ford, Y.Narushima, H.Yabe: Multi-step nonlinear conjugate gradient methods for unconstrained minimization. conjugate gradients. Computational Optimization and Applications 40 (2007) 191-216.

[114] A.Forsgren, P.E.Gill, M.H.Wright: Interior methods for nonlinear optimization. SIAM Review 44 (2002) 525597.

[115] R.W.Freund, N.M.Nachtigal: A new Krylov-subspace method for symmetric indefinite linear systems. Report No. ORNL/TM-12754, Oak Ridge National Laboratory, Oak Ridge, Tennessee, 1994.

[116] F.Gao, L.Han: Implementing the Nelder-Mead simplex algorithm with adaptive parameters. Computational Optimization and Applications. To appear.

[117] R.P.Ge: A filled function method for finding a global minimizer of a function of several variables. Math. Programming 46 (1990) 191-204.

[118] R.P.Ge, Y.F.Qin: A Class of filled functions for finding global minimizers of a function of several variables, J. Optimization Theory and Applications 54 (1987) 241-252.

[119] J.C.Gilbert, C.Lemarechal: Some numerical experiments with variable-storage quasi-Newton algorithms. Math. Programming, 45 (1989) 407-435.

[120] J.C.Gilbert, J.Nocedal: Global convergence properties of conjugate gradient methods for optimization. SIAM J. Optimization 2 (1992) 21-42.

[121] P.E.Gill, M.W.Leonard: Limited-memory reduced-Hessian methods for large-scale unconstrained optimization. Technical Report NA 97-1, Department of Mathematics, University of California, San Diego, 1997.

[122] P.E.Gill, W.Murray: A numerically stable form of the simplex algorithm. Linear Algebra Appl. 7 (1973) 99-138.

[123] P.E.Gill, W.Murray: Newton type methods for unconstrained and linearly constrained optimization. Math. Programming 7 (1974) 311-350.

[124] P.E.Gill, W.Murray: Numerically stable methods for quadratic programming. Math. Programming 14 (1978) 349-372.

[125] P.E.Gill, W.Murray: Conjugate-gradient methods for large-scale nonlinear optimization. Systems Optimization Laboratory, Department of Operations Research, Stanford Univarsity, Report No. SOL-79-15, 1979.

[126] P.E.Gill, W.Murray, M.H.Wright: Practical optimization. Academic Press, London 1981.

[127] D.Goldfarb: A family of variable metric algorithms derived by variational means. Math Comput. 24 (1970) 23-26.

[128] D.Goldfarb, A.U.Idnani: A numerically stable dual method for solving strictly convex quadratic programms. Report No. 81-102, Dept.of Computer Sci., The City College of New York, 1981.

[129] A.A.Goldstein: On steepest descent. SIAM J. Control 3 (1965) 147-151.

[130] G.H.Golub, C.F.Van Loan: Matrix computations (second edition). Johns Hopkins University Press, Baltimore 1989.

[131] G.H.Golub, D.P.OLeary: Some history of the conjugate gradient methods and Lanczos algorithms: 1948 -1976. SIAM Review 31 (1989) 50-100.

[132] N.I.M.Gould, S.Lucidi, M.Roma, P.L.Toint: Solving the trust-region subproblem using the Lanczos method. Report No. RAL-TR-97-028, 1997.

[133] J.L.Greenstadt: Variational determination of some Broyden-like formulas. Working paper, IBM Scientific Center, Palo Alto, California 1975.

[134] A.Griewank, P.L.Toint: Partitioned variable metric updates for large scale structured optimization problems. Numer. Math. 39 (1982) 119-137.

[135] L.Grippo, F.Lampariello, S.Lucidi: A nonmonotone line search technique for Newton's method. SIAM J. Numer. Anal. 23 (1986) 707-716.

bibitemgrl1 L.Grippo, S.Lucidi: A globally convergent version of the Polak-Ribiere conjugate gradient method. Mathematical Programming 78 (1997) 375-391.

[136] L.Grippo, S.Lucidi: Convergence conditions, line search algorithms and trust region implementations for the Polak-Ribiere conjugate gradient method. Optimization Methods and Software 20 (2005) 71-98.

[137] E.Hairer, S.P.Norsett, G.Wanner: Solving ordinary differential equations I. Springer Series in Computational Mathematics 8, Springer Verlag, Berlin 1987.

[138] E.Hairer, S.P.Norsett, G.Wanner: Solving ordinary differential equations II. Springer Series in Computational Mathematics 8, Springer Verlag, Berlin 1987.

[139] W.W.Hager: Minimizing a quadratic over a sphere. SIAM J. Optimization 12 (2001) 188-208.

[140] W.W.Hager, H.Zhang: A survey of nonlinear conjugate gradient methods. Pacific Journal of Optimization 2 (2006) 35-58.

[141] W.W.Hager, H.Zhang: A New conjugate gradient method with guaranteed descent and an efficient line search. SIAM J. Optimization 16 (2005) 170-192.

[142] W.W.Hager, H.Zhang: Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranted descent. ACM Transactions on Mathematical Software 32 (2006) 113-137.

[143] J.Hald, K.Madsen: Linearly constrained minimax optimization. Math. Programming 20 (1981) 49-62.

[144] S.P.Han: Variable metric methods for minimizing a class of nondifferentiable functions. Math. Programming 20 (1981) 1-13.

[145] J.Han, G.Liu, D.Sun, H.Yin: Two fundamental convergence theorems for nonlinear conjugate gradient methods and their applications. Acta Math. Appl. Sinica, 17 (2001) 38-46.

[146] J.Han, G.Liu, H.Yin: Convergence properties of conjugate gradient methods with strong Wolfe linesearch. Systems Sci. Math. Sci. 11 (1998) 112-116.

[147] R.J.Hanson, P.Dyer: A computational algorithm for sequential estimation. Mathematics of Computation 14 (1971) 285-290.

Convergence properties of conjugate gradient methods with strong Wolfe linesearch. Systems Sci. Math. Sci. 11 (1998) 112-116.

[148] M.R.Hestenes, Conjugate direction methods in optimization. Springer-Verlag, New York, 1980.

[149] M.R.Hestenes, C.M.Stiefel: Methods of conjugate gradient for solving linear systems. J. Res. NBS 49 (1964) 409-436.

[150] H.Hirst: N-step quadratic convergence in the conjugate gradient method. PhD Dissertation, Department of Mathematics, Pennsylvania State University, State College, PA, 1989.

[151] W.Hock, K.Schittkowski: Test examples for nonlinear programming codes. Lecture notes in economics and mathematical systems 187. Springer Verlag, Berlin 1981.

[152] R.Hooke, T.A.Jeeves: Direct search solution of numerical and statistical problems. J. Assoc. Comp. Mach. 8 (1961) 212-221.

[153] S.Hoshino: A formulation of variable metric methods. J. Inst. Math. Appl. 10 (1972) 394-403.

[154] M.E.Hribar, J.Nocedal: Improvement to the Horizontal Subproblem. Preprint 1996.

[155] Y.F.Hu, C.Storey: Efficient generalized conjugate gradient algorithms. Part 2: Implementation. J. Optimizaton Theory and Applications 69 (1991) 139-152.

[156] Y.Hu, C.Storey: Global convergence result for conjugate gradient methods. J. Optimization Theory and Applications 71 (1991) 399-405.

[157] 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.

[158] H.Huang, Z.Wei, Y.Shengwei: The proof of the sufficient descent condition of the Wei-Yao-Liu conjugate gradient method under the strong Wolfe-Powell line search. Applied Mathematics and Computation 189 (2007) 1241-1245.

[159] C.M.Ip, M.J.Todd: Optimal conditioning and convergence in rank one quasi-Newton updates. SIAM J. Numer. Anal. 25 (1988) 206-221.

[160] P.Ječmen: JUFOEd - Editor pro systém UFO. Technická Universita Liberec 2010.

[161] P.Ječmen: JUFOPlot - Zobrazovač výstupu systému UFO. Technická Universita Liberec 2010.

[162] K.M.Khoda, Y.Liu, C.Storey: Generalized Polak-Ribiere algorithm. J. Optimization Theory and Applications 75 (1992) 345-354.

[163] K.C.Kiwiel: An ellipsoid trust region bundle method for nonsmooth convex minimization. SIAM J. on Control and Optimization 27 (1989) 737-757.

[164] R.Klessig, E.Polak: Efficient implementation of the Polak-Ribiere conjugate gradient algorithm. SIAM J. Control 10 (1972) 524-549.

[165] C.X.Kou, Y.H.Dai: A modified self-scaling memoryless Broyden-Fletcher-Goldfarb-Shanno method for unconstrained optimization. J. Optimization Theory and Applications, (2014) to appear.

[166] M.Lalee, J.Nocedal, T Plantenga: On the implementation of an algorithm for large-scale equality constrained optimization. SIAM J. Optimization, Vol.8, 1998, pp.682-706.

[167] C.L.Lawson, R.J.Hanson: Solving least squares problems. Prentice-Hall, Englewood Cliffs, New Jersey 1974.

[168] R.B.Lehoucq, D.C.Sorensen, and C.Yang: ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods. SIAM, Philadelphia, 1998.

[169] A.V.Levy, A.Montalvo: The tunneling algorithm for the global minimization of functions. SIAM Journal Sci. Stat. Comp. 6 (1985) 15-19.

[170] G.Li: Successive column correction algorithms for solving sparse nonlinear systems of equations. Mathematical Programming 43 (1989) 187-207.

[171] G.Li, C.Tang, Z.Wei: New conjugacy condition and related new conjugate gradient methods for unconstrained optimization. J. of Computational and Applied Mathematics 202 (2007) 523-539.

[172] P.Lindstrom, P.A.Wedin: A new linesearch algorithm for nonlinear least squares problems. Math. Programming 29 (1984) 268-296.

[173] G.Liu, J.Han, H.Yin: Global convergence of the Fletcher-Reeves algorithm with an inexact line search. Appl. Math. J. Chinese Univ. Ser. B 10 (1995) 75-82.

[174] G.Liu, L.Jing, L.Han, D.Han: A class of nonmonotone conjugate gradient methods for unconstrained optimization. J. Optimization Theory and Applications 101 (1999) 127-140.

[175] D.C.Liu, J.Nocedal: On the limited memory BFGS method for large-scale optimization. Math. Programming 45 (1989) 503-528.

[176] Y.Liu, C.Storey: Efficient generalized conjugate gradient algorithms. Part 1: Theory. J. Optimizaton Theory and Applications 69 (1991) 129-137.

[177] L.Lukšan: New Combined Method for Unconstrained Minimization. Computing 28 (1982) 155-169.

[178] L.Lukšan: Quasi-Newton Methods without Projections for Unconstrained Minimization. Kybernetika 18 (1982) 290-306.

[179] L.Lukšan: Quasi-Newton Methods without Projections for Linearly Constrained Minimization. Kybernetika 18 (1982) 307-319.

[180] L.Lukšan: Variable Metric Method with Limited Storage for Large Scale Unconstrained Minimization. Kybernetika 18 (1982) 517-528.

[181] L.Lukšan: Variable Metric Methods for Linearly Constrained Nonlinear Minimax Approximation. Computing 30 (1983) 315-334.

[182] 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.

[183] L.Lukšan: An implementation of recursive quadratic programming variable metric methods for linearly constrained nonlinear minimax approximation. Kybernetika 21 (1985) 22-40.

[184] L.Lukšan: A Compact Variable Metric Algorithm for Linearly Constrained Nonlinear Minimax Approximation. Kybernetika 21 (1985) 405-427.

[185] L.Lukšan: Variable Metric Methods for a Class of Extended Conic Functions. Kybernetika 21 (1985) 96-107.

[186] L.Lukšan: A Compact Variable Metric Algorithm for Linear Minimax Approximation. Computing 36 (1986) 355-373.

[187] L.Lukšan: Dual Method for Solving a Special Problem of Quadratic Programming as a Subproblem at Nonlinear Minimax Approximation. Applications of Mathematics 31 (1986) 379-395.

[188] L.Lukšan: Conjugate Direction Algorithms For Extended Conic Functions. Kybernetika 22 (1986) 31-46.

[189] L.Lukšan: Conjugate Gradient Algorithms for Conic Functions. Applications of Mathematics 31 (1986) 427-440.

[190] L.Lukšan: Computational experience with improved variable metric methods for unconstrained minimization. Kybernetika 26 (1990) 415-431.

[191] L.Lukšan: Variable metric methods. Unconstrained minimization. Academia, Prague 1990 (in Czech).

[192] L.Lukšan: A note on comparison of statistical software for nonlinear regression. Computational Statistics Quaterly 6 (1991) 321-324.

[193] L.Lukšan: Computational experience with improved conjugate gradient methods for unconstrained minimization. Kybernetika 28 (1992) 249-262.

[194] L.Lukšan: Variationally derived scalling and variable metric updates from the preconvex part of the Broyden family. J. Optimizaton Theory and Applications 73 (1992) 299-307.

[195] L.Lukšan: Inexact trust region method for large sparse nonlinear least squares. Kybernetika 29 (1993) 305-324.

[196] L.Lukšan: Inexact trust region method for large sparse systems of nonlinear equations. J. Optimizaton Theory and Applications 81 (1994) 569-590.

[197] L.Lukšan: Computational experience with known variable metric updates. J. Optimizaton Theory and Applications 83 (1994) 27-47.

[198] L.Lukšan: Combined trust region methods for nonlinear least squares. Kybernetika 32 (1996) 121-138.

[199] L.Lukšan: Efficient trust region method for nonlinear least squares. Kybernetika 32 (1996) 105-120.

[200] L.Lukšan: Hybrid methods for large sparse nonlinear least squares. J. Optimizaton Theory and Applications 89 (1996) 575-595.

[201] L.Lukšan: Numerické optimalizační metody. Technical Report V-1058. Prague, ICS AS CR, 2009.

[202] L.Lukšan, J.Hartman: Implementace automatického derivování v systému UFO. Technical Report V-1002. Prague, ICS AS CR 2007.

[203] J.Hartman, L.Lukšan, J.Zítko: Automatic differentiation and its program realization. Kybernetika 45 (2009) 865-883.

[204] L.Lukšan, C.Matonoha, J.Vlček: A shifted Steihaug-Toint method for computing trust-region step. Technical Report V-914. Prague, ICS AS CR, 2004.

[205] L.Lukšan, C.Matonoha, J.Vlček: Interior point method for nonlinear nonconvex optimization. Numerical Linear Algebra with Applications 11 (2004) 431-453.

[206] L.Lukšan, C.Matonoha, J.Vlček J.: Algorithm 896: LSA: Algorithms for Large-Scale Optimization. ACM Transactions on Mathematical Software 36 (2009) No. 3.

[207] L.Lukšan, C.Matonoha, J.Vlček: Nonsmooth equation method for nonlinear nonconvex optimization. In: Conjugate Gradient Algorithms and Finite Element Methods (M.Křížek, P.Neittaanmäki, R.Glowinski, S.Korotov eds.). Springer Verlag, Berlin 2004.

[208] L.Lukšan, C.Matonoha, J.Vlček: Interior point methods for large-scale nonlinear programming. Optimization Methods and software 20 (2005) 569-582.

[209] L.Lukšan, C.Matonoha, J.Vlček: Primal interior-point method for large sparse minimax optimization. Kybernetika 45 (2009) 841-864.

[210] L.Lukšan, C.Matonoha, J.Vlček: Interior point methods for minimization of composite nonsmooth functions. Technical Report V-987. Prague, ICS AS CR, 2006.

[211] L.Lukšan, C.Matonoha, J.Vlček: Trust-region interior point method for large sparse l_1 optimization. Optimization Methods and Software 22 (2007) 737-753.

[212] L.Lukšan, C.Matonoha, J.Vlček: Primal interior point method for minimization of generalized minimax functions. Kybernetika (2010) 697-721.

[213] L.Lukšan, C.Matonoha, J.Vlček: On Lagrange multipliers of trust-region subproblems. BIT Numerical Analysis 48 (2008) 763-768.

[214] L.Lukšan, C.Matonoha, J.Vlček: Computational experience with modified conjugate gradient methods for unconstrained optimization. Technical Report V-1038. Prague, ICS AS CR 2008.

[215] L.Lukšan, C.Matonoha, J.Vlček: Interior point method for nonlinear programming with complementarity constraints. Technical Report V-1039. Prague, ICS AS CR 2008.

[216] L.Lukšan, C.Matonoha, J.Vlček: Sparse test problems for unconstrained optimization. Technical Report V-1064. Prague, ICS AS CR 2010.

[217] L.Lukšan, C.Matonoha, J.Vlček: Band preconditioners for the matrix free truncated Newton method. Technical Report V-1079. Prague, ICS AS CR 2010.

[218] L.Lukšan, C.Matonoha, J.Vlček: Modified CUTE Problems for Sparse Unconstrained Optimization. Technical Report V-1081. Prague, ICS AS CR 2010.

[219] L.Lukšan, E.Spedicato: Variable metric methods for unconstrained optimization and nonlinear least squares. Journal of Computational and Applied Mathematics 124 (2000) 61-93.

[220] L.Lukšan, J.Stuchlý, J.Vlček: Trust-region interior point method for large sparse inequality constrained optimization. Technical Report V-956. Prague, ICS AS CR, 2005.

[221] L.Lukšan, J.Vlček: Simple scaling for variable metric updates. Technical Report V-611. Prague, ICS AS CR, 1995.

[222] L.Lukšan, J.Vlček: Optimization of dynamical systems. Kybernetika 32 (1996) 465-482.

[223] L.Lukšan, J.Vlček: Efficient algorithm for large sparse equality constrained nonlinear programming problems. Technical Report V-652. Prague, ICS AS CR 1996.

[224] L.Lukšan, J.Vlček: Truncated trust region methods based on preconditioned iterative subalgorithms for large sparse systems of nonlinear equations. J. Optimization Theory and Applications 95 (1997) 637-658.

[225] L.Lukšan, J.Vlček: Subroutines for testing large sparse and partially separable unconstrained and equality constrained optimization problems. Technical Report V-767. Prague, ICS AS CR 1998.

[226] L.Lukšan, J.Vlček: A bundle-Newton method for nonsmooth unconstrained minimization. Mathematical Programming 83 (1998) 373-391.

[227] L.Lukšan, J.Vlček: Computational experience with globally convergent descent methods for large sparse systems of nonlinear equations. Optimization Methods and Software 8 (1998). 201-223.

[228] L.Lukšan, J.Vlček: Indefinitely preconditioned inexact Newton method for large sparse equality constrained nonlinear programming problems. Numerical Linear Algebra with Applications 5 (1998) 219-247.

[229] L.Lukšan, J.Vlček: Globally convergent variable metric method for convex nonsmooth unconstrained minimization. J. Optimization Theory and Applications 102 (1999) 593-613.

[230] L.Lukšan, J.Vlček: Preconditioning of saddle-point systems. In: Proceedings of the conference SIMONA 2000, Liberec 2000.

[231] L.Lukšan, J.Vlček: Algorithm 811: NDA: Algorithms for nondifferentiable optimization. Transactions on Mathematical Software 27 (2001) 193-213.

[232] L.Lukšan, J.Vlček: Numerical experience with iterative methods for equality constrained nonlinear programming problems. Optimization Methods and Software 16 (2001) 257-287.

[233] L.Lukšan, J.Vlček: Test problems for nonsmooth unconstrained and linearly constrained optimization. Technical Report V-798. Prague, ICS AS CR 2000.

[234] L.Lukšan, J.Vlček: Test problems for unconstrained optimization. Technical Report V-897. Prague, ICS AS CR, 2003.

[235] L.Lukšan, J.Vlček: Discrete minimax approximation with application to filter design. Proc. of the 2-nd International Workshop on Simulation, Modelling and Numerical Analysis, SIMONA 2003. Liberec, 2003.

[236] L.Lukšan, J.Vlček: Variable metric methods for nonsmooth optimization. In: Numerical Linear Algebra and Optimization (Ya-xiang Yuan ed.), Science Press, Beijing, 2003.

[237] L.Lukšan, J.Vlček: Efficient methods for large-scale unconstrained optimization. Proceedings on the conference "Large-scale Nonlinear Optimization", Erice 2004.

[238] L.Lukšan, J.Vlček: Variable metric method for minimization of partially separable nonsmooth functions. Pacific Journal on Optimization 2 (2006).

[239] L.Lukšan, J.Vlček: Recursive form of general limited memory variable metric methods. Kybernetika 49 (2013) 224-235.

[240] L.Lukšan, J.Vlček: Efficient tridiagonal preconditioner for the matrix-free truncated Newton method. Applied Mathematics and Computation 235 (2014) 394-407.

[241] L.Lukšan, J.Vlček: New quasi-Newton method for solving systems of nonlinear equations. Applications of Mathematics 62 (2017) 121-134.

[242] L.Lukšan, J.Vlček: New partitioned quasi-Newton method for nonlinear least squares problems. Technical Report V-1246. Prague, ICS AS CR, 2017.

[243] K.Madsen: An algorithm for minimax solution of overdetermined systems of non-linear equations. J. Institute of Mathematics and its Applications, 16 (1975) 321-328.

[244] K.Madsen, H.Schjaer-Jacobsen: Linearly constrained minimax optimization. Mathematical Programming 14 (1987) 208-223.

[245] M.Mäkelä, J.Neittaanmäki: Nonsmooth Optimization. World Scientific Publishing Co. Ltd. London 1992.

[246] J.M.Martinez: A quasi-Newton method with modification of one column per iteration. Computing 33 (1984) 353-362.

[247] 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.

[248] E.S.Marwill: Exploiting sparsity in Newton-like methods. Ph.D. Thesis, Cornell University, Ithaca 1978.

[249] H.Matthies, G.Strang: The solution of nonlinear finite element equations. Int. J. for Numerical Methods in Engineering 14 (1979) 1613-1623.

[250] G.P.McCormick, K.Ritter: Alternative Proofs of the convergence properties of the conjugate-gradient method. J. Optimization Theory and Applications 13 (1975) 497-518.

[251] M.F.McGuire, P.Wolfe: Evaluating a restart procedure for conjugate gradients. Report RC-4382, IBM Research Center, Yorktown Heights, 1973.

[252] J.Miao: Two infeasible interior-point predictor-corrector algorithms for linear programming. SIAM J. Optimization 6 (1996) 587-599.

[253] A.Miele, J.W.Cantrell: Study on a memory gradient method for the minimization of functions. J. Optimization Theory and Applications 3 (1969) 459-185.

[254] 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.

[255] S.Mizuno: Polynomiality of infeasible-interior-point algorithms for linear programming. Math Programming 67 (1994) 109-119.

[256] J.J.Moré: The Levenberg-Maquardt algorithm. Implementation and theory. In: "Numerical Analysis" (G.A.Watson ed.) Springer Verlag, Berlin 1978.

[257] J.J.Moré, B.S.Garbow, K.E.Hillstrm: Testing unconstrained optimization software. ACM Trans. Math. Software 7 (1981) 17-41.

[258] J.J.Moré, D.C.Sorensen: Computing a trust region step. Report No. ANL-81-83, Argonne National Laboratory. 1981.

[259] H.Mukai: Readily implementable conjugate gradient methods. Mathematical Programming 17 (1979) 298-319.

[260] Y.Narushima, H.Yabe: Global convergence of a memory gradient method for unconstrained optimization. Computational Optimization and Applications 35 (2006) 325-346.

[261] S.G.Nash: Newton-type minimization via Lanczos method. SIAM Journal on Numerical Analysis **21** (1984), 770-788.

[262] S.G.Nash: Preconditioning of truncated-Newton methods. SIAM Journal on Scientific and Statistical Computation **6** (1985), 599-616.

[263] S.G.Nash, A.Sofer: Preconditioning reduced matrices. SIAM J. on Matrix Analysis and Application 17 (1996) 47-68.

[264] J.L.Nazareth: A conjugate direction algorithm without line searches. J. Optimization Theory and Applications 23 (1977) 373-387.

[265] J.L.Nazareth: A relationship between the BFGS and conjugate gradient algorithms and its implications for the new algorithms. SIAM J. Numerical Analysis 16 (1979) 794-800.

[266] J.L.Nazareth: Conjugate gradient methods less dependent on conjugacy. SIAM Review 28 (1986) 501-511.

[267] J.L.Nazareth: A view of conjugate gradient-related algorithms for nonlinear optimization. In: Proceedings of the AMS-IMS-SIAM Summer Research Conference on Linear and Nonlinear Conjugate Gradient-Related Methods, University of Washington, Seattle, WA (July 9-13, 1995).

[268] J.L.Nazareth: Conjugate-gradient methods. IN: Encyclopedia of Optimization (C.Floudas, P.Pardalos, eds.) Kluwer Academic Publishers, Boston, 1999.

[269] J.L.Nazareth, J.Nocedal: Properties of conjugate gradient methods with inexact line searches. Systems Optimization Laboratory, Department of Operations Research, Stanford Univarsity, Report No. SOL-78-1, 1978.

[270] J.A.Nelder, R.Mead: A simplex method for function minimization. Computer J. 7 (1965) 308-313.

[271] A.Neumaier: On convergence and restart conditions for a nonlinear conjugate gradient method. Preprint 1997.

[272] J.Nocedal: Updating quasi-Newton Matrices with limited storage. Math. Comput. 35 (1980) 773-782.

[273] J.Nocedal: Theory of Algorithm for Unconstrained Optimization. Acta Numerica, Cambridge University Press (1991) 199-242.

[274] J.Nocedal: Conjugate Gradient Methods and Nonlinear Optimization. In: Proceedings of the AMS-IMS-SIAM Summer Research Conference on Linear and Nonlinear Conjugate gradient-Related Methods, University of Washington, Seattle, WA (July 9-13, 1995).

[275] J.Nocedal: Large scale unconstrained optimization. In: State of the Art in Numerical Analysis (A. Watson, I. Du., eds.) Oxford University Press, (1997) 311-338.

[276] J.Nocedal, Y.Yuan: Combining trust region and line search techniques. To appear.

[277] 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.

[278] S.S.Oren, E. Spedicato: Optimal conditioning of self scaling variable metric algorithms. Math Programming 10 (1976) 70-90.

[279] 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.

[280] J.M.Perry: A class of conjugate gradient algorithms with a two-step variable-metric memory. Discussion Paper 269, Center for Mathematical Studies in Economics and Management Sciences, Northwestern University, Evanston, Illinois, 1977.

[281] E.Polak, G.Ribiere: Note sur la convergence des methodes de directions conjuges. Revue Francaise Inform. Mech. Oper. 16-R1 (1969) 35-43.

[282] E.Polak, J.O.Royset, R.S.Womersley: Algorithm with adaptive smoothing for finite minimax problems. Journal of Optimization Theory and Applications 119 (2003) 459-484.

[283] B.T.Polyak: The conjugate gradient method in extreme problems. USSR Comp. Math. Math. Phys. 9 (1969) 94-112.

[284] 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.

[285] 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.

[286] M.J.D.Powell: Some convergence properties of the conjugate gradient method. Mathematical Programming 11 (1976) 42-49.

[287] M.J.D.Powell: Restart procedures of the conjugate gradient method. Math. Programming 12 (1977) 241-254.

[288] M.J.D.Powell: A fast algorithm for nonlinearly constrained optimization calculations. In: "Numerical analysis" (G.A.Watson ed.). Springer Verlag, Berlin 1977.

[289] M.J.D.Powell: Nonconvex minimization calculations and the conjugate gradient method. In: Numerical Analysis (Dundee, 1983), Lecture Notes in Mathematics, Vol. 1066, Springer-Verlag, Berlin, (1984) 122-141.

[290] M.J.D.Powell: Convergence properties of algorithms for nonlinear optimization. SIAM Review 28 (1986) 487-500.

[291] D.G.Pu: A Class of modified Broyden algorithms. J. Computational Mathematics 12 (1994) 366-379.

[292] R.Pytlak: On the convergence of conjugate gradient algorithm. IMA J. of Numerical Analysis 14 (1989) 443-460.

[293] R.Pytlak: Global convergence of the method of shortest residuals by Y.Dai and Y.Yuan. Numerische Mathematik 91 (2002) 319-321.

[294] R.Pytlak, T.Tarnawski: Preconditioned conjugate gradient algorithms for nonconvex problems. Pacific Journal of Optimization 2 (2006) 81-104.

[295] R.Pytlak, T.Tarnawski: On the method of shortest resduals for unconstrained optimization. J. Optimization Theory and Applications 133 (2007) 99-110.

[296] H.Ramsin, P.A.Wedin: A Comparison of Some Algorithms for the Nonlinear Least Squares Problem. BIT 17 (1977) 72-90.

[297] 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.

[298] 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.

[299] M.Rojas, S.A.Santos, D.C.Sorensen: A new matrix-free algorithm for the large-scale trust-region subproblem. SIAM J. Optimization 11 (2000) 611-646.

[300] Y.Saad, M.Schultz: GMRES a Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM Journal on Scientific and Statistical Computations 7 (1986) 856-869.

[301] S.Sanmayias, E.Vercher: A generalized conjugate gradient algorithm. J. Optimization Theory and Applications 98 (1998) 489-502.

[302] S.Schlenkrich, A.Griewank, A.Walther: On the local convergence of adjoint Broyden methods. Mathematical Programming 121 (2010),221-247.

[303] S.Schlenkrich, A.Walther: Global convergence of quasi-Newton methods based on adjoint Broyden updates. Applied Numeerical Mathematics 59 (2009),1120-1136.

[304] R.B.Schnabel, E.Eskow: A new Choleski factorization. SIAM J. Sci. Stat. Comput. 11 (19900, 1136-1158.

[305] 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.

[306] D.F.Shanno: Conditioning of quasi-Newton methods for function minimization. Math. Comput. 24 (1970) 647-656.

[307] D.F.Shanno: On the convergence of a new conjugate gradient algorithm. SIAM J. Numerical Analysis 15 (1978) 1247-1257.

[308] D.F.Shanno: Conjugate gradient methods with inexact searches. Math. Oper. Res. 3 (1978) 244-256.

[309] D.F.Shanno: Globally convergent conjugate gradient algorithms. Mathematical Programming 33 (1985) 61-67.

[310] D.F.Shanno, K.J.Phua: Matrix conditioning and nonlinear optimization. Math. Programming 14 (1978) 144-160.

[311] D.F.Shanno, J.Vanderbei: Interior-point methods for nonconvex nonlinear programming: Orderings and higher-order methods. Mathematical Programming, 87, 303-316, 2000.

[312] S.B.Sheng, Z.H.Zou: A new secant method for nonlinear least squares problems. Numerical Mathematics, A Journal of Chinese Universities, 2 (1993) 125137.)

[313] Y.Shengwei, Z.Wei, H.Huang: A note about WYL's conjugate gradient method and its applications. Applied Mathematics and Computation 191 (2007) 381-388.

[314] J.R.Shewchuk: An introduction to the conjugate gradient method without the agonizing pain. See http://www.cs.cmu.edu/~jrs/jrspapers.html, 1994.

[315] Z.Shi, J.Guo: A new family of conjugate gradient methods. J. of Computational and Applied Mathematics 224 (2009) 444-457.

[316] Z.Shi, J.Shen: Convergence of Liu-Storey conjugate gradient method. European J. of Operational Research 182 (2007) 552-560.

[317] E.Spedicato: A class of rank-one positive definite quasi-Newton updates for unconstrained minimization. Math. Operationsforsch. Statist. Ser. Optimization 14 (1963) 61-70.

[318] 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.

[319] E.Spedicato, M.T.Vespucci: Numerical experiments with variations of the Gauss-Newton algorithm for nonlinear least squares. J. Optimizaton Theory and Applications 57 (1988) 323-339.

[320] P.Stange, A. Griewank, M. Bollhoffer: On the efficient update of rectangular LU factorizations subject to low rank modifications. ETNA - Electronic Transactions on Numerical Analysis 26 (2007) 161-177.

[321] 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.

[322] T.Steihaug: Local and superlinear convergence for truncated iterated projections methods. Math. Programming 27 (1983) 176-190.

[323] T.Steihaug: The conjugate gradient method and trust regions in large-scale optimization. SIAM J. Numer. Anal. 20 (1983) 626-637.

[324] G.W.Stewart: A modification of Davidon's minimization method to accept difference approximations of derivatives. J. ACM 14 (1967) 72-83.

[325] J.Stoer: On the relation between quadratic termination and convergence properties of minimization algorithms. Numerische Mathematik 28 (1977) 343-366.

[326] J.Sun, J.Zhang: Global convergence of conjugate gradient methods without line search. Ann. Oper. Res., 163 (2001) 161-173.

[327] 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.

[328] M.Šiška: Macroprocessor UFO (version 1990). Report No. 484 (in Czech), Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.

[329] C.Tang, Z Wei, G.Li: A new version of the Liu-Storey conjugate gradient method. Applied Mathematics and Computation 189 (2007) 302-313.

[330] P.L.Toint: On sparse and symmetric matrix updating subject to a linear equation. Math of Comp. 31 (1977) 954-961.

[331] P.L.Toint: Towards an efficient sparsity exploiting Newton method for optimization. In I.S.Duff ed.: Sparse Matrices and Their Uses. Academic ress, London, pp. 57-88.

[332] P.L.Toint: On large scale nonlinear least squares calculations. SIAM J. Sci. Stat. Comput. 8 (1987) 416-435.

[333] C.H.Tong: A comparative study of preconditioned Lanczos methods for nonsymmetric linear systems. Report No. SAND91-8240B, Sandia National Laboratories, Livermore 1992.

[334] D.Touati-Ahmed, C.Storey: Efficient hybrid conjugate gradient techniques. J. Optimization Theory and Applications 64 (1990), pp. 379-397.

[335] M.Tůma: A quadratic programming algorithm for large and sparse problems. Kybernetika 27 (1991) 155-167.

[336] 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.

[337] M.Tůma: Sparse fractioned variable metric updates. Report No. 497, Institute of Computer and Information Sciences, Czechoslovak Academy of Sciences, Prague 1991.

[338] H.A.Van der Vorst: Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 15 (1992) 631-644.

[339] J.Vanderbei, D.F.Shanno: An interior point algorithm for nonconvex nonlinear programming. Computational Optimization and Applications, 13, 231-252, 1999.

[340] J.Vanderbei, D.F.Shanno: Interior-point methods for nonconvex nonlinear programming: Filter methods and merit functions. Report No. ORFE-00-06, Operations Research and Financial Engineering, Princeton University, December 2000.

[341] P.S.Vassilevski, D.Lazarov: Preconditioning mixed finite element saddle-point elliptic problems. Numerical Linear Algebra with Applications 3 (1996) 1-20.

[342] J.Vlček: Bundle algorithms for nonsmooth unconstrained optimization. Technical Report V-608. Prague, ICS AS CR, 1994.

[343] J.Vlček, L.Lukšan: Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. J. Optimization Theory and Applications 111 (2001) 407-430.

[344] J.Vlček, L.Lukšan: New variable metric methods for unconstrained minimization covering the large-scale case. Technical Report V-876. Prague, ICS AS CR, 2002.

[345] Vlček J., Lukšan L.: Additional properties of shifted variable metric methods. Technical Report V-899. Prague, ICS AS CR, 2004.

[346] J.Vlček, L.Lukšan: Shifted limited-memory variable metric methods for large-scale unconstrained minimization. J. of Computational and Applied Mathematics, 186 (2006) 365-390.

[347] J.Vlček, L.Lukšan: New class of limited-memory variationally-derived variable metric methods. Technical Report V-973. Prague, ICS AS CR, 2006.

[348] J.Vlček, L.Lukšan: Limited-memory projective variable metric methods for unconstrained minimization. Technical Report V-1036. Prague, ICS AS CR 2008.

[349] J.Vlček, L.Lukšan: Transformations enabling to construct limited-memory Broyden class methods. Technical Report V-1037. Prague, ICS AS CR 2008.

[350] J.Vlček, L.Lukšan: Generalizations of the limited-memory BFGS method based on quasi-product form of update. Computational and Applied Mathematics Journal of Computational and Applied Mathematics 241 (2013) 116-129.

[351] J.Vlček, L.Lukšan: A conjugate directions approach to improve the limited-memory BFGS method. Applied Mathematics and Computation 219 (2012) 800-809.

[352] J.Vlček, L.Lukšan: A modified limited-memory BNS method for unconstrained minimization based on the conjugate directions idea. Optimization Methods and Software 30 (2015) 616-633.

[353] J.Vlček, L.Lukšan: Properties of the block BFGS update and its aplication to the limited-memory block BNS method for unconstrained minimization. Technical Report V-1244. Prague, ICS AS CR 2017.

[354] J.Vlček, L.Lukšan: A combination of the repeated BNS limited-memory method for unconstrained minimization with methods based on conjugate directions. Technical Report V-1245. Prague, ICS AS CR 2017.

[355] C.Wang, S.Lian: Global convergence properties of two new dependent Fletcher-Reeves conjugate gradient methods. Applied Mathematics and Computation 181 (2006) 920-931.

[356] C.Wang, Y.Zhang: Global convergence properties of s-related conjugate gradient methods. Chinese Science Bulletin, 43 (1998) 1959-1965.

[357] Z.Wei, G.Li, L.Qi: New nonlinear conjugate gradient formulas for large-scale unconstrained optimization problems Applied Mathematics and Computation 179 (2006) 407-430.

[358] Z.Wei, S.Yao, L.Liu: The convergence properties of some new conjugate gradient methods. Applied Mathematics and Computation 183 (2006) 1341-1350.

[359] Z.Wei, L.Liu, S.Yao: The superlinear convergence of a new quasi-Newton-SQP method for constrained optimization. Applied Mathematics and Computation (2007), to appear.

[360] P.Wolfe: Convergence conditions for ascent methods. SIAM Review 11 (1969) 226-235.

[361] P.Wolfe: Convergence conditions for ascent methods II: Some corrections. SIAM Review, 13 (1971) 185-188.

[362] P.Wolfe: A method of conjugate subgradients for minimizing nondifferentiable functions. Mathematical Programming Study 3 (1975) 145-173.

[363] M.H.Wright: Interior methods for constrained optimization, Acta Numerica, 1991.

[364] C.X.Xu: Hybrid Method for Nonlinear Least-Square Problems without Calculating Derivatives J. Optimization Theory and Applications 65 (1990) 555-574.

[365] S.Xu: Smoothing methods for minimax problems. Computational Optimization and Applications 20 (2001) 267-279.

[366] H.Yabe, T.Takahashi: Factorized quasi-Newton methods for nonlinear least squares problems. Math. Programming 51 (1991) 75-100.

[367] H.Yabe, M.Takano: Global convergence properties of nonlinear conjugate gradient methods with modified secant conditions. Computational Optimization and Applications 28 (2004) 203-225.

[368] Y.Ye: Interior Point Algorithms - Theory and Analysis, Wiley-Interscience Publication, 1997.

[369] G.H.Yu, L.Guan and W.Chen: Spectral conjugate gradient methods with sucient descent property for large-scale unconstrained optimization. Optimization Methods and Software 23 (2008) 275-293.

[370] G.Yu, Y.Zhao, Z.Wei: A descent nonlinear conjugate gradient method for large-scale unconstrained optimization. Applied Mathematics and Computation 187 (2007) 636-643.

[371] Y.Yuan: Analysis on the conjugate gradient method. Optimization Methods and Software 2 (1993) 19-29.

[372] Y.Yuan: On the truncated conjugate gradient method. Mathematical Programming A-87 (2000) 561-573.

[373] Y.Yuan, J.Stoer: A subspace study on conjugate algorithms. Z. Angew. Math. Mech. 75 (1995) 69-77.

[374] J.Z.Zhang, N.Y.Deng, L.H.Chen: New quasi-Newton equation and related methods for unconstrained optimization. J. Optimization Theory and Applications, Vol.102, 1999, pp.147-167.

[375] J.Z.Zhang, C.X.Xu: Properties and numerical performance of quasi-Newton methods with modified quasi-Newton equations. J. Computational and Applied Mathematics 137 (2001) 269-278.

[376] L.Zhang: A new Liu-Storey type nonlinear conjugate gradient method for unconstrained optimization problems. J. of Computational and Applied Mathematics (2008). To appear.

[377] L.Zhang: Two modified Dai-Yuan nonlinear conjugate gradient methods. Numerical Algorithms (2009). To appear.

[378] L.Zhang, W.Zhou: Two descent hybrid conjugate gradient methods for optimization. J. of Computational and Applied Mathematics 216 (2008) 251-264.

[379] L.Zhang, W.Zhou, D.Li: Global convergence of a modified Fletcher-Reeves conjugate gradient method with Armijo-type line search. Numerische Mathematik 104 (2006) 561-572.

[380] L.Zhang, W.Zhou, D.Li: A descent modified Polak-Ribiere-Polyak conjugate gradient method and its global convergence. IMA J. of Numerical Analysis 26 (2006) 629-640.

[381] 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.

[382] 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.

[383] Y.Zhang, K.Wang: A new form of conjugate gradient methods with guaranteed descent and global convergence properties. Numerical Algorithms. To appear.

[384] A.Žilinskas, A.A.Thorn: Global optimization. Springer Verlag, Berlin 1990.

A Demonstration of the text dialogue mode

Suppose that the model function has the form

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

(the Rosenbrock function) and the starting point is $x_1 = -1.2$ and $x_2 = 1.0$. If we type the statement `UFOGO` (without batch input file specification), then the following questions (which we supplement together with answers) appear on the screen.

UFO PREPROCESSOR V.3.1.

? INPUT ( ) ?

USER SUPPLIED INPUT:

HERE THE STARTING POINT, BOUNDS FOR VARIABLES,
TYPES OF CONSTRAINTS, THE STRUCTURE OF SPARSE PROBLEM,
AND OTHER INPUT DATA HAVE TO BE SPECIFIED.

X(1) = -1.2D0; X(2) = 1.0D0

? GRAPH (N) ?

SPECIFICATION OF GRAPHICAL OUTPUT
N - GRAPHICAL OUTPUT SUPPRESSED
Y - GRAPHICAL OUTPUT REQUIRED

? DISPLAY (N) ?

SPECIFICATION OF EXTENDED SCREEN OUTPUT
N - EXTENDED SCREEN OUTPUT SUPPRESSED
Y - EXTENDED SCREEN OUTPUT REQUIRED

? MODEL (FF) ?

TYPE OF OBJECTIVE FUNCTION
FF - GENERAL FUNCTION
FL - LINEAR FUNCTION
FQ - QUADRATIC FUNCTION
AF - SUM OF FUNCTIONS
AQ - SUM OF SQUARES
AP - SUM OF POWERS
AM - MINIMAX
DF - DIFFERENTIAL SYSTEM WITH GENERAL INTEGRAL CRITERION
DQ - DIFFERENTIAL SYSTEM WITH INTEGRAL OF SQUARES
DE - DIFFERENTIAL EQUATIONS
NE - NONLINEAR EQUATIONS
NO - MODEL IS NOT SPECIFIED

? NF (0) ?

NUMBER OF VARIABLES

2

? IEXT (0) ?

TYPE OF EXTREMUM
0 - MINIMUM
1 - MAXIMUM

? FMODELF (*) ?

MODEL OF OBJECTIVE FUNCTION

FF = <FORTRAN_EXPRESSION>

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

? GMODELF (*) ?

MODEL OF GRADIENT OF OBJECTIVE FUNCTION

GF(1) = <FORTRAN_EXPRESSION>
GF(2) = <FORTRAN_EXPRESSION>
.
.
GF(NF) = <FORTRAN_EXPRESSION>

? HMODELF (*) ?

MODEL OF HESSIAN MATRIX

HF(1) = <FORTRAN_EXPRESSION>
HF(2) = <FORTRAN_EXPRESSION>
.
.
HF(M) = <FORTRAN_EXPRESSION>

? KCF (2) ?

COMPLEXITY OF THE OBJECTIVE FUNCTION
1 - EASY COMPUTED FUNCTION
2 - REASONABLE BUT NOT EASY COMPUTED FUNCTION
3 - EXTREMELY COMPLICATED FUNCTION

? KSF (1) ?

SMOOTHNESS OF THE OBJECTIVE FUNCTION:
1 - SMOOTH AND WELL-CONDITIONED FUNCTION
2 - SMOOTH BUT ILL-CONDITIONED FUNCTION
3 - NONSMOOTH FUNCTION

? HESF (D) ?

TYPE OF HESSIAN MATRIX:
- D - DENSE
- S - SPARSE WITH KNOWN (GENERAL) STRUCTURE
- N - HESSIAN MATRIX IS NOT USED

? KBF (0) ?

TYPE OF SIMPLE BOUNDS:
- 0 - NO SIMPLE BOUNDS
- 1 - ONE SIDED SIMPLE BOUNDS
- 2 - TWO SIDED SIMPLE BOUNDS

? KBC (0) ?

TYPE OF GENERAL CONSTRAINTS:
- 0 - NO GENERAL CONSTRAINTS
- 1 - ONE SIDED GENERAL CONSTRAINTS
- 2 - TWO SIDED GENERAL CONSTRAINTS

? EXTREM (L) ?

TYPE OF OPTIMIZATION
- L - LOCAL OPTIMIZATION
- G - GLOBAL OPTIMIZATION

? NORMF (0) ?

SCALING SPECIFICATION FOR VARIABLES:
- 0 - NO SCALING IS PERFORMED
- 1 - SCALING FACTORS ARE DETERMINED AUTOMATICALLY
- 2 - SCALING FACTORS ARE SUPPLIED BY USER

? INPUTDATA (N) ?

READ INPUT VALUES OF X (Y OR N)

? TEST (N) ?

STANDARD TEST OF EXTERNAL SUBROUTINES:
- N - NO TEST
- Y - PERFORM TEST BEFORE SOLUTION
- A - PERFORM TEST AFTER SOLUTION
- O - PERFORM TEST WITHOUT SOLUTION

? KOUT (0) ?

LEVEL OF TEXT FILE OUTPUT:
ABS(KOUT)=0 - NO PRINT OR PAPER SAVING PRINT
ABS(KOUT)=1 - STANDARD PRINT OF ITERATIONS
ABS(KOUT)=2 - ADDITIONAL PRINT OF STEPSIZE SELECTION
ABS(KOUT)=3 - ADDITIONAL PRINT OF DIRECTION DETERMINATION AND VARIABLE METRIC UPDATE
ABS(KOUT)=4 - 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 (-2) ?

LEVEL OF BASIC SCREEN OUTPUT:
ABS(MOUT)=0 - NO OUTPUT
ABS(MOUT)=1 - FINAL OUTPUT
ABS(MOUT)=2 - ADDITIONAL OUTPUT IN EACH ITERATION
ABS(MOUT)=3 - ADDITIONAL FINAL OUTPUT OF LINEAR OR QUADRATIC PROGRAMMING
ABS(MOUT)=4 - ADDITIONAL OUTPUT IN EACH ITERATION OF LINEAR OR QUADRATIC PROGRAMMING
MOUT<0 - FINAL OUTPUT WITH TERMINATION CRITERION

1

? NOUT (0) ?

LEVEL OF BASIC SCREEN OUTPUT:
0 - BASIC FINAL OUTPUT
1 - EXTENDED FINAL OUTPUT

1

? MSELECT (1) ?

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

? LAPACK (N) ?

USE LAPACK SUBROUTINES
N - ONLY UFO SUBROUTINES
Y - CONNECTION TO LAPACK POSSIBLE

? OUTPUT ( ) ?

USER SUPPLIED OUTPUT:

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

? OUTPUTDATA (N) ?

WRITE OUTPUT VALUES OF X (Y OR N)

UFO PREPROCESSOR STOP

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

The result of the UFO preprocessor action is the following source 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 IMD
      INTEGER IX(1)
      REAL*8 UXVDOT
      REAL*8 GF(2)
      REAL*8 X(2)
      REAL*8 HD(2)
      REAL*8 HF(2*(2+1)/2)
      REAL*8 S(2)
      REAL*8 ALF
      REAL*8 BET
      REAL*8 XO(2)
      REAL*8 GO(2)
      INTEGER IMB
*
*     commons placed here were omitted
*     since they require a large space
*
```

```
*     ------------------
*     END OF DECLARATIONS
*     ------------------
*
      OPEN (2,FILE='P.OUT',STATUS='UNKNOWN')
      OPEN (3,FILE='P.DIM',STATUS='UNKNOWN')
      CALL UYCLEA
      CALL UYINTP
*
*     ----------
*     METHOD (1)
*     ----------
*
      CALL UYINT1
      CALL UOTES1('VM','L','I','1','B','FF
     &    ','D',NF)
      X(1)=-1.2D0
      X(2)=1.0D0
      CALL UYCLST
      WRITE(3,'(/''PROBLEM: NEXT ='',I8)') NEXT
      IF (NF.GT.2) THEN
      CALL UOERR2('UZLMIN',80,NF,2)
      CALL UOERR4
      ITERM=-80
      TXFU='LACK SPC'
      ENDIF
      WRITE(3,'(''NUMBER OF VARIABLES:        NF  ='',I8)') NF
      M=NF*(NF+1)/2
      IF (ITERM.LT.0) STOP
      CALL UYTIM1(ITIME)
      NDECF=0
      IF (ITERM.NE.0) GO TO 11200
      CALL UO0FU1(NF,NA,NAL,MAL,NC,NCL,MCL,EPS0,EPS1,EPS2,EPS3,EPS4,EPS5
     &    ,EPS6,EPS7,EPS8,EPS9,ETA0,ETA1,ETA2,ETA3,ETA4,ETA5,ETA6,ETA7,E
     &    TA8,ETA9,ALF1,ALF2,ALF3,BET1,BET2,BET3,GAM1,GAM2,GAM3,DEL1,DEL
     &    2,DEL3,RPF1,RPF2,RPF3,RGF1,RGF2,RGF3,FMIN,XMAX,XDEL,REXP,MET,M
     &    ET1,MET2,MET3,MES,MES1,MES2,MES3,MOT,MOT1,MOT2,MOT3,MOS,MOS1,M
     &    OS2,MOS3,MEP,MEP1,MEP2,MEP3,MEG,MEG1,MEG2,MEG3,MEX,MEX1,MEX2,M
     &    EX3,MED,MED1,MED2,MED3,MCG,MCG1,MFP,MFP1,MPF,MPF1,MGF,MGF1,MLP
     &    ,MLP1,MQP,MQP1,MEQ,MEQ1,MSG,MSG1,KSF,KCF,KSA,KCA,KSC,KCC,KTERS
     &    ,INITD,INITS,INITH,IREM,IADD,IRES1,IRES2,MRED,IRAN1,IRAN2,ISAM
     &    1,ISAM2,KINP,IPRN)
*
*     ----------------------
*     VARIABLE METRIC METHOD
*     TEMPLATE : U1FDU1
*     ----------------------
*
      ASSIGN 11130 TO IMD
      CALL UYPRO1('UXFU',1)
      CALL UYPRO2(FMIN,FO)
11110 CONTINUE
```

```
*
*     -----------------
*     MODEL DESCRIPTION
*     -----------------
*
11600 CALL UF1F01(NF,GF,GF,FF,F)
      GOTO (11640,11610,11620) ISB+1
11610 CONTINUE
      ASSIGN 11710 TO IMB
11700 CONTINUE
      NFV=NFV+1
      FF=1.0D2*(X(1)**2-X(2))**2+(X(1)-1.0D0)**2
      GOTO IMB
11710 CONTINUE
      GOTO 11600
11620 CONTINUE
      CALL UF0GS2(NF,X,IX,X,GF,FF,HD,R,SNORM,1.0D-15,1.0D-15,2,1)
      GOTO (11600,11630) ISB+1
11630 CONTINUE
      ASSIGN 11910 TO IMB
      GOTO 11700
11910 CONTINUE
      GO TO 11620
11640 CONTINUE
*
*     ------------------------
*     END OF MODEL DESCRIPTION
*     ------------------------
*
      GO TO IMD
11130 CONTINUE
      CALL UYTRUG(NF,N,X,GF,GF,UMAX,GMAX)
      CALL UO2FU3(NF,M,NA,NC,X,GF,HF,X,X,F,DMAX,GMAX)
      CALL UYFUT1(N,F,FO,UMAX,GMAX,DMAX,ITES,IRES1,IRES2,INEW)
      IF(ITERM.NE.0) GOTO 11190
11140 CONTINUE
      ASSIGN 11140 TO IMD
      CALL UUDSD1(N,HF,1)
      GOTO (11150,11110) ISB+1
11150 CONTINUE
      IF(ITERM.NE.0) GOTO 11190
      CALL UYCPSD(NF,IX,HF,HD,MCG1)
      CALL UYTRUH(NF,N,X,HF)
*
*     ------------------------
*     DIRECTION DETERMINATION
*     TEMPLATE : UDGLG1
*     ------------------------
*
      CALL UOD1D1
      IF (IDECF.LT.0) THEN
      IDECF=9
```

```
      INF=0
      ENDIF
      TDXX(1:4)='INV '
      IF (IDECF.EQ.0) THEN
*
*     INVERSION
*
      ALF=ETA2
      CALL UXDPGF(N,HF,INF,ALF,BET)
      CALL UXDPGI(N,HF)
      NDECF=NDECF+1
      IDECF=9
      ELSE IF (IDECF.EQ.9) THEN
      ELSE
      ITERD=-1
      TDXX='BAD DEC9'
      CALL UOERR1('UDDLI1',1)
      GO TO 12630
      ENDIF
      GNORM=SQRT(UXVDOT(N,GF,GF))
*
*     NEWTON LIKE STEP
*
      CALL UXDSMM(N,HF,GF,S)
      CALL UXVNEG(N,S,S)
      INITD=MAX(ABS(INITD),1)
      ITERD=1
      IF(INF.EQ.0) THEN
      TDXX(5:8)=' POS'
      ELSEIF(INF.LT.0) THEN
      TDXX(5:8)=' ZER'
      ELSE
      TDXX(5:8)=' NEG'
      ENDIF
      SNORM=SQRT(UXVDOT(N,S,S))
      NRED=INF
      CALL UOD1D5(ALF,BET,INF)
12630 CALL UOD1D2(N,GF,S)
*
*     ----------------------------
*     END OF DIRECTION DETERMINATION
*     ----------------------------
*
      IF (KD.GT.0) P=UXVDOT(N,GF,S)
      CALL UD1TL1(NF,N,GF,S,EPS0,ALF1,ALF2,R,P,GNORM,SNORM,RMIN,RMAX,XMA
     &    X,XDEL,MES,INITD,INITH)
      IF(ITERM.NE.0) GOTO 11190
      IF(IREST.NE.0) GOTO 11140
      CALL UYTRUS(NF,X,X,XO,GF,GO,S,S,RO,FP,FO,F,PO,P,CMAX,CMAXO)
11170 CONTINUE
      ASSIGN 11170 TO IMD
      CALL USOL01(EPS1,RO,RP,R,FO,FP,F,PO,PP,FMIN,FMAX,PAR1,PAR2,RMAX,RM
```

```
     &    IN,SNORM,MODE,KTERS,MES,MES1,MES2,INITS,MRED)
      GOTO (11174,11172) ISB+1
11172 CONTINUE
      CALL UXVDIR(NF,R,S,XO,X)
      GOTO 11110
11174 CONTINUE
      IF (ITERS.LE.0) THEN
      CALL UYZER0(NF,X,XO,R,F,FO,FF,P,PO,MOT3)
      IF(IDIR.EQ.0) THEN
      CALL UYRES1(TSXX)
      CALL UYSET1
      GO TO 11140
      ELSE IF (MOT3.EQ.0) THEN
      CALL UYSET1
      GO TO 11140
      ELSE
      ITERD=0
      ENDIF
      ENDIF
      IF(KD.GT.LD) THEN
      ASSIGN 11180 TO IMD
      GO TO 11110
      ENDIF
11180 CONTINUE
      TXFU=TUXX
      CALL UYUPSD(NF,X,IX,XO,GF,GO,HD,P,MCG1)
      CALL UYTRUD(NF,X,X,XO,GF,GO,R,F,FO,P,PO,DMAX)
      CALL UUDBI1(N,HF,S,XO,GO,R,PO,F,FO,P,1.0D 60,8)
      IF(IDIR.EQ.0) THEN
      IF(ITERH.NE.0) CALL UYRES1('UPDATE  ')
      GOTO 11130
      ELSE
      GOTO 11140
      ENDIF
11190 CONTINUE
      IF(ITERM.LT.0) TXFU=TDXX
      CALL UYEPI1(1)
11200 CONTINUE
      CALL UOERR3(KOUT,LOUT,MOUT,ITERM,IER)
      CALL UO1FU2(NF,NA,NC,X,X,X,X,FF,F,FO,DMAX,GMAX,XMAX,EPS0,EPS1,EPS2
     &    ,EPS3,EPS4,EPS5,BET1,BET2,GAM1,GAM2,ETA1,ETA2,MET,MET1,MET2,ME
     &    T3,MOT,MOT1,MOT2,MOT3,MES,MES1,MES2,MES3,MOS,MOS1,MOS2,MOS3,IN
     &    ITD,INITS,INITH,IRES1,KTERS,IPRN)
13599 CONTINUE
*
*     -----------------
*     END OF METHOD (1)
*     -----------------
*
      CALL UYTIM2(ITIME)
      CLOSE (2)
      CLOSE (3)
```

```
      END
*
*     ------------------------
*     INITIATION OF METHOD (1)
*     ------------------------
*
      SUBROUTINE UYINT1
*
*     commons placed here were omitted
*     since they require a large space
*
      REAL*8 XDELS,RPF1S,RPF2S,RPF3S,RGF1S,RGF2S,RGF3S
      COMMON/UMCLST/ XDELS,RPF1S,RPF2S,RPF3S,RGF1S,RGF2S,RGF3S
      ETA0=1.0D-15
      ETA9=1.0D 60
      ITR=6
      IRD=5
      IWR=2
*
*     many other assignments follow which were
*     omitted since they require a large space
*
      END
*
*     ---------------------
*     INITIATION OF PROBLEM
*     ---------------------
*
      SUBROUTINE UYINTP
*
*     commons placed here were omitted
*     since they require a large space
*
      NF=2
      IEXT=0
      KCF=2
      KSF=1
      KBF=0
      KBC=0
      NORMF=0
      KDF=0
      KDA=-1
      KDC=-1
      KDE=-1
      KDY=-1
      END
*     ----------------------------------------
*     BROYDEN CLASS OF VARIABLE METRIC UPDATES
*     TEMPLATE : UUDBI1
*     ----------------------------------------
      SUBROUTINE UUDBI1(N,H,S,XO,GO,R,PO,F,FO,P,ETA9,MET)
*
```

```
*     commons placed here were omitted
*     since they require a large space
*
      REAL*8 H(N*(N+1)/2),S(N),XO(N),GO(N),R,PO,ETA9
      REAL*8 F,FO,P
      REAL*8 AA,CC
      COMMON /UMFUN1/ AA,CC
      REAL*8 UXVDOT,UNFUN1
      REAL*8 DIS,POM,POM3,POM4,A,B,C,GAM,RHO,PAR
      REAL*8 DEN
      INTEGER IUPDT
      LOGICAL L1,L3
      EXTERNAL UNFUN1
      IF (MET.LE.0) GO TO 22
      CALL UOU1D1(N,XO,GO)
      IF (IDECF.NE.9) THEN
      ITERH=-1
      TUXX='BAD DEC9'
      CALL UOERR1('UUDBI2',1)
      GO TO 22
      ENDIF
      L1=ABS(4).GE.3.OR.ABS(4).EQ.2.AND.NIT.EQ.KIT
      L3=.NOT.L1
*
*     DETERMINATION OF THE PARAMETERS A, B, C
*
      B=UXVDOT(N,XO,GO)
      IF (B.LE.ZERO) THEN
      ITERH=2
      TUXX='B - NEG.'
      GO TO 22
      ENDIF
      CALL UXDSMM(N,H,GO,S)
      A=UXVDOT(N,GO,S)
      IF (A.LE.ZERO) THEN
      ITERH=1
      TUXX='A - NEG.'
      GO TO 22
      ENDIF
      IF(MET.GE.4.OR.L1) THEN
      IF (ITERD.NE.1) THEN
      MET=1
      C=ZERO
      ELSE
      C=-R*PO
      IF (C.LE.ZERO) THEN
      ITERH=3
      TUXX='C - NEG.'
      GO TO 22
      ENDIF
      ENDIF
      ELSE
```

```
      C=ZERO
      ENDIF
*
*     DETERMINATION OF THE PARAMETER RHO (NONQUADRATIC PROPERTIES)
*
      IF (FO-F+P.EQ.0) THEN
      RHO=ONE
      ELSE
      RHO=HALF*B/(FO-F+P)
      ENDIF
      IF(RHO.LE.1.0D-2) RHO=ONE
      IF(RHO*1.0D-2.GE.ONE) RHO=ONE
      AA=A/B
      CC=C/B
      IUPDT=0
      IF (L1) THEN
*
*     DETERMINATION OF THE PARAMETER GAM (SELF SCALING)
*
      IF (C.LE.ZERO) THEN
      PAR=A/B
      POM3=0.8D 0
      POM4=8.0D 0
      ELSE
      PAR=SQRT(A/C)
      POM3=0.7D 0
      POM4=6.0D 0
      ENDIF
      GAM=RHO/PAR
      IF (NIT.NE.KIT) THEN
      L3=GAM.LT.POM3.OR.GAM.GT.POM4
      ENDIF
      ENDIF
      IF (L3) THEN
      GAM=ONE
      PAR=RHO/GAM
      ENDIF
*
*     NEW UPDATE
*
      POM=ONE/(AA*CC)
      IF (POM.LT.ONE) THEN
      DEN=MAX(POM+1.0D-15,SQRT(C/A))
      POM=(DEN-POM)/(ONE-POM)
      TUXX='NEW     '
      GO TO 20
      ENDIF
   17 CONTINUE
*
*     BFGS UPDATE
*
      POM=ONE
```

```
      DIS=PAR+AA
      CALL UXVDIR(N,-DIS,XO,S,XO)
      DIS=ONE/(B*DIS)
      CALL UXDSMU(N,H,DIS,XO)
      CALL UXDSMU(N,H,-DIS,S)
      TUXX='BFGS    '
      GO TO 21
   20 CONTINUE
*
*     GENERAL UPDATE
*
      DEN=PAR+POM*AA
      DIS=POM/DEN
      CALL UXDSMU(N,H,(PAR*DIS-ONE)/A,S)
      CALL UXVDIR(N,-DIS,S,XO,S)
      CALL UXDSMU(N,H,DEN/B,S)
   21 CONTINUE
      ITERH=0
      IF (GAM.EQ.ONE) GO TO 22
*
*     SCALING
*
      CALL UXDSMS(N,H,GAM)
   22 CONTINUE
      CALL UOU1D2(N,H,S,RHO,GAM,PAR,A,B,C,POM,ETA9)
      RETURN
      END
```

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

```
   0 NIT=   40 NFV=  138 NFG=    0  GRAD TOL  F=  .5038712822E-13 G= .828D-05
 FF =   .5038712822D-13
 X  =   .1000000098D+01   .1000000177D+01
```

B The BEL interpreter

The BEL (Batch Editor Language) interpreter, developed as a part of the UFO project, is especially determined for the generation of computer programs, batch editing of texts, preparation of print files, filtering of text files etc. The BEL interpreter allows us to generate a prescribed output file from the input file (template), which is a mixture of text lines and special instructions.

B.1 General description

The BEL interpreter, realized by code `BEL.EXE`, requires input text file `BEL.TEM`, output text file `BEL.OUT` and an internal table of symbols. The input text file (template) consists of standard text lines together with the BEL instructions. The output text file contains a text generated by the BEL interpreter. The table of symbols contains names and values of the macrovariables used.

Although the BEL interpreter can be used in various general applications, it was developed especially for the generation of FORTRAN programs. It is:

1. Interpreter; since instructions contained in the input text are interpreted and immediately realized.
2. Batch editor; since it serves for editing batch files.
3. Macroprocessor; since it makes it possible to define or modify special macrovariables which can be substituted into the processed text.

The macrovariable can be an integer constant, a logical constant, a string of characters, a set of text lines, a set of BEL instructions, even a text file.

The BEL instructions, contained in the input text file, can be of two types:

1. Directives, i.e. control instructions and instructions for manipulation with the table of symbols. These instructions begin with the special character CHDIR. In the subsequent text, we will suppose that CHDIR='$' ('$' is the default value).
2. Substitutions, i.e.instructions for substituting macrovariables into the text. These instructions begin with the special character CHSUB. In the subsequent text, we will suppose that CHSUB='$' ('$' is the default value).

The BEL interpreter works in the following way:

1. The line of the input file is read.
2. The line is recognized and if the character CHSUB is found, a pertinent substitution is realized.
3. If the first character (different from blank) is CHDIR, the line is a directive line. The recognized directive is realized.

This process is repeated until directives $STOP, $EXIT or the end of the input file is found. Note that we suppose that CHSUB and CHDIR have the same values. This is allowed since the correct meaning is recognized from the context.

At the end of this section, we stress some specific features and advantages of the BEL interpreter.

1. The substitution is recursive. The depth of recursion (the number of nested files) only depends on the declared work space size.
2. Substitution is allowed in both the text lines and the directives.
3. The names and values of macrovariables can have an arbitrary length which again only depends on the declared work space size.

4. The set of directives is relatively small with a consistent syntax. It contains all important instructions ($IF-$ELSEIF-$ELSE-$ENDIF, $DO-$ENDDO, $REPEAT-$UNTIL etc.)

5. The control parameters (CHDIR, CHSUB etc.) can be changed during the work of the BEL interpreter. This makes it possible to generate a program written in the BEL language which can be immediately processed.

6. The optional parameter of the BEL interpreter (e.g. the number of the I/O unit) can be changed during its work. This possibility enables us to change the input file without recursive nesting.

7. The BEL interpreter is a fully portable device. It can be implemented in an arbitrary system containing FORTRAN 77 compiler.

B.2 List of instructions

Substitutions:

$INTEGER	- Substitute by the absolute label computed from the relative label.
$NAME, $(NAME)	- Substitute by the value of the macrovariable NAME.
$DATA(NAME)	- Substitute by a new item from the list of items which is a value of the macrovariable NAME.
$DEF(NAME)	- Substitute by '.TRUE.' if the macrovariable NAME is defined in the table of symbols. Otherwise substitute by '.FALSE.'
$INT(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is an integer constant. Otherwise substitute by '.FALSE.'
$LOG(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is a logical constant. Otherwise substitute by '.FALSE.'
$REAL(NAME)	- Substitute by '.TRUE.' if the value of the macrovariable NAME is a real constant. Otherwise substitute by '.FALSE.'
$$	- Substitute '$' (replace '$$' by '$'). This makes possible to insert the character CHSUB into the text.

Directives:

$ADD	- Add a value to a macrovariable.
$ADD, $ENDADD	- Add text lines to a macrovariable.
$CLEAR	- Clear value of a macrovariable which is a list of items type.
$DO, $ENDDO	- Cycle.
$ERASE	- Erase a macrovariable from the table of symbols.
$EXIT	- Return from the deepest nested file or termination of the BEL interpreter work if there are no nested files.
$HELP, $CHECK	- Set a default value to a macrovariable which has not been previously defined.
$IF, $ELSEIF, $ELSE, $ENDIF	- Conditioned instruction.
$INCLUDE	- Insert a macrovariable or a text file into the output file.
$OPTION	- Change some optional parameter of the BEL interpreter.
$REM	- Remark.
$REPEAT, $UNTIL	- Cycle.
$RESTORE	- Adjust the list of items pointer to the first item.
$REWIND	- Rewind the file on a given unit.
$SET	- Set a value to a macrovariable.
$SET, $ENDSET	- Set text lines to a macrovariable.
$STOP	- Termination of the BEL interpreter work.
$SUBST	- Substitute a text file into the input file.

B.3 Special characters

The following special characters are important for the BEL interpreter work:

$ - CHSUB (Substitution Character) - this is the first character in every substitution. If '$' should be inserted into the text, we have to use '$$'.

$ - CHDIR (Directive Character) - if the first character on the line is CHDIR, then the line is a directive line (CHSUB and CHDIR are distinguished by the context).

& - CHCON (Continuation Character) - if the last character on the line is CHCON, then it is assumed that the logical line continues on the next physical line.

; - CHEOL (End Of Line Character) - this character specifies the end of the logical line if it does not coincide with the end of the physical line. This makes it possible to write several logical lines by using the same physical line.

\ - CHDS (Data Separator Character) - this character separates individual items in the list of items type macrovariable.

The use of special characters can be demonstrated by the following simple example. Assume that the input text has the form

```
$A='Paul\Peter\Jane\Mary'
This is a list of my brothers and sis&
ters:
$DO(I=1,4); $DATA(A); $ENDDO
```

Then the output from the BEL interpreter has the form

```
This is a list of my brothers and sisters:
Paul
Peter
Jane
Mary
```

The special characters can be changed by the directive $OPTION. But no special character has to be the alphabet or the digit. Moreover, different special characters have to differ (with the exception of CHSUB and CHDIR).

B.4 Description of instructions

This section contains a detailed description of the syntax and action of individual BEL instructions. The following definitions will be used:

<digit> ::= 0 | 1 | 2 | 3 | ..... | 9

<alphabet> ::= A | B | C | D | ..... | Z

<character> ::= an arbitrary character with the exception of apostrophe

<integer constant> ::= (+ | -) <digit> {<digit>}

<logical constant> ::= .TRUE. | .FALSE.

<macroname> ::= <alphabet> {<alphabet> | <digit>}

<string of characters> ::= '{<character> | ''}

<text> ::= <string of characters> '{; <string of characters>}

<list of items> ::= <string of characters> '{\ <string of characters>}

Substitutions:

$INTEGER

Syntax:
The type of INTEGER is an integer constant. Although it can have an arbitrary value, an application to the UFO source program generation requires it to be positive and lower than LABEL2 (see the directive $OPTION).

Action:
The integer constant INTEGER is a relative label in a given template. The absolute label, substituted into the UFO source program, is computed by the formula LABEL=LABEL1+K∗LABEL2, where LABEL1 and LABEL2 are options of the BEL interpreter (see the directive $OPTION), and K is a serial number of the application of the directive $SUBST.

Example:

$10

generates

10010

if the main template is used or

10110

after the first application of the directive $SUBST.

$NAME, $(NAME)

Syntax:
The type of NAME is a macroname. This substitution has two forms, either $NAME or $(NAME). The latter form is required if the substitution appears inside a continuous string of characters to separate the NAME from the adjacent text.

Action:
The string '$NAME' is replaced by the value of the macrovariable NAME.

Example:

$A='UFO'
$A SYSTEM

generates

UFO SYSTEM

$DATA(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The string '$DATA(NAME)' is replaced by the next item of the list of items, which is a value of the macrovariable NAME. If the next item does not exist, the list of items pointer is returned to the first item. Additional information is contained in the description of the directive $RESTORE.

Example:

$LIST='ITEM1\ITEM2\ITEM3'
$DATA(LIST)
$DATA(LIST)

$DATA(LIST)
$DATA(LIST)

generates

ITEM1
ITEM2
ITEM3
ITEM1

$DEF(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the macrovariable NAME is defined in the table of symbols, the string '$DEF(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=10
$DEF(A)

generates

.TRUE.

$INT(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is an integer constant, the string '$INT(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=-25
$INT(A)

generates

.TRUE.

$LOG(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is a logical constant, the string '$LOG(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

$A=.FALSE.
$LOG(A)

generates

.TRUE.

$REAL(NAME)

Syntax:
The type of NAME is a macroname.

Action:
If the value of the macrovariable NAME is a real constant (i.e. a string of characters which satisfies the syntactic rules for FORTRAN real constants), the string '$REAL(NAME)' is replaced by the logical constant .TRUE., otherwise it is replaced by the logical constant .FALSE..

Example:

```
$A='-0.09D-12'
$REAL(A)
```

generates

.TRUE.

$$

Action:
The string '$$' is replaced by the character '$'. This substitution allows us to insert the character '$' into the generated text or into the macrovariable.

Example:

```
$I='NAME'
$$DEF($I)
```

generates

$DEF(NAME)

Directives:

$ADD(NAME1,NAME2 or VALUE)

Syntax:
The type of NAME1 and NAME2 is a macroname.
The type of VALUE is an integer constant or a logical constant or a string of characters.

Action:
The value of the macrovariable NAME2 or the VALUE is added to the value of the macrovariable NAME1 (the resulting value of the macrovariable NAME1 is $NAME1$NAME2 in the first case).

Example:

```
$NAME='TOM'
$ADD(NAME,' JONES')
Name: $NAME
```

generates

Name: TOM JONES

$ADD(NAME)
TEXT
$ENDADD

Syntax:
The type of NAME is a macroname.
The type of TEXT is a text.

Action:
The TEXT is added to the value of the macrovariable NAME.

Example:

```
$SET(A)
   Day: 31
$ENDSET
$ADD(A)
   Month: December
   Year: 1998
$ENDADD
$A
```

generates

```
   Day: 31
   Month: December
   Year: 1998
```

Remark: Only substitutions are realized in the text TEXT (not directives).

$CLEAR(NAME)

Syntax:
The type of NAME is a macroname.

Action:
This directive clears a list-of-items-type value of the macrovariable NAME, i.e. it deletes all duplications of items. Small and capital letters of items are not distinguished.

Example:

```
$DECL='N\IX(N)\N\M\ I\J\N\M'
$CLEAR(DECL)
$END='$DATA(DECL)'
$REPEAT
   $I='$DATA(DECL)'
   INTEGER $I
$UNTIL(I=END)
```

generates

```
   INTEGER IX(N)
   INTEGER M
   INTEGER I
   INTEGER J
   INTEGER N
```

$DO(NAME=INDEX1,INDEX2,INDEX3)
TEXT
$ENDDO

Syntax:
The type of NAME is a macroname.
The type of INDEX1, INDEX2, INDEX3 is a macroname or an integer constant.
The type of TEXT is a text.

Action:
This directive has a similar meaning as the statement DO in the FORTRAN language:

NAME is the cycle counter.
INDEX1 is the initial value of the cycle counter.
INDEX2 is the final value of the cycle counter.
INDEX3 is the change of the cycle counter after a cycle step.
If INDEX3 is not present, the default value INDEX1=1 is assumed.
The cycle counter NAME does not have to be changed in the cycle step.
The value INDEX3 does not have to be equal to 0.
The body of the cycle is terminated by $ENDDO.
If INDEX1>INDEX2 and INDEX3>0 or INDEX1<INDEX2 and INDEX3<0, then the cycle is not realized.
Cycles can be nested. The maximum depth of nested cycles is 20.

Example:

```
$A='X\Y\Z'
$DO(I=1,5,2)
   A($I,1)=C($I)+$DATA(A)
$ENDDO
```

generates

```
   A(1,1)=C(1)+X
   A(3,1)=C(3)+Y
   A(5,1)=C(5)+Z
```

$ERASE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The macrovariable NAME is erased from the table of symbols.

Example:

```
$A=1
$DEF(A)
$ERASE(A)
$DEF(A)
```

generates

```
   .TRUE.
   .FALSE.
```

$EXIT

Action:
The directive $EXIT has the same meaning as the end of the file achievement. If the nested files are processed (see the description of the directive $SUBST), the directive $EXIT realizes return to the higher level file (if the higher level file does not exist, then $EXIT has the same meaning as $STOP).

$HELP
TEXT
$CHECK(NAME,DEFAULT,TYPE,LEVEL,TRANSFER)

Syntax:
The type of TEXT is a text.
The type of NAME is a macroname.

The type of DEFAULT is either a macroname or an integer constant or a logical constant or a string of characters.
The type of TYPE is either a list of items or one of the strings INT (integer), LOG (logical), REAL (real).
The type of LEVEL is an integer constant.
The type of TRANSFER is a logical constant.

Action:
The text TEXT appears on the screen if the dialogue mode is used. The value of the macrovariable $NAME is checked to have the type TYPE. If the macrovariable $NAME is not defined or if it has a wrong value, the value DEFAULT is used. The value of LEVEL gives the lowest level of the dialogue (1,2,3 or 4) from which the text TEXT appears on the screen. The value of TRANSFER specifies transfer of the variable $NAME into the UFO source program (YES if transfer is accepted or NO if transfer is suppressed).

Example:

```
$HELP
   TYPE OF THE HESSIAN MATRIX:
         D - DENSE
         B - SPARSE WITH KNOWN (PARTITIONED) STRUCTURE
         S - SPARSE WITH KNOWN (GENERAL) STRUCTURE
         N - HESSIAN MATRIX IN NOT USED
$CHECK(HESF,'N','D\B\S\N',1,NO)
```

$IF(CONDITION) LINE

Syntax:
The CONDITION can be of the following types:
The type of CONDITION is a macroname and a value of CONDITION is a logical constant.
The type of CONDITION is a logical constant (.TRUE. or .FALSE.).
The type of CONDITION is a string of the form PART1<operator>PART2.
The type of PART1 and PART2 can be a macroname or an integer constant or a logical constant or a string (values of PART1 and PART2 have to be of the same type) and <operator> can have the following forms:

= equal to
<> not equal to
< less than (for integer values only)
<= less than or equal to (for integer values only)
> greater than (for integer values only)
>= greater than or equal to (for integer values only)

LINE is either a text line or a directive.

Action:
If the condition CONDITION is satisfied, LINE is inserted into the output file (if it is a text line) or carried out (if it is a directive). If the values of PART1 and PART2 are strings, then small and capital letters are not distinguished and blanks are ignored.

Example:

```
$A='J O H N'
$IF(A='John') Yes
$IF(A<>'Mary') No
```

generates

```
   Yes
   No
```

$IF(CONDITION1)
TEXT1
$ELSEIF(CONDITION2)
TEXT3
.
.
$ELSE
TEXT
$ENDIF

Syntax:
CONDITION1 and CONDITION2 have the same syntax and meaning as CONDITION in the previous case. The number of repeated $ELSEIF is not limited, $ELSEIF or $ELSE can be omitted.

Action:
This directive has a similar meaning as the conditioned statement IF-ELSEIF-ELSE-ENDIF in the FORTRAN language. The conditioned statements can be nested. The maximum depth of nested conditioned statements is 20.

Example:

```
$A=10
$L=.FALSE.
$IF(A=10)
    A = A + 1
    B = B + 1
    $IF(L)
    C = C + 1
    $ENDIF
$ELSE
    WRITE(*,*) I
$ENDIF
```

generates

```
    A = A + 1
    B = B + 1
```

$INCLUDE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The directive $INCLUDE(NAME) is a special case of substitution. This directive makes it possible to insert (into the generated text) one or more lines, which were previously assigned to the macrovariable NAME. In contrast to the standard substitution $NAME, the inserted lines are not processed by the BEL interpreter, so the directives are not carried out.

Example:

```
$SET(LINES)
    $ADD(A)
    X = Y + Z
    CALL SUB(X)
    $ENDADD
$ENDSET
$INCLUDE(LINES)
```

generates

```
$ADD(A)
X = Y + Z
CALL SUB(X)
$ENDADD
```

$INCLUDE('FILE')

Syntax:
The type of FILE is a string.

Action:
The directive $INCLUDE('FILE') is a special case of substitution. This directive makes it possible to insert (into the generated text) the text which is stored in the file with the name FILE. The inserted text is not processed by the BEL interpreter, so the directives are not carried out.

Example:

$INCLUDE('C:\UFO\UMCOMN.I')

includes FORTRAN common blocks into the generated text (these common blocks are stored in the file `C:\UFO\UMCOMN.I`.

$OPTION(OPTIONNAME=NAME or VALUE)

Syntax:
OPTIONNAME is a selected name from the table of optional parameters (see below).
The type of NAME is a macroname. The value of NAME has to be an integer constant or a logical constant or a string of character and has to correspond to the type of OPTIONNAME.
The type of VALUE has to be an integer constant or a logical constant or a string of character and has to correspond to the type of OPTIONNAME.

Action:
This directive makes us possible to change selected optional parameter of the BEL interpreter. Optional parameters are contained in the following table.

Name	Type	Default	Description
CHDIR	char.	'$'	see B.3
CHEOL	char.	';'	see B.3
CHCON	char.	'&'	see B.3
CHDS	char.	'\'	see B.3
FN1	char.	' '	first part of the file name
FN2	char.	'.I '	last part of the file name
ILNLEN	int.	80	physical length of the input line
OLNLEN	int.	80	physical length of the output line
IUNIT	int.	-	No. of the input file unit
OUNIT	int.	-	No. of the output file unit
INUNIT	int.	-	No. of the $INCLUDE files unit
IIUNIT	int.	-	No. of the interactive mode input unit
OIUNIT	int.	-	No. of the interactive mode output unit
DIALOG	int.	1	level of dialogue (0 or 1 or 2)
MODERW	int.	1	READ/WRITE mode (1 or 2 or 3)
LABEL1	int.	10000	initial label
LABEL2	int.	100	difference between two consecutive labels
LSUBS	log.	.TRUE.	substitutions carried out
LOUT	log.	.TRUE.	output file created
LSMLET	log.	.TRUE.	small letters used in instructions
LFORTO	log.	.TRUE.	output in standard FORTRAN format
LFRFMT	log.	.TRUE.	input in free FORTRAN format (used only if LFORTO=.TRUE.)
SIFDEC	log.	.FALSE.	using the SIF decoder
DIALGR	log.	.FALSE.	using the graphic dialogue

$REM

Action:
The rest of the line (following after $REM) is ignored by the BEL interpreter. The directive $REM is used for remarks.

$REPEAT
TEXT
$UNTIL(CONDITION)

Syntax:
The type of TEXT is text.
CONDITION has the same syntax and meaning as that in the directive $IF(...).

Action:
This directive has a similar meaning as the statement REPEAT-UNTIL in the PASCAL language:
The cycle is terminated whenever the condition CONDITION is satisfied (at least one realization is carried out).
Cycles can be nested. The maximum depth of nested cycles is 20.

Example:

```
$N=10
$REAL='X($N)\G($N)\H($N,$N)\.END.'
$REPEAT
   $I='$DATA(REAL)'
   $IF(I<>'.END.') REAL $I
$UNTIL(I='.END.')
```

generates

```
REAL X(10)
REAL G(10)
REAL H(10,10)
```

$RESTORE(NAME)

Syntax:
The type of NAME is a macroname.

Action:
The directive $RESTORE(NAME) can only be used if the value of the macrovariable NAME is a list of items. Such a macrovariable uses a pointer which points out the next called item. The directive $RESTORE adjust this pointer to point out the first item of the list (if the end of this list is found, the pointer is adjusted to point out the first item without applying the directive $RESTORE).

Example:

```
$A='X\Y\Z'
   $DATA(A)
   $DATA(A)
$RESTORE(A)
   $DATA(A)
```

generates

```
X
Y
X
```

$REWIND(UNIT)

Syntax:
The type of UNIT is an integer constant.

Action:
The file opened on the unit with the number UNIT is rewound, so it can again be read from the first record (numbering of I/O units is used in the FORTRAN language).

$NAME1 = NAME2 or VALUE

$SET(NAME1 = NAME2 or VALUE)

Syntax:
The type of NAME1 and NAME2 is a macroname.
The type of VALUE is an integer constant or a logical constant or a string of characters.
This directive has two forms. The latter form is used if the macroname is identical with a directive (e.g. $SET(REM='REMARK')).

Action:
The new macrovariable with the name NAME1 and the value equal to the value of the macrovariable NAME2 or constant VALUE is inserted into the table of symbols. If the macrovariable NAME1 has already been defined in the table of symbols, then it is changed.

```
$SET(NAME)
   TEXT
$ENDSET
```

Syntax:
The type of NAME is a macroname.

The type of TEXT is text.

Action:
The macrovariable NAME is inserted into the table of symbols with the value TEXT. If the macrovariable NAME has already been defined in the table of symbols, then it is changed.

Example:

```
$ENDTEST=100
$SET(INIT)
   CALL EIUD01(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
   IF (IERR.NE.0) GO TO $ENDTEST
$ENDSET
$INIT
```

generates

```
   CALL EIUD01(NF,X,FMIN,XMAX,NEXT,IEXT,IERR)
   IF (IERR.NE.0) GO TO 100
```

Remark: Only substitutions are realized in the text TEXT (not directives).

$STOP

Action:
The directive $STOP terminates the BEL interpreter work.

$SUBST('FILE')

Syntax:
The type of FILE is a string.

Action:
This directive performs the following actions:
The new reference label is computed (using the parameters LABEL1 and LABEL2 of the BEL interpreter).
The file with the name FILE is opened.
This file is processed by the BEL interpreter.
The file with the name FILE is closed.
The old reference label is restored.

This directive is similar to the directive $INCLUDE('FILE'). But the inserted text is now processed by the BEL interpreter. All substitutions and directives are carried out. The directive $SUBST('FILE') serves for dividing large texts into segments and makes it possible to generate texts by using conditioned branching. This is advantageously used for generation of the UFO source program where templates corresponding to individual subroutines are such segments.

Example:

```
$SUBST('C:\UFO\PROBLEM.UFO')
```

inserts a template, written in the UFO control language, into the generated text (this template is stored in the file `C:\UFO\PROBLEM.UFO`).

B.5 Error messages

The BEL interpreter checks input templates and detects errors in the text written by the batch editing language. A typical error message has the form

```
+++ FATAL ERROR:     9
    FILE:  UZMETH.I
    LINE:   295
    CAUSE: MACRO NOT DEFINED
    NAME:  COLLECTION
+++ EXECUTION ABORTED
```

Here UZMETH.I is the template where the error was detected, 9 is the error number, 295 is the number of line containing incorrect text, MACRO NOT DEFINED is the explanation and COLLECTION is the name of undefined macrovariable.

The following table presents all UFO error messages (error numbers and explanations):

```
 1 - INPUT FILE DOES NOT EXIST OR I/O ERROR
 2 - MAX. LENGTH OF BUFFER EXCEEDED
 4 - MAX. LENGTH OF SYMBOL TABLE EXCEEDED
 5 - ATTEMPT TO APPEND TO READ ONLY VARIABLE
 6 - ATTEMPT TO APPEND TO UNDEFINED VARIABLE
 7 - ATTEMPT TO OVERWRITE OR DELETE UNDEFINED VARIABLE
 8 - ATTEMPT TO OVERWRITE OR DELETE RO OR RA VARIABLE
 9 - MACRO NOT DEFINED
10 - OUTPUT HARDWARE ERROR
11 - WRONG FORMAT OF FORTRAN LINE
14 - MAX. NUMBER OF POINTERS TO SYMBOL TABLE EXCEEDED
15 - UNEXPECTED CHDS
16 - MAX. VALUE OF LABEL EXCEEDED
17 - UNEXPECTED CHDIR
20 - SYNTAX ERROR
21 - '(' EXPECTED
22 - '(' EXPECTED
23 - MACRO NAME EXPECTED
24 - APOSTROPHE EXPECTED
25 - EOL EXPECTED
26 - INTEGER CONSTANT EXPECTED
27 - MEMORY EXHAUSTED
28 - INCLUDED FILE DOES NOT EXIST OR I/O ERROR
29 - '=' EXPECTED
30 - IDENTIFIER, STRING, INTEGER OR LOGICAL CONSTANT EXPECTED
31 - ',' EXPECTED
32 - ERROR IN DIRECTIVE OPTION
33 - UNEXPECTED 'ENDSET'
34 - UNEXPECTED 'ENDADD'
35 - UNEXPECTED 'ENDIF'
36 - UNEXPECTED 'ELSE'
37 - UNEXPECTED 'ELSEIF'
38 - TOO MANY NESTED 'IF-ENDIF'
39 - REWIND - I/O ERROR
40 - TOO MANY NESTED 'SUBST(FILE)'
41 - SUBSTITUTED FILE DOES NOT EXIST OR I/O ERROR
42 - TOO MANY NESTED 'REPEAT-UNTIL'
43 - UNEXPECTED 'UNTIL'
44 - IN DO(I=IND1,IND2,IND3) IS IND3=0
45 - UNEXPECTED 'ENDDO'
46 - TOO MANY NESTED 'DO-ENDDO'
```

```
    47 - UNEXPECTED END OF FILE
    48 - AUTOMATICAL DIFFERENTIATION ERROR
```

B.6 Organization and compilation of the UFO source modules

The BEL interpreter driven by the special input template `UZDPRE.I` can be used for preprocessing UFO source modules. The UFO source modules `*.I` usually consist of two parts. The first part is a template written in the UFO control language. This template, which begins with substitution $INTERFACE and terminates by directive $EXIT, serves for generation of the UFO source program by the UFO control language preprocessor (described in Section B.7). The second part is an actual source module which can contain statements of the UFO control language, e.g., substitutions $FLOAT, $P, $DBLE for defining the precision of the final Fortran 77 source code (single or double). The BEL interpreter driven by template `UZDPRE.I` skips the interface template and starts the processing on the line containing remark *IMPLEMENTATION. If the first part is missing, the remark *IMPLEMENTATION can be omitted. If the second part is missing, the macrovariable $INTERFACE and directive $EXIT need not be used. The more details can be obtained by reading template `UZDPRE.I`:

```
$REM +-------------------------------------------------------------+
$REM +  TEMPLATE: UZDPRE                                  01/12/89  +
$REM +                                                              +
$REM +  PURPOSE:  DRIVER FOR UFO SOURCE-MODULE PREPROCESSOR         +
$REM +-------------------------------------------------------------+

$REM +----+ INSTALATION +-----------------------------------------+

$P='D'
$FLOAT='REAL*8'
$DBLE='DBLE'
$GRAPHICS=6
$OPENFILE=.TRUE.
$OPTION(FN2='.I')

$REM +----+ INITIATION +-------------------------------------------+

$OPTION(LFRFMT=.FALSE.)
$NEXT=0
$SET(INTERFACE)
  $OPTION(LOUT=.FALSE.)
  $OPTION(LSUBS=.FALSE.)
  $OPTION(LFORTO=.FALSE.)
  $OPTION(CHDIR='*')

$ENDSET
$IF($DEF(PREDAT))
  $PREDAT
$ENDIF

$REM +----+ GRAPHIC INTERFACE +-------------------------------------+

$SET(FIFAC)
  $IF(GRAPHICS=0)
  $ELSEIF(GRAPHICS=1)
```

```
      INCLUDE 'FGRAPH.FI'
      INCLUDE 'FLIB.FI'
  $ELSEIF(GRAPHICS=4)
      INCLUDE 'FGRAPH.FI'
      INCLUDE 'FLIB.FI'
  $ELSE
  $ENDIF
$ENDSET
$SET(FDECL)
  $IF(GRAPHICS=0)
  $ELSE
    $IF(GRAPHICS=1)
      INCLUDE 'FGRAPH.FD'
      INCLUDE 'FLIB.FD'
    $ELSEIF(GRAPHICS=4)
      INCLUDE 'FGRAPH.FD'
      INCLUDE 'FLIB.FD'
    $ELSE
      USE DFLIB
    $ENDIF
      STRUCTURE /MYITEM/
      CHARACTER*10 NAZEV
      INTEGER POZICE
      INTEGER*2 DELKA
      INTEGER BARVA
      INTEGER HI
      ENDSTRUCTURE
  $ENDIF
$ENDSET

$REM +----+ INITIATION OF SPECIAL MACROVARIABLES +-------------------+

$SUBST('UZSETM')

$OPTION(IUNIT=9)
```

The first several statements of this template are system dependent and define an individual UFO installation. Here $FLOAT='REAL*8', $P='D', $DBLE='DBLE' correspond to double precision Fortran 77 source code and $GRAPHICS=6 defines the graphics library used. Option $OPTION(FN2='.I') defines extensions of source modules which have names `*.I`.

Another part of template `UZDPRE.I` contains the initiation of important macrovariables used. Option $OPTION(LFRMT=.FALSE.) means that the input file is written in the free Fortran format (with possible non-Fortran characters). Options contained in macrovariable $INTERFACE have the following meaning: $OPTION(LOUT=.FALSE.) means that the interface will be ignored, $OPTION(LSUBS=.FALSE.) means that substitutions are not carried out, $OPTION(LFORTO=.FALSE.) means that the temporary output is written in the free format (with possible non-Fortran characters) and $OPTION(CHDIR='*') means that the directive character CHDIR will be '*' (thus remark *IMPLEMENTATION is interpreted as the directive which switches the BEL interpreter into the normal mode (with substitutions and records into the output file).

Finally, $FIFAC and $FDECL define the graphic interfaces and $OPTION(IUNIT=9) means that the input unit IUNIT is changed in such a way that the subsequent input data are read from file `BEL.DAT` (where source module `*.I` has to be copied).

If a source module *.I consists of two parts, the above process has one disadvantage. If the interface template contains remark

```
* remark
```

which should be copied into the UFO source program, then $OPTION(CHDIR='*') causes that it is understood as a directive. Therefore, the above remark has to be replaced by the sequence of statements

```
$CHPOM='$$'
$IF(.FALSE.)
*IF(.TRUE.)
*CHPOM='*'
*OPTION(CHDIR='$')
$ENDIF
* remark
$OPTION(CHDIR=CHPOM)
```

in the interface template. This is unnecessary if the processed template has only one part and substitution $INTERFACE is not used.

The output file BEL.OUT obtained by the BEL interpreter contains a resulting source module written in the standard Fortran format which can be copied into corresponding file *.FOR and compiled. All required steps, i.e., copying *.I into BEL.DAT, processing of BEL interpreter, copying BEL.OUT into *.FOR and compilation *.FOR to *.OBJ is realized by using the batch file PRE.BAT. The object module *.OBJ is added to the UFO library of object modules by using the batch file UF.BAT.

B.7 The UFO control language preprocessor

The UFO system is organized in such a way that the UFO source program P.FOR (or P.F) need not be written in Fortran 77 immediately. Instead, the procedure written in the UFO control language is supplied and the UFO control language preprocessor (UFOCLP), which is the BEL interpreter driven by the special input template UZDCLP.I, generates the resulting Fortran 77 source program. Besides template UZDCLP.I, UFOCLP uses many additional templates: user supplied template P.UFO, system templates UZ*.I, drivers for individual classes of methods U0*.I, U1*.I, U2*.I and other templates, the interfaces of files *.I, which realize various parts of the source program. Basic properties of UFOCLP can be understood by reading template UZDCLP.I:

```
$REM +-------------------------------------------------------------+
$REM +  TEMPLATE: UZDCLP                                  01/12/98  +
$REM +                                                             +
$REM +  PURPOSE:  DRIVER FOR UFO CONTROL LANGUAGE PREPROCESSOR     +
$REM +-------------------------------------------------------------+

$REM +----+ INSTALATION +-------------------------------------------+

$OPTION(MODERW=3)
$OPTION(ILNLEN=200)
$OPTION(OLNLEN=200)
$OPTION(FN2='.I')
$P='D'
$USERUNIT=12
$OUTPUTUNIT=11
$FLOAT='REAL*8'
$DBLE='DBLE'
```

```
$GRAPHICS=6
$OPTION(DIALGR=.TRUE.)

$REM +----+ OUTPUT UNIT FOR ERROR MESSAGES +------------------------+
$SET(ERROR)
$OPTION(OUNIT=6)
ERROR:
$ENDSET

$REM +----+ INITIATION +-----------------------------------------+

$OPTION(LSUBS=.FALSE.)


$REM +----+ INITIATION OF THE SYSTEM MACROVARIABLES +---------------+

$SET(BATCH)
  $OPTION(DIALOG=0)
  $OPTION(DIALGR=.FALSE.)
  $DIA=0
$ENDSET

$SET(GDIALOGUE)
  $OPTION(DIALOG=1)
  $OPTION(DIALGR=.TRUE.)
  $DIA=1
$ENDSET

$SET(TDIALOGUE)
  $OPTION(DIALOG=1)
  $OPTION(DIALGR=.FALSE.)
  $DIA=1
$ENDSET

$DIALOGUE=GDIALOGUE

$SET(STANDARD)
  $SUBST('UZSTAN')
$ENDSET

$SET(METHOD)
  $SUBST('UZMETH')
$ENDSET

$SET(TSTART)
  CALL UYTIM1(ITIME)
  $ADD(INTEGER,'\ITIME')
$ENDSET

$SET(TSTOP)
  CALL UYTIM2(ITIME)
$ENDSET
```

```
$NUMBERS='1\2\3\4\5\6\7\8\9\0'

$SET(VARERASE)
  CALL UYCLEA
$ENDSET

$SET(INITIATION)
  $SUBST('UZINIT')
$ENDSET

$SET(LABEL)
  $L CONTINUE
  $IF(TRACE='YES') $SUBST('UZTPNT')
$ENDSET

$INTERFACE='$REM'

$SUBST('UZSETM')

$OPTION(MODERW=2)

$INTEGER='.END'
$INTEGER2='.END'
$LOGICAL='.END'
$REAL='.END'
$REAL4='.END'

$OPTION(MODERW=1)
$DIA=1

$SET(MODERASE)
  $SUBST('UZMDER')
$ENDSET
$SET(METERASE)
  $SUBST('UZMTER')
$ENDSET
$SET(RUNERASE)
  $SUBST('UZMTER')
  $LMET=-1
$ENDSET

$SET(GLOBAL)
  $OPTION(LFORTO=.FALSE.)
  $$SUBST('UZDECL')
  $OPTION(LFORTO=.TRUE.)
  $ERASE(GLOBAL)
  $SET(UFOINPUT)
    $SUBST('UZINPT')
  $ENDSET
  $IF($DEF(DELETION))
    OPEN(7,FILE='P.PER',STATUS='REPLACE')
```

```
    CLOSE (7)
    $ERASE(DELETION)
  $ENDIF
$ENDSET

$REM $INPUT=' '

$SET(SUBROUTINES)

$ENDSET

$NEXT=0
$METHODINIT=.FALSE.
$OPTION(LSUBS=.TRUE.)

$REM +----+ 1. PASS +-----------------------------------------------+

$TMPUNIT=$OUNIT
$INITUNIT=$IUNIT
$OPTION(IUNIT=USERUNIT)
$OPTION(LFORTO=.FALSE.)
$$SUBST('UZIPAR')
$$SUBROUTINES

$REM +----+ 2. PASS +-----------------------------------------------+

$OPTION(LFORTO=.TRUE.)
$OPTION(OUNIT=OUTPUTUNIT)
$REWIND(TMPUNIT)
$OPTION(IUNIT=TMPUNIT)
```

The first several statements of this template are system dependent and define an individual UFO installation. Here $FLOAT='REAL*8', $P='D', $DBLE='DBLE' correspond to double precision Fortran 77 source code and $GRAPHICS=6 defines the graphics library used (option $OPTION(DIALGR=.TRUE.) determines the graphic dialogue). Option $OPTION(MODERW=3) means that the subsequent macrovariables cannot be rewritten in the symbol table, ILNLEN and OLNLEN are lengths of input line and output line, respectively and $OPTION(FN2='.I') defines extensions of intermediate templates which have names `*.I`. Option $OPTION(LSUBS=.FALSE.) means that substitutions are not carried out.

Another part of template `UZDCLP.I` contains initiation of system macrovariables:

$BATCH	- Switch to the batch mode.
$DIALOGUE	- Switch to the default dialogue mode (text or graphic).
$TDIALOGUE	- Switch to the text dialogue mode.
$GDIALOGUE	- Switch to the graphic dialogue mode.
$STANDARD	- Standard frame of the UFO source program.
$METHOD	- Generation of the optimization method.
$TSTART	- Start of the time measurement.
$TSTOP	- Termination of the time measurement and print of the measured time.
$VARERASE	- Clearing the common variables.
$INITIATION	- Initiation of the global variables.
$LABEL	- Labelling in the UFO source program with possible tracing.
$MODERASE	- Cancellation of the current model.
$METERASE	- Cancellation of the current method.

$GLOBAL - Global declarations.
$SUBROUTINES - User supplied subroutines.

The substituted template `UZSETM.I` contains a definition of additional important macrovariables, e.g., $UKMAI1, $UKMCI1, $SETAG, $SETCG. Options $OPTION(MODERW=2) and $OPTION(MODERW =1) mean that the subsequent macrovariables will be appended or rewritten in the symbol table, respectively.

After the initiation of system macrovariables, the first pass of UFOCLP is started. Thus, the input unit IUNIT is changed in such a way that the subsequent input data are read from file `P.UFO` and a temporary output is written into `BEL.OUT`. Option $OPTION(LFORTO=.FALSE.) means that the temporary output is written in the free format (with possible non-Fortran characters) and statements $$SUBST('UZIPAR'), $$SUBROUTINES mean that subroutines containing initiation of parameters and user supplied subroutines will be added to the UFO source program in the second pass.

In the second pass, the output temporary file `P.UFO` is rewound and the input unit IUNIT is changed in such a way that the subsequent input data are read from file `P.UFO`. The output unit OUNIT is changed in such a way, that the final source program is written into `P.FOR`. Option $OPTION(LFORTO=.TRUE.) means that the UFO source program is written in the standard Fortran format.

B.8 Basic templates and the UFO source program generation

This section contains information which can be useful for UFO administrators and builders. As it is shown in Section B.7, the generation of the UFO source program `P.FOR` (or `P.F`) starts by reading template `P.UFO`, where basic macrovariables defining the problem and the method used are defined and a frame of the UFO source program is described. The standard frame of the UFO source program is determined by substitution of macrovariable $STANDARD (synonym for template `UZSTAN.I`) realizing the sequence $GLOBAL, $INITIATION, $INPUT, $TSTART, $METHOD, $TSTOP, $OUTPUT. Here $INPUT contains user's input data (defined by using $SET(INPUT) and $ENDSET), $OUTPUT contains user's output data (defined by using $SET(OUTPUT) and $ENDSET), $TSTART and $TSTOP are points denoting start and stop of the time measurement, $METHOD, $INITIATION, $MODERASE, $METERASE are synonyms for templates `UZMETH.I`, `UZINIT.I`, `UZMDER.I`, `UZMTER.I` and $GLOBAL realizes global declarations by using template `UZDECL.I`. Global declarations are defined by list of items $INTEGER, $INTEGER2, $LOGICAL, $REAL, $REAL4 filled up in various templates by requirements of individual methods.

The UFO source program is generated by system templates `UZ*.I`, which call additional templates (interfaces of the UFO source modules `*.I`, see Section B.6). The basic system templates are:

`UZPROB.I` - Problem definition. Here macrovariables described in Section 2 are determined.

`UZMETH.I` - Method selection. Here input specifications are defined, possible tests of external subroutines are performed, and global or local optimization method (by using templates `UZGMIN.I` or `ULGMIN.I`) is specified.

`UZGMIN.I` - Global optimization method selection. Here a particular global optimization method and its parameters described in Section 3 are determined. Template `UZGMIN.I` usually calls template `UZLMIN.I`

`UZLMIN.I` - Local optimization method selection. Here a particular local optimization method and its parameters described in Section 3 are determined.

`UZLINS.I` - Step-size selection. Here a suitable line-search or trust-region method and its parameters described in Section 3 are determined.

`UZMODN.I` - Objective function definition with the choice of a way for computation of derivatives. If $NUMDER=1, first and second derivatives are computed by using the objective function. If $NUMDER=2, first and second derivatives are computed by using the approximating functions. This template usually calls template `UZMODL.I`.

`UZMODL.I` - Objective function definition. Here a suitable module is called, which is determined by the model used (described in Section 2). Also a structure of this model, which can be dense, sparse, partitioned, is taken into account.

There are many other system templates which have to be modified only in case the UFO system is substantially changed (e.g., templates `UZMOD*.I`, `UZ*MOD.I`, `UZ*SET.I` with '*' replaced by A,C,E,F,Y used for objective function definition or templates `UZMDER.I` and `UZMTER.I` for cancellation of the current model and method, respectively.) Individual system templates are listed in Section C.1

Template `UZLMIN.I` calls drivers for individual classes of methods, which have the form:

Udmscn.I,

where the character "d" corresponds to the degree of derivatives used (0, 1, 2), the character "m" denotes the model used:

F - general objective function or sum of approximating functions,
L - linear objective function,
Q - quadratic objective function,
S - sum of squares or powers of approximating functions,
A - sum of absolute values of approximating functions,
M - maximum of values of approximating functions (minimax),
C - general objective function with complementarity constraints,
E - system of equations,

the character "s" denotes the problem structure:

D - methods for dense problems,
S - methods for sparse problems,
B - methods for partially separable problems,
L - limited memory methods,

the character "c" corresponds to the type of constraints:

U - unconstrained problems,
L - linearly constrained problems,
N - recursive quadratic programming methods for general nonlinear programming problems,
E - recursive quadratic programming methods for problems with equality constraints,
I - interior point methods for general nonlinear programming problems,
F - nonsmooth equation methods for general nonlinear programming problems,
D - ordinary differential equations

and the character "n" corresponds to the driver number (1-6).

A typical driver template has the following form:

```
$INTERFACE U0FDU1                                              97/12/01

  $N='N'
  $ADD(INTEGER,'\IMD')
  $HELP
    TYPE OF METHOD:
      P - PATTERN SEARCH
      S - SIMPLEX METHOD
  $CHECK(TYPE,'P','P\S',2,NO)
  $IF(TYPE='P') $TYP='PS'
  $IF(TYPE='S') $TYP='SM'
  $IF(TYPE='P')
    $HELP
      NUMBER OF METHOD:
        1 - BASIC PATTERN SEARCH WITH SORTING
```

```
    $CHECK(NUMBER,'1','1',2,NO)
  $ELSE
    $HELP
      NUMBER OF METHOD:
        1 - BASIC SIMPLEX METHOD
    $CHECK(NUMBER,'1','1\2',2,NO)
  $ENDIF
  $ISNF=0
  $SUBST('UOOFU1')
*
*     ----------------
*     HEURISTIC METHOD
*     TEMPLATE: UOFDU1
*     ----------------
*
  $LMD=$10
  ASSIGN $20 TO IMD
  CALL UYPRO1('UXFU',0)
  CALL UYPRO2(FMIN,FO)
  N=NF
$L=$10 ; $LABEL
  $KD=0
  $SUBST('UZMODL')
  GO TO IMD
$L=$20 ; $LABEL
  $SUBST('UO2FU3')
  $SUBST('UYFUT1')
  IF(ITERM.NE.0) GOTO $50
$L=$30 ; $LABEL
  ASSIGN $30 TO IMD
*
*     ----------------------
*     DIRECTION DETERMINATION
*     TEMPLATE : UDD$(TYP)$NUMBER
*     ----------------------
*
  $SUBST('UDD$(TYP)$NUMBER')
*
*     ----------------------------
*     END OF DIRECTION DETERMINATION
*     ----------------------------
*
  GOTO ($40,$10) ISB+1
$L=$40 ; $LABEL
  ASSIGN $40 TO IMD
  $SUBST('US0$(TYP)$NUMBER')
  GOTO ($45,$10) ISB+1
$L=$45 ; $LABEL
  TXFU=TUXX
  IF(ITERM.EQ.0) GOTO $20
  TXFU=TDXX
$L=$50 ; $LABEL
```

```
CALL UYEPI1(0)
$SUBST('UO1FU2')
$ADD(INIT)
  NXFU='UOFDU1'
$ENDADD
$EXIT
```

The first part of this template contains definition of important macrovariables (parameters of the method selected), e.g., $TYPE and $NUMBER. The second part contains instructions for corresponding parts of the UFO source program. Here the objective function is defined by `UZMODL.I`, direction is determined by `UDDPS1.I` or `UDDSM1.I` and step-size is selected by `USOPS1.I` or `USOSM1.I`. Statements $L=$10, $L=$20 etc. define labels 11110, 11120 etc. in the generated source program and substitution $LABEL defines possible tracing. Modules `UOOFU1.I`, `UO1FU2.I` and `UO2FU3` serve for printing intermediate and final results. Modules `UYPRO1.I`, `UYPRO2.I`, `UYEPI1.I` serve for defining and clearing system variables and `UYFUT1.I` contains termination criteria. The form of the UFO source program is also influenced by macrovariables $INIT, $SINIT, $SINID and $CONST, which are substituted into the UFO source program by system templates `UZINIT.I`, `UZINPT.I` and `UZMETH.I`. The text from $INIT is substituted into `SUBROUTINE UYINT1`, which is a part of the UFO source program. The texts from $SINIT and $SINID are substituted before and after user's input $INPUT, respectively. The text from $CONST is substituted in such a way that statements of a generated method follow immediately.

Individual driver templates are listed in Section C.2

C List of important templates

C.1 List of system templates

The BEL interpreter templates:

`UZDPRE` - Template for the compilation of UFO source modules.
`UZDCLP` - Template for the UFO control language preprocessor.

Basic system templates:

`UZARRY` - Names of the basic arrays.
`UZDECL` - Global declaration.
`UZINIT` - Initiation for the UFO control program generation.
`UZINPT` - Substitution of the user input.
`UZIPAR` - Substitution of the subroutines serving for initiation of problem and method parameters.
`UZMDER` - The erase of the model macrovariables.
`UZMTER` - The erase of the method macrovariables.
`UZSETG` - Initiation of graphic choices.
`UZSETM` - Initiation of special macrovariables.
`UZSTAN` - Standard form of the UFO control program.
`UZTEST` - Testing of external subroutines.
`UZTPNT` - Tracing in the UFO control program.

Templates for the problem specification:

`UZPROB` - Problem definition.
`UZMODN` - Including the generalized objective function into the UFO control program.
`UZMODL` - Including the standard objective function into the UFO control program.
`UZAMOD` - Including the approximating function into the UFO control program.
`UZASET` - Finding parameters of the approximating function.
`UZMODA` - Definition of the approximating function.
`UZCMOD` - Including the constraint function into the UFO control program.
`UZCSET` - Finding parameters of the constraint function.
`UZMODC` - Definition of the constraint function.
`UZEMOD` - Including the state function into the UFO control program.
`UZESET` - Finding parameters of the state function.
`UZMODE` - Definition of the state function.
`UZFMOD` - Including the model function into the UFO control program.
`UZMODF` - Definition of the model function.
`UZYMOD` - Including the initial function into the UFO control program.
`UZYSET` - Finding parameters of the initial function.
`UZMODY` - Definition of the initial function.
`UZSIFD` - Selection of a test function from the CUTE collection.

Templates for the method selection:

`UZMETH` - Including the optimization method into the UFO control program.
`UZGMIN` - Choice of the global optimization method.
`UZGSLM` - Termination of the global optimization.
`UZLMIN` - Choice of the local optimization method.

UZLIND - Preparation of line search.
UZLINS - Choice of the line search method and including line search into the UFO control program.
UZSRND - Selection of the random number generator.

Templates for automatic differentiation:

UZADA1 - Automatic differentiation of the approximating function - the first derivatives.
UZADA2 - Automatic differentiation of the approximating function - the second derivatives.
UZADC1 - Automatic differentiation of the constraint function - the first derivatives.
UZADC2 - Automatic differentiation of the constraint function - the second derivatives.
UZADD1 - Declaration for automatic differentiation - the first derivatives.
UZADD2 - Declaration for automatic differentiation - the second derivatives.
UZADF1 - Automatic differentiation of the model function - the first derivatives.
UZADF2 - Automatic differentiation of the model function - the second derivatives.
UZADS1 - Basic template for automatic differentiation - the first derivatives.
UZADS2 - Basic template for automatic differentiation - the second derivatives.

C.2 List of driver templates

Drivers for unconstrained or box constrained minimization of general objective functions:

U0FDU1 - Heuristic methods for small-size unconstrained problems.
U1FBU1 - Variable metric methods for partially separable unconstrained or box constrained problems.
U1FDU1 - Variable metric methods for dense unconstrained or linearly constrained problems.
U1FDU2 - Conjugate direction methods for dense unconstrained or linearly constrained problems.
U1FDU3 - Proximal bundle methods for dense nonsmooth unconstrained problems.
U1FDU5 - Bundle variable metric methods for dense nonsmooth unconstrained or linearly constrained problems.
U1FLU1 - Conjugate gradient methods and variable metric methods with limited memory, based on the Strang recurrences, for unconstrained or box constrained problems.
U1FLU2 - Variable metric methods with limited memory, based on compact matrix updates, for unconstrained or box constrained problems.
U1FLU3 - Variable metric methods with limited memory, based on reduced Hessians, for unconstrained or box constrained problems.
U1FLU4 - Variable metric methods with limited memory, based on shifted product-form updates, for unconstrained or box constrained problems.
U1FLU5 - Variable metric methods with limited memory, based on projected product-form updates, for unconstrained or box constrained problems.
U1FLU7 - Variable metric methods with limited memory for nonsmooth unconstrained or box constrained problems.
U1FSU1 - Variable metric methods for sparse or partially separable unconstrained or box constrained problems.
U1FSU2 - Truncated Newton methods for unconstrained or box constrained problems.
U1FSU5 - Bundle variable metric methods for sparse nonsmooth unconstrained or box constrained problems.
U2FBU1 - Modified Newton methods for partially separable unconstrained or box constrained problems.
U2FDU1 - Modified Newton methods for dense unconstrained or linearly constrained problems.
U2FDU3 - Bundle Newton methods for dense nonsmooth unconstrained problems.
U2FSU1 - Modified Newton methods for sparse or partially separable unconstrained or box constrained problems.

Drivers for unconstrained or box constrained minimization of sum of squares or powers and for solving nonlinear equations:

`U0SDU1` - Quasi-Newton methods for solving dense nonlinear equations.
`U0SDU2` - Discrete Newton and Brent methods for solving dense nonlinear equations.
`U0SSU1` - Quasi-Newton methods for solving sparse nonlinear equations.
`U0SSU2` - Quasi-Newton methods with limited-memory for solving nonlinear equations.
`U0SSU3` - Truncated Newton methods for solving nonlinear equations.
`U1SDU1` - Modified Gauss-Newton methods, based on solving linear over-determined systems, for dense unconstrained or linearly constrained nonlinear least squares.
`U1SSU1` - Modified Gauss-Newton methods, based on solving linear over-determined systems, for sparse unconstrained or box constrained nonlinear least squares.
`U2SBU1` - Modified Gauss-Newton methods, based on solving normal linear systems, for partially separable unconstrained or box constrained nonlinear least squares.
`U2SDU1` - Modified Gauss-Newton methods, based on solving normal linear systems, for dense unconstrained or linearly constrained nonlinear least squares.
`U2SSU1` - Modified Gauss-Newton methods, based on solving normal linear systems, for sparse unconstrained or box constrained nonlinear least squares.

Drivers for unconstrained and box constrained minimization of special nonsmooth problems:

`U1ASU1` - Primal interior point variable metric methods for sparse unconstrained sum of absolute values.
`U2ASU1` - Primal interior point modified Newton methods for sparse unconstrained sum of absolute values.
`U1MDU1` - Sequential quadratic programming variable metric methods for dense unconstrained or linearly constrained minimax problems.
`U1MSU1` - Primal interior point variable metric methods for sparse unconstrained or box constrained minimax problems.
`U1MSU2` - Smoothing variable metric methods for sparse unconstrained minimax problems.
`U2MDU1` - Sequential quadratic programming variable metric methods for dense unconstrained or linearly constrained minimax problems.
`U2MSU1` - Primal interior point modified Newton methods for sparse unconstrained or box constrained minimax problems.
`U2MSU2` - Smoothing modified Newton methods for sparse unconstrained minimax problems.

Drivers for linearly constrained smooth problems:

`U1LDL1` - Simplex type methods for dense linear programming problems.
`U1LSL1` - Simplex type methods for sparse linear programming problems.
`U1LSL2` - Primal-dual interior point methods for sparse linear programming problems.
`U1QDL1` - Simplex type methods for dense quadratic programming problems.
`U1QSL1` - Simplex type methods for sparse quadratic programming problems.
`U1QSL2` - Primal-dual interior point methods for sparse quadratic programming problems.
`U1FSL1` - Sequential quadratic programming variable metric methods for sparse problems with general linear constraints.
`U1FSL2` - Conjugate direction methods for sparse problems with general linear constraints.
`U2FSL1` - Sequential quadratic programming modified Newton methods for sparse problems with general linear constraints.

Drivers for nonlinearly constrained smooth problems:

`U1FDN1` - Sequential quadratic programming variable metric methods for dense problems with general nonlinear constraints.

`U1FDN3` - Sequential minimax programming variable metric methods for dense problems with general nonlinear constraints.

`U2FDN1` - Sequential quadratic programming modified Newton methods for dense problems with general nonlinear constraints.

`U1FSE1` - Sequential quadratic programming variable metric methods for sparse problems with nonlinear equality constraints.

`U1FSE2` - Sequential quadratic programming variable metric methods with limited memory for sparse problems with nonlinear equality constraints.

`U1FSE3` - Sequential quadratic programming truncated Newton methods for sparse problems with nonlinear equality constraints.

`U2FSE1` - Sequential quadratic programming modified Newton methods for sparse problems with nonlinear equality constraints.

`U1FSF1` - Fisher-Burmeister nonsmooth equation variable metric methods for sparse problems with general nonlinear constraints.

`U1FSF2` - Fisher-Burmeister nonsmooth equation variable metric methods with limited memory for sparse problems with general nonlinear constraints.

`U1FSF3` - Fisher-Burmeister nonsmooth equation truncated Newton methods for sparse problems with general nonlinear constraints.

`U2FSF1` - Fisher-Burmeister nonsmooth equation modified Newton methods for sparse problems with general nonlinear constraints.

`U1FSI1` - Primal-dual interior point variable metric methods for sparse problems with general nonlinear constraints.

`U1FSI2` - Primal-dual interior point variable metric methods with limited memory for sparse problems with general nonlinear constraints.

`U1FSI3` - Primal-dual interior point truncated Newton methods for sparse problems with general nonlinear constraints.

`U2FSI1` - Primal-dual interior point modified Newton methods for sparse problems with general nonlinear constraints.

Drivers for nonlinearly constrained special nonsmooth problems:

`U1ADN1` - Sequential quadratic programming variable metric methods for dense sum of absolute values with general nonlinear constraints.

`U1ASE1` - Primal-dual interior point variable metric methods for sparse sum of absolute values with general nonlinear constraints (two slack reformulation).

`U1ASE2` - Primal-dual interior point variable metric methods with limited memory for sparse sum of absolute values with general nonlinear constraints (two slack reformulation).

`U1ASI1` - Primal-dual interior point variable metric methods for sparse sum of absolute values with general nonlinear constraints (one slack reformulation).

`U1ASI2` - Primal-dual interior point variable metric methods with limited memory for sparse sum of absolute values with general nonlinear constraints - one slack reformulation.

`U2ADN1` - Sequential quadratic programming modified Newton methods for dense sum of absolute values with general nonlinear constraints.

`U2ASE1` - Primal-dual interior point modified Newton methods for sparse sum of absolute values with general nonlinear constraints (two slack reformulation).

`U2ASI1` - Primal-dual interior point modified Newton methods for sparse sum of absolute values with general nonlinear constraints (one slack reformulation).

U1MDN1 - Sequential quadratic programming variable metric methods for dense minimax problems with general nonlinear constraints.
U1MSI1 - Primal-dual interior point variable metric methods for sparse minimax problems with general nonlinear constraints.
U1MSI2 - Primal-dual interior point variable metric methods with limited memory for sparse minimax problems with general nonlinear constraints.
U2MDN1 - Sequential quadratic programming modified Newton methods for dense minimax problems with general nonlinear constraints.
U2MSI1 - Primal-dual interior point modified Newton methods for sparse minimax problems with general nonlinear constraints.

Drivers for problems with complementarity constraints:

U1PSI1 - Primal-dual interior point variable metric methods with exact penalty function for sparse problems with nonlinear complementarity constraints.
U1PSI2 - Primal-dual interior point variable metric methods with limited memory with exact penalty function for sparse problems with nonlinear complementarity constraints.
U1PSI3 - Primal-dual interior point truncated Newton methods with exact penalty function for sparse problems with nonlinear complementarity constraints.
U2PSI1 - Primal-dual interior point modified Newton methods with exact penalty function for sparse problems with nonlinear complementarity constraints.

C.3 List of templates for direction determination

Direction determination for unconstrained minimization of general objective functions:

UDDPS1 - The Hooke-Jeeves pattern search method.
UDDSM1 - The Nelder-Mead simplex method.
UDDSM2 - The modified Nelder-Mead simplex method.
UDGGG1 - The dog-leg trust region method.
UDGGG2 - The multiple dog-leg trust region method.
UDGGM3 - The Steihaug-Toint trust region method.
UDGGM4 - The shifted Steihaug-Toint trust region method.
UDGGM5 - The simplified Lanczos type trust region method.
UDGGM6 - The Gould-Lucidi-Roma-Toint Lanczos type trust region method.
UDGGM7 - The Moré-Sorensen optimum trust region method.
UDGGM8 - The Hager trust region method that uses projections on subspaces.
UDGGM9 - The Rojas-Sorensen Arnoldi type trust region method.
UDGRM7 - The Gould-Toint optimum cubic regularization method.
UDDLI1 - Multiplication by an approximation of the inverse Hessian matrix.
UDGLG1 - The direct method based on Gaussian elimination.
UDGLG2 - The modified direct method based on Gaussian elimination.
UDGLM3 - The iterative conjugate gradient method.
UDPLI1 - Computation of direction vector for variable metric limited memory methods, based on shifted product-form updates.
UDPLI2 - Computation of direction vector for variable metric limited memory methods based on corrected product-form updates.
UDPLI3 - Computation of direction vector for variable metric limited memory methods based on projected product-form updates.
UDRLI1 - Computation of direction vector for variable metric limited memory methods based on reduced Hessians - standard form.

UDRLG1 - Computation of direction vector for variable metric limited memory methods based on reduced Hessians - factorized form.

UDVLC1 - Computation of direction vector for nonlinear conjugate gradient methods.

UDVLC2 - Computation of direction vector for modified nonlinear conjugate gradient methods.

UDVLV1 - Computation of direction vector for the vector-type BFGS limited memory method.

UDVLV2 - Computation of direction vector for the vector-type Liu-Storey limited memory method.

UDVLV3 - Computation of direction vector for the vector-type variable metric limited memory methods transformed to the BFGS form.

UDVLV4 - Computation of direction vector for the vector-type variable metric limited memory methods transformed to the BFGS form - simplified version.

UDVLV5 - Computation of direction vector for the vector-type variable metric limited memory methods that use vectors from the previous iteration.

UDVLV6 - Computation of direction vector for the vector-type variable metric limited memory methods that construct conjugate directions.

Direction determination for unconstrained minimization of nonlinear least-squares:

UDGGA1 - The dog-leg trust region method that uses the Jacobian matrix.

UDGGA3 - The CGLS based trust region method that uses the Jacobian matrix.

UDGGA4 - The LSQR based trust region method that uses the Jacobian matrix.

UDGGA7 - The Moré-Sorensen optimum trust region method that uses the Jacobian matrix.

UDGLA1 - The direct method that uses the orthogonal decomposition of the Jacobian matrix.

UDGLA3 - The iterative CGLS method that uses the Jacobian matrix.

UDGLA4 - The iterative LSQR method that uses the Jacobian matrix.

UDSGV3 - The CGLS based trust region method that uses the Jacobian matrix corrected by a simple Broyden update.

UDSLV3 - The iterative CGLS based method that uses the Jacobian matrix corrected by a simple Broyden update.

UDGTG1 - The modified dog-leg trust region method that uses the normal-equation matrix.

UDGTM7 - The modified Moré-Sorensen optimum trust region method that uses the normal-equation matrix.

UDGMM1 - The direct Levenberg-Marquardt method.

Direction determination for solving nonlinear equations:

UDDGQ2 - The multiple dog-leg trust region method for dense nonlinear equations.

UDGGE2 - The multiple dog-leg trust region method for sparse nonlinear equations.

UDGGE3 - The CGS based trust region method with double smoothing.

UDGGE4 - The GMRES based trust region method.

UDGGE5 - The BICGSTAB based trust region method.

UDGLE3 - The iterative CGS method with double smoothing.

UDGLE4 - The iterative GMRES method.

UDGLE5 - The iterative BICGSTAB method.

UDDEQ1 - The original Brent method.

UDDEQ2 - The modified Brent method.

UDDEQ3 - The simple discrete Newton method.

UDLLI1 - The inverse column-update method with complete LU factorization.

UDLLI3 - The inverse column-update method with incomplete LU factorization and smoothed CGS correction.

Direction determination for unconstrained minimization of special nonsmooth functions:

UDSGX1 - The dog-leg trust region method applied to the barrier function obtained by the primal interior point method - the minimax case.

UDSLX1 - Direct elimination method applied to the barrier function obtained by the primal interior point method - the minimax case.

UDSLX2 - Direct elimination method applied to the barrier function obtained by the exponential smoothing - the minimax case.

UDSLXI - Computing the Hessian matrix of the barrier function - the minimax case.

UDSGA1 - The dog-leg trust region method applied to the barrier function obtained by the primal interior point method - the sum of absolute values case.

UDSLA7 - The Moré-Sorensen trust region method applied to the barrier function obtained by the primal interior point method - the sum of absolute values case.

UDSLA1 - Direct elimination method applied to the barrier function obtained by the primal interior point method - the sum of absolute values case.

UDSLAI - Computing the Hessian matrix of the barrier function - the sum of absolute values case.

UDLLN1 - Computation of direction vector for nonsmooth matrix-type variable metric limited memory methods.

Direction determination for recursive quadratic programming methods for sparse problems with equality constraints:

UDSLK1 - The direct method based on Bunch-Parlett elimination applied to the KKT system.

UDSLK3 - The iterative conjugate gradient method applied to the KKT system.

UDSLK4 - The iterative conjugate residual method applied to the KKT system.

UDSLK5 - The iterative QMR method applied to the KKT system.

UDSLK6 - The iterative CGS method applied to the KKT system.

UDSLKA - Computation of the shift parameter for methods that use the KKT system.

UDSLKD - Computation of the directional derivative for methods that use the KKT system.

UDSLKI - Determination of the preconditioner for iterative methods that use the KKT system.

UDSLKP - Preconditioning of iterative methods that use the KKT system.

UDSLKT - Test for termination of iterative methods that use the KKT system.

UDSLG3 - The iterative conjugate gradient method with the Gill-Murray decomposition applied to the range-space system.

UDSLG4 - The iterative conjugate gradient method with the Bunch-Parlett decomposition applied to the range-space system.

UDSLGA - Computation of the shift parameter for methods that use the range-space system.

UDSLGD - Computation of the directional derivative for methods that use the range-space system.

UDSLGI - Determination of the preconditioner for iterative methods that use the range-space system.

UDSLGP - Preconditioning of iterative methods that use the range-space system.

UDSLGT - Test for termination of iterative methods that use the range-space system.

UDSGZ3 - The iterative locally constrained method applied to the null-space system.

UDSGZD - Computation of the directional derivative for locally constrained methods.

UDSLZ3 - The iterative conjugate gradient method applied to the null-space system.

UDSLZD - Computation of the directional derivative for methods that use the null-space system.

UDSLZI - Determination of the preconditioner for iterative methods that use the null-space system.

UDSLZP - Preconditioning of iterative methods that use the null-space system.

UDSLZR - Determination of vertical direction in methods that use the null-space system.

UDSLZT - Test for termination of iterative methods that use the null-space system.

Direction determination for primal-dual interior point methods for sparse nonlinear programming problems:

UDSGI3 - The iterative locally constrained method applied to the KKT system.
UDSGID - Computation of the directional derivative for locally constrained methods.
UDSLI1 - The direct method based on Bunch-Parlett elimination applied to the KKT system.
UDSLI3 - The iterative conjugate gradient method applied to the KKT system.
UDSLID - Computation of the directional derivative for methods that use the KKT system.
UDSLII - Determination of the preconditioner for iterative methods that use the KKT system.
UDSLIP - Preconditioning of iterative methods that use the KKT system.
UDSLIT - Test for termination of iterative methods that use the KKT system.
UDSLQ3 - The iterative conjugate gradient method applied to the KKT system when the objective function is quadratic.
UDSLQI - Determination of the preconditioner for iterative methods that use the KKT system when the objective function is quadratic.

Direction determination for nonsmooth equation methods for sparse nonlinear programming problems:

UDSLF1 - The direct method based on Bunch-Parlett elimination applied to the KKT system.
UDSLF3 - The iterative conjugate gradient method applied to the KKT system.
UDSLFD - Computation of the directional derivative for methods that use the KKT system.
UDSLFT - Test for termination of iterative methods that use the KKT system.

Direction determination for primal-dual interior point methods for sparse problems with complementarity constraints:

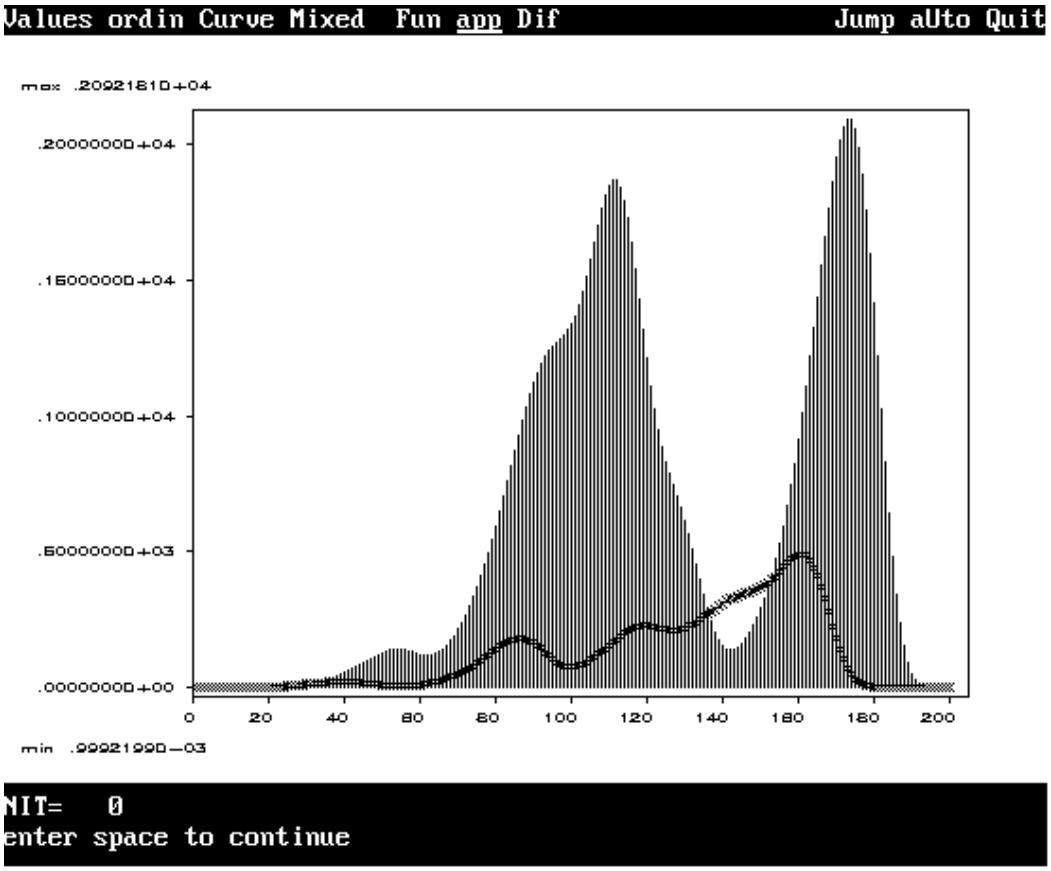
UDSLC1 - The direct method based on Bunch-Parlett elimination applied to the KKT system.
UDSLC3 - The iterative conjugate gradient method applied to the KKT system.
UDSLCD - Computation of the directional derivative for methods that use the KKT system.
UDSLCI - Determination of the preconditioner for iterative methods that use the KKT system.
UDSLCP - Preconditioning of iterative methods that use the KKT system.
UDSLCT - Test for termination of iterative methods that use the KKT system.

Solution of linear and quadratic programming subproblems:

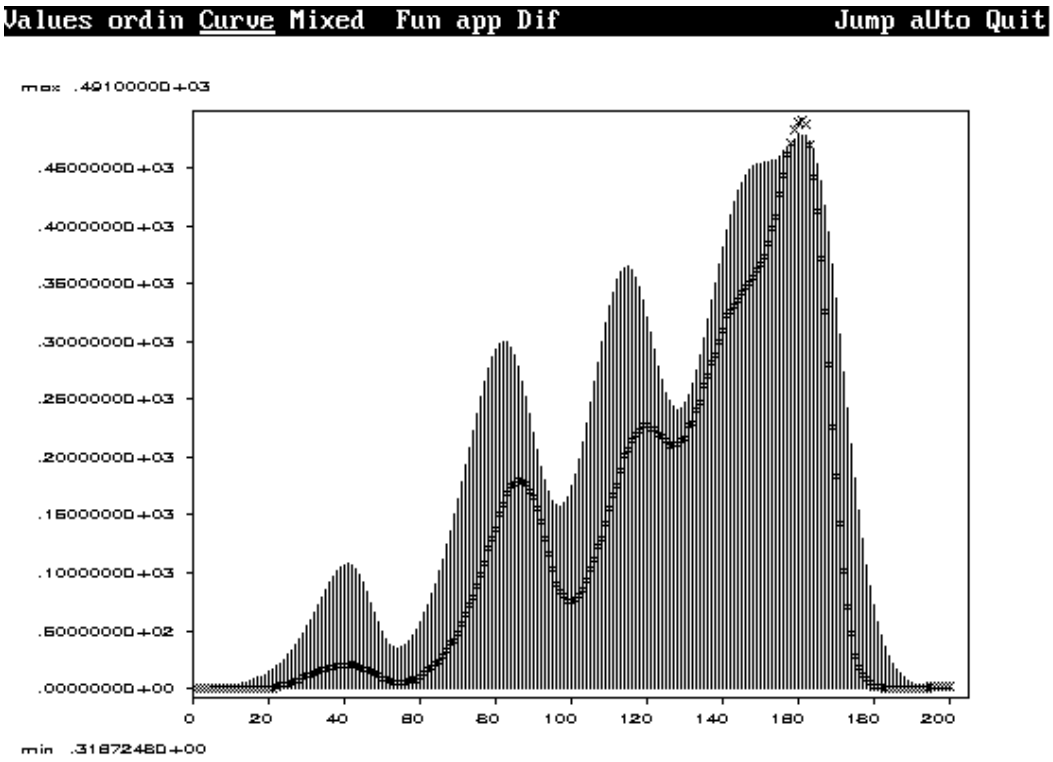
UDDBQ1 - Quadratic programming subproblem for dense nonsmooth problems - the proximal bundle method.
UDDBQ2 - Quadratic programming subproblem for dense nonsmooth problems - the bundle Newton method.
UDDBQ3 - Quadratic programming subproblem for dense nonsmooth problems - the bundle variable metric method.
UDDSQ1 - Quadratic programming subproblem for dense nonlinear programming problems.
UDDXL1 - Linear programming subproblem for dense unconstrained minimax problems.
UDDXQ1 - Quadratic programming subproblem for dense unconstrained minimax problems.
UDSSQ1 - Quadratic programming subproblem for sparse nonlinear programming problems.
UDSSQ3 - Quadratic programming subproblem for sparse nonlinear programming problems.
UDSLL1 - Solution of the KKT system for the Miao first infeasible predictor-corrector method for linear programming.
UDSLL2 - Solution of the KKT system for the Miao second infeasible predictor-corrector method for linear programming.
UDSLL3 - Solution of the KKT system for the Mizuno infeasible predictor-corrector method for linear programming.

D Demonstration of internal FORTRAN graphic output

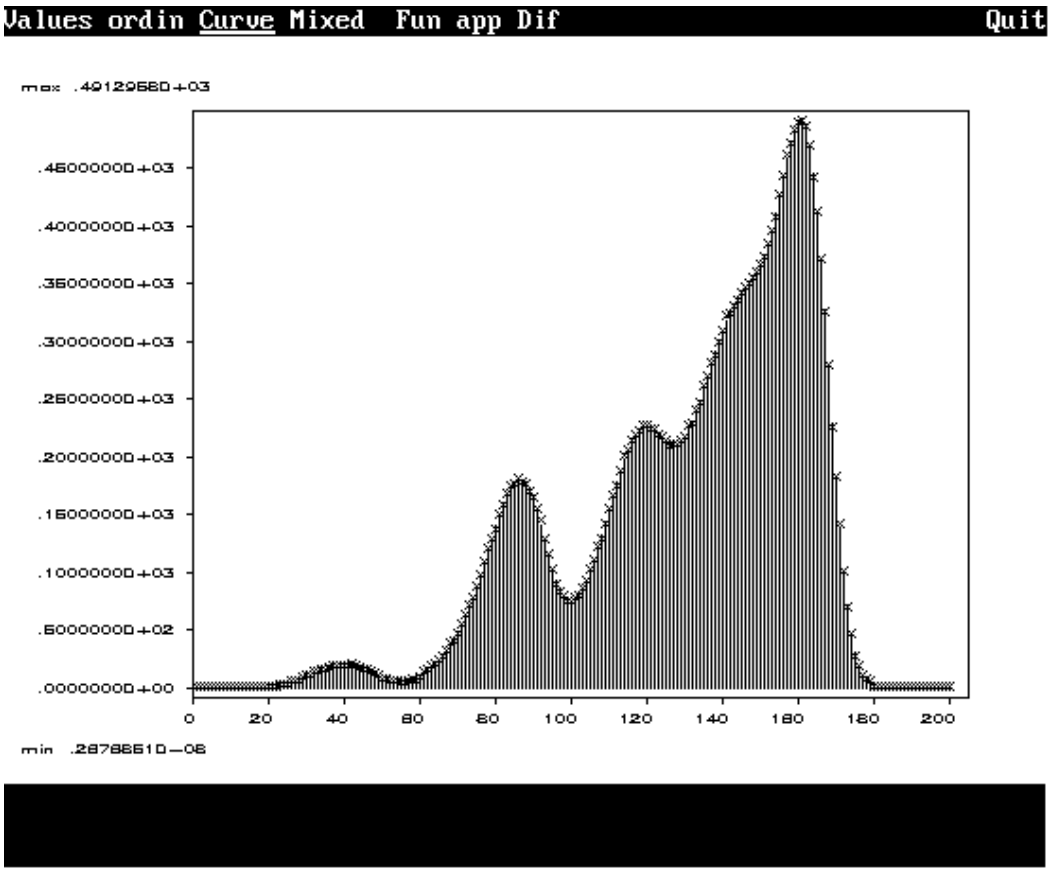
D.1 Nonlinear regression



Nonlinear regression: Iteration 0

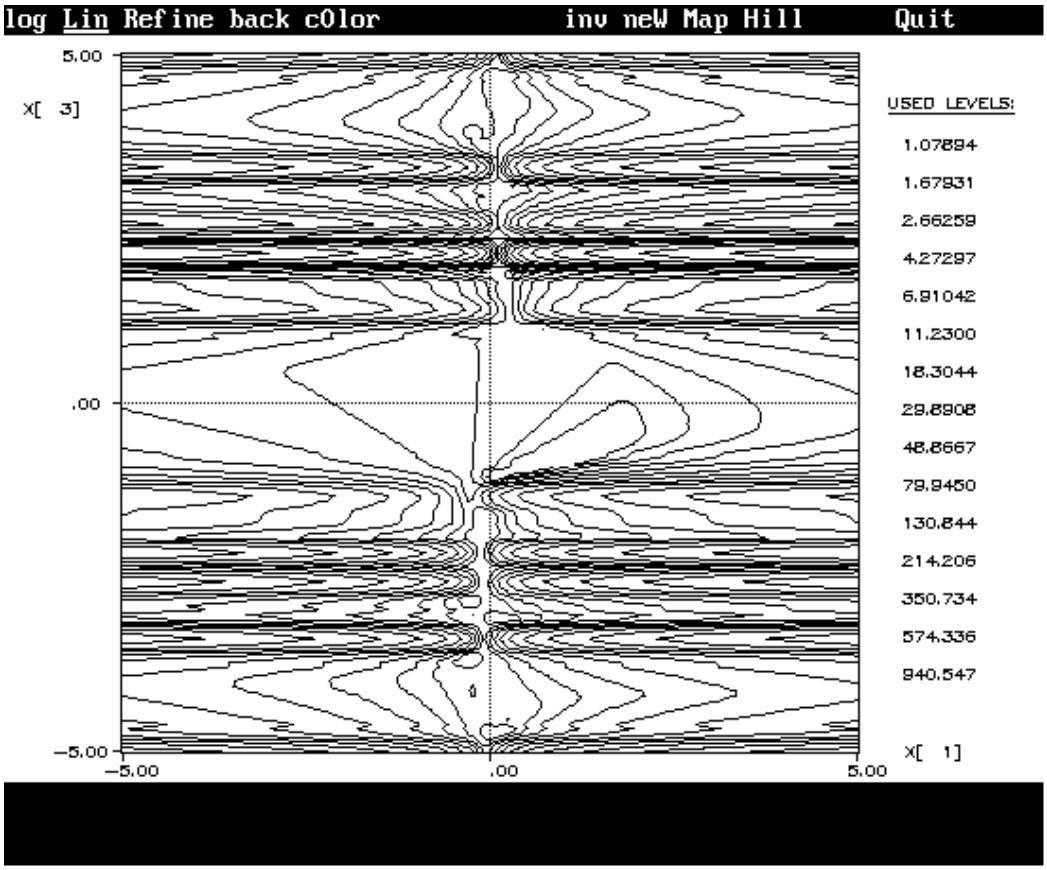


Nonlinear regression: Iteration 6

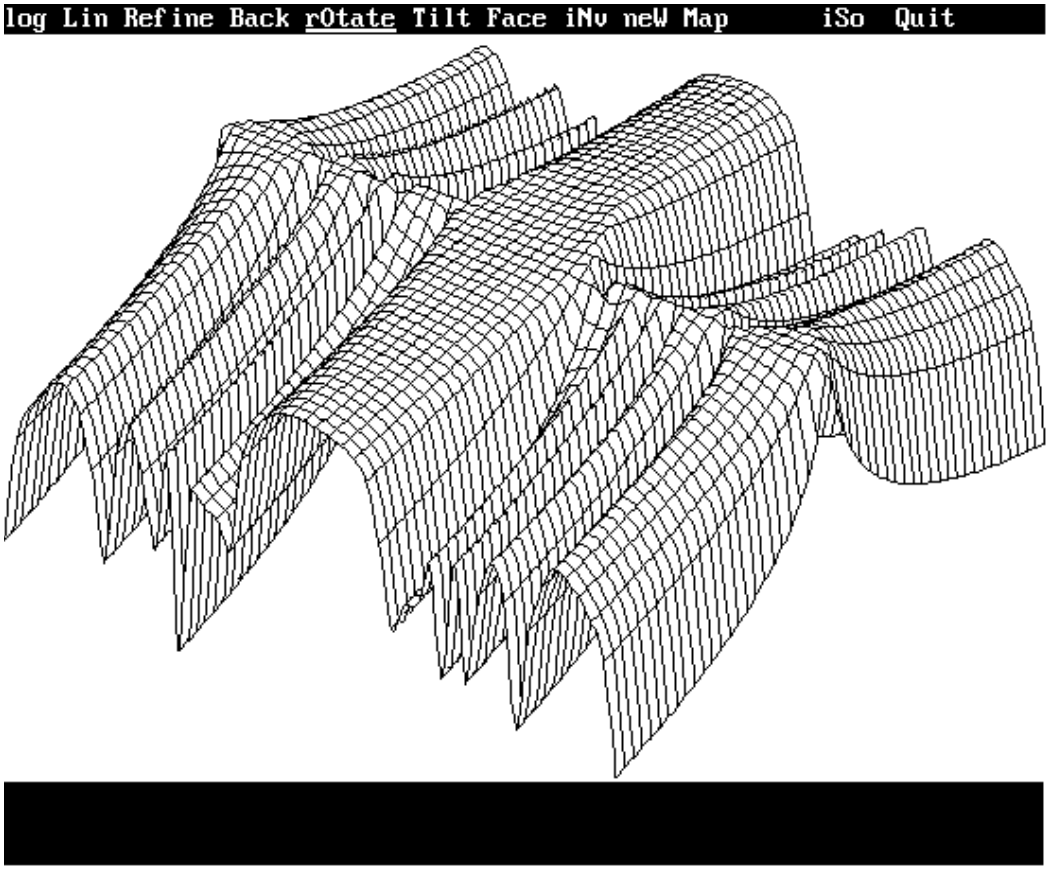


Nonlinear regression: Final solution

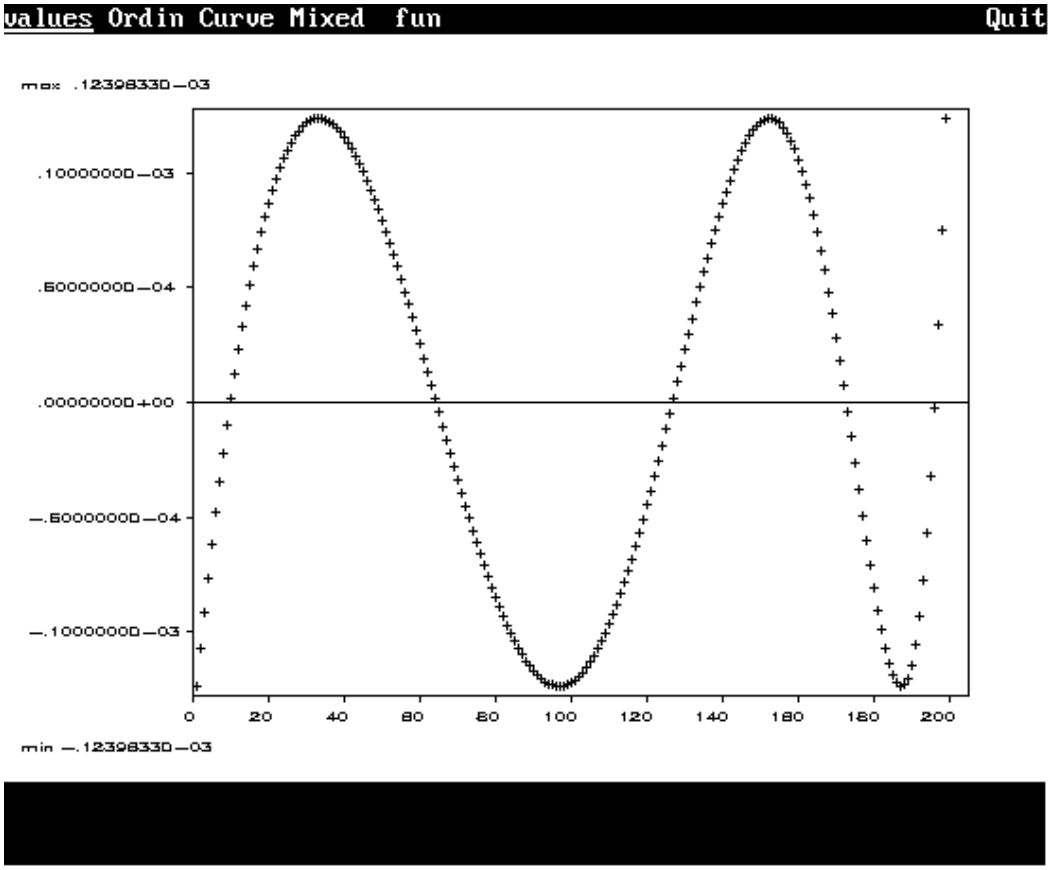
D.2 Nonlinear minimax optimization



Nonlinear minimax optimization: Isolines

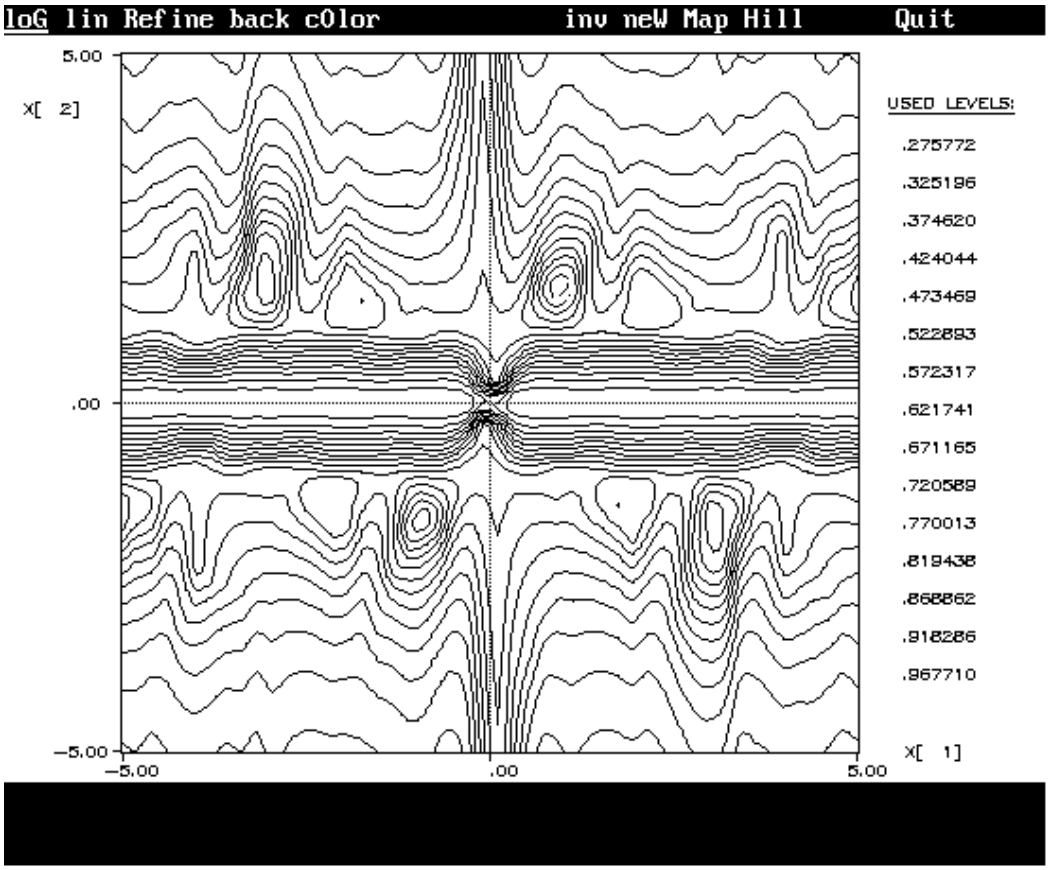


Nonlinear minimax optimization: Surface

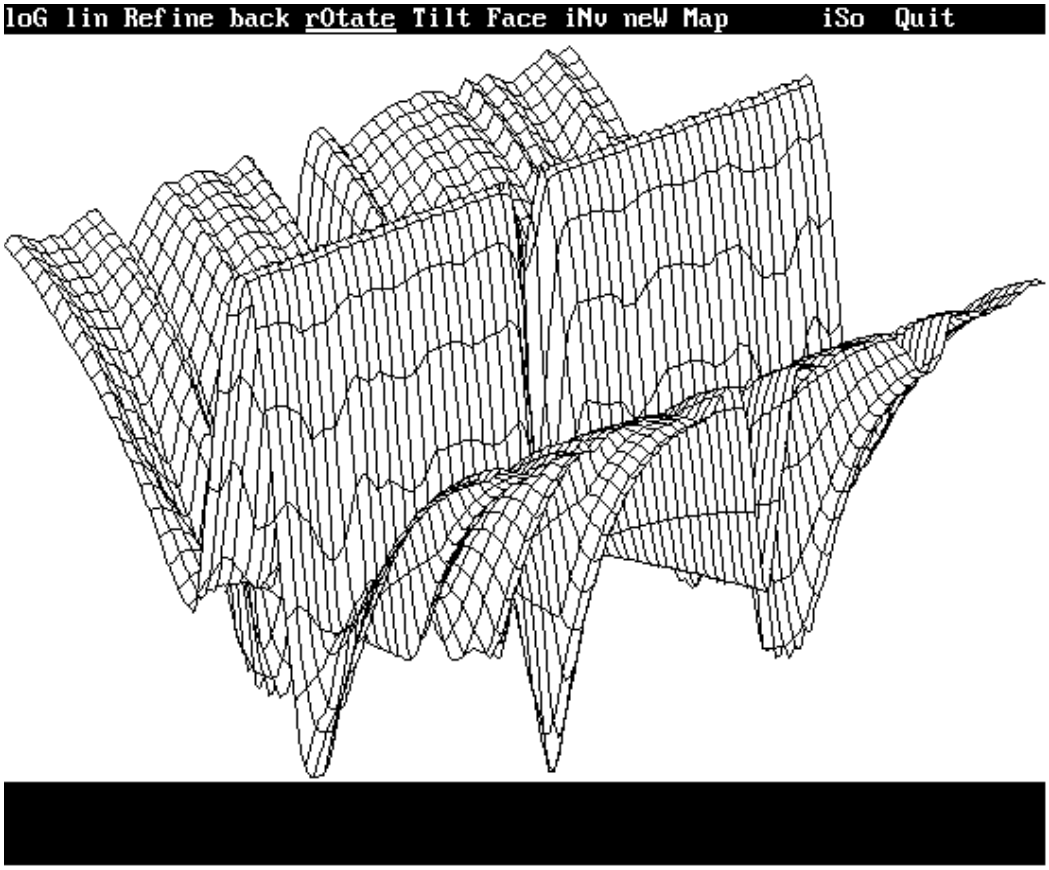


Nonlinear minimax optimization: Graph

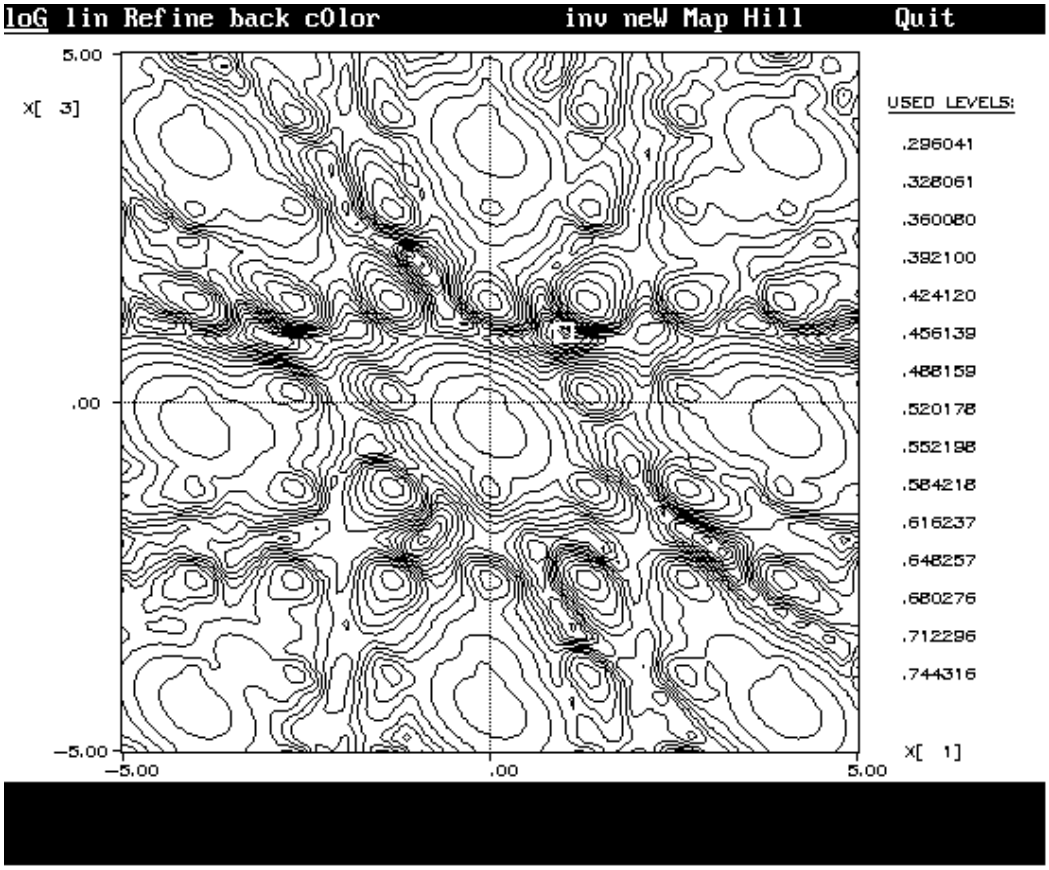
D.3 Transformer network design



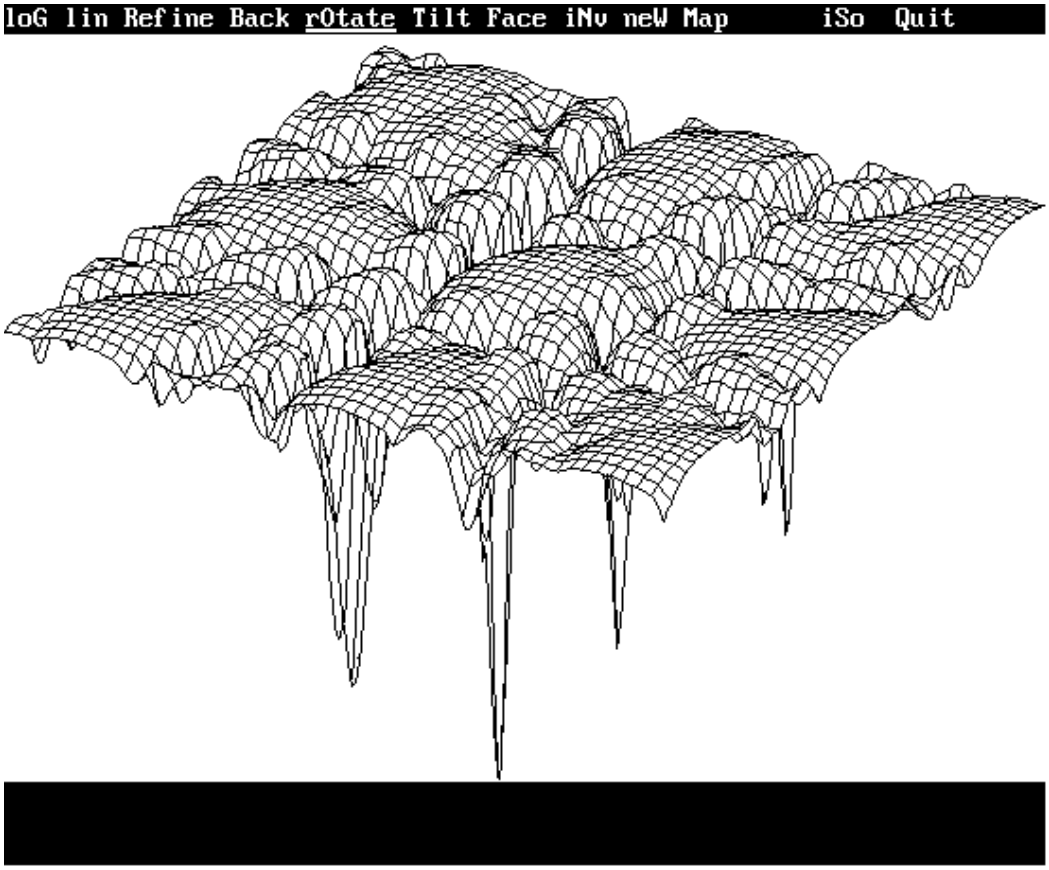
Transformer network design: Isolines



Transformer network desogn : Surface

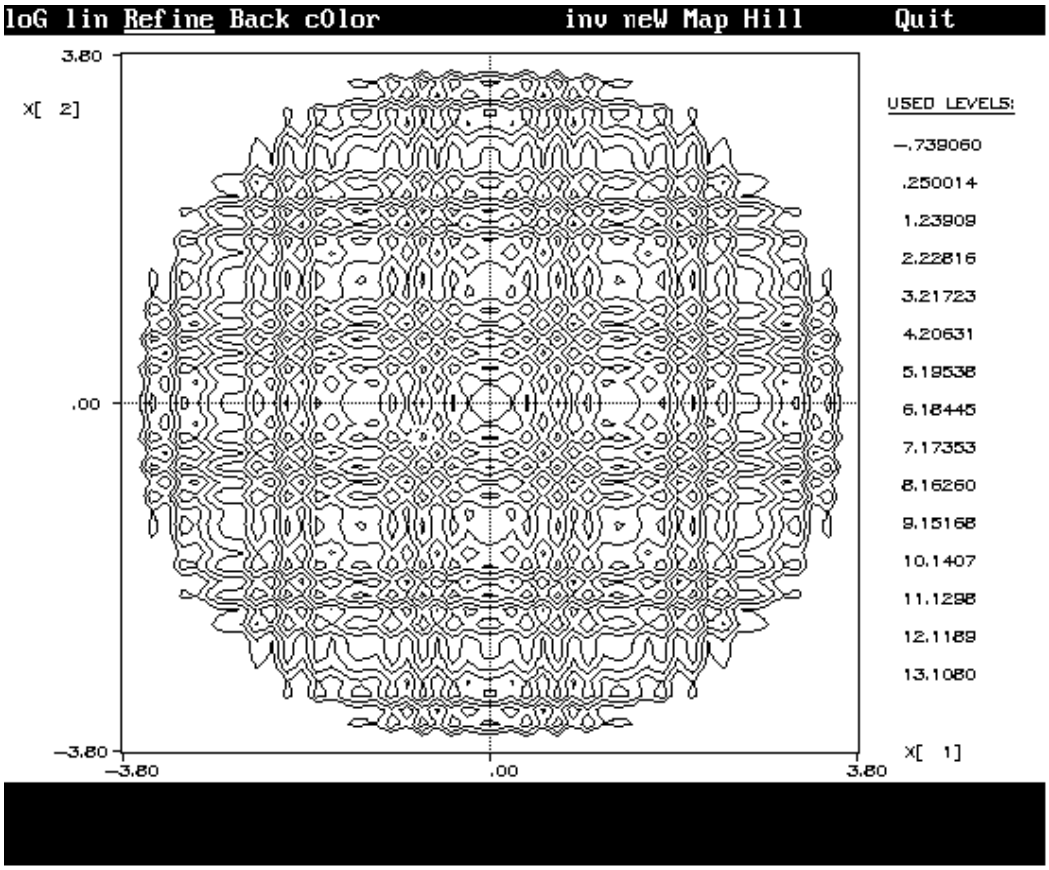


Transformer network design: Isolines



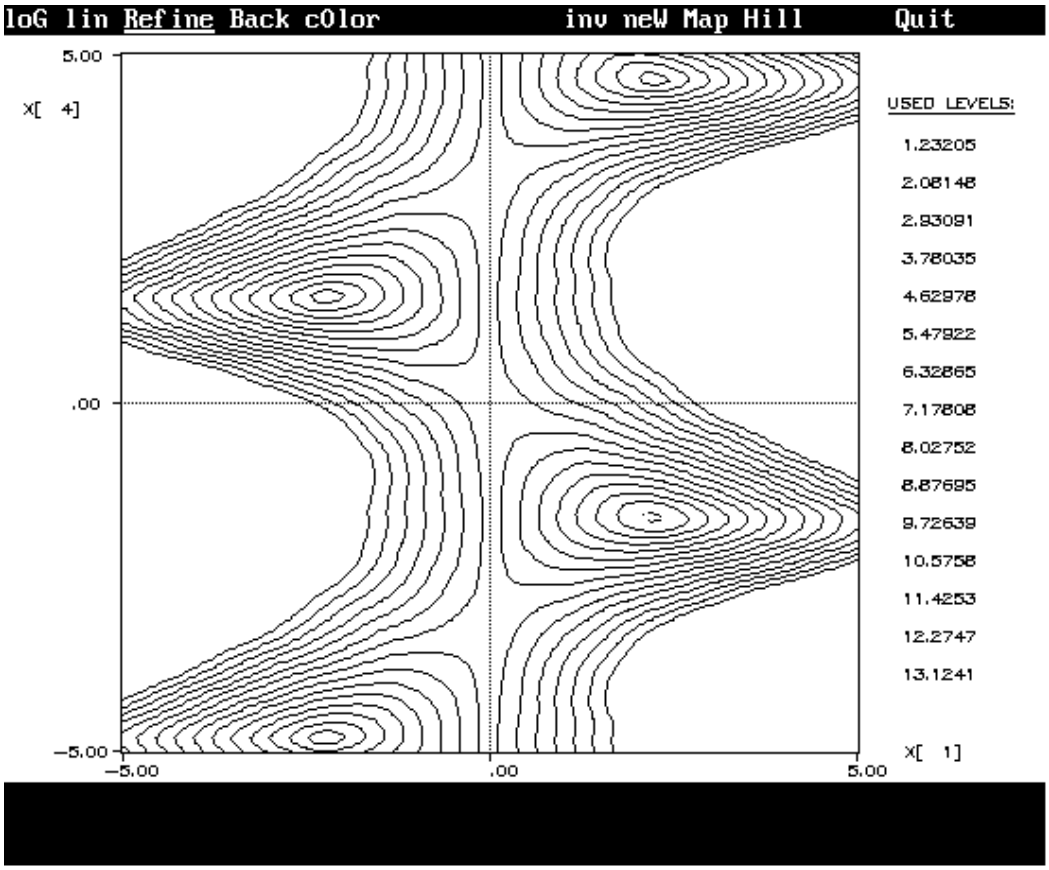
Transformer network design: Surface

D.4 Global optimization



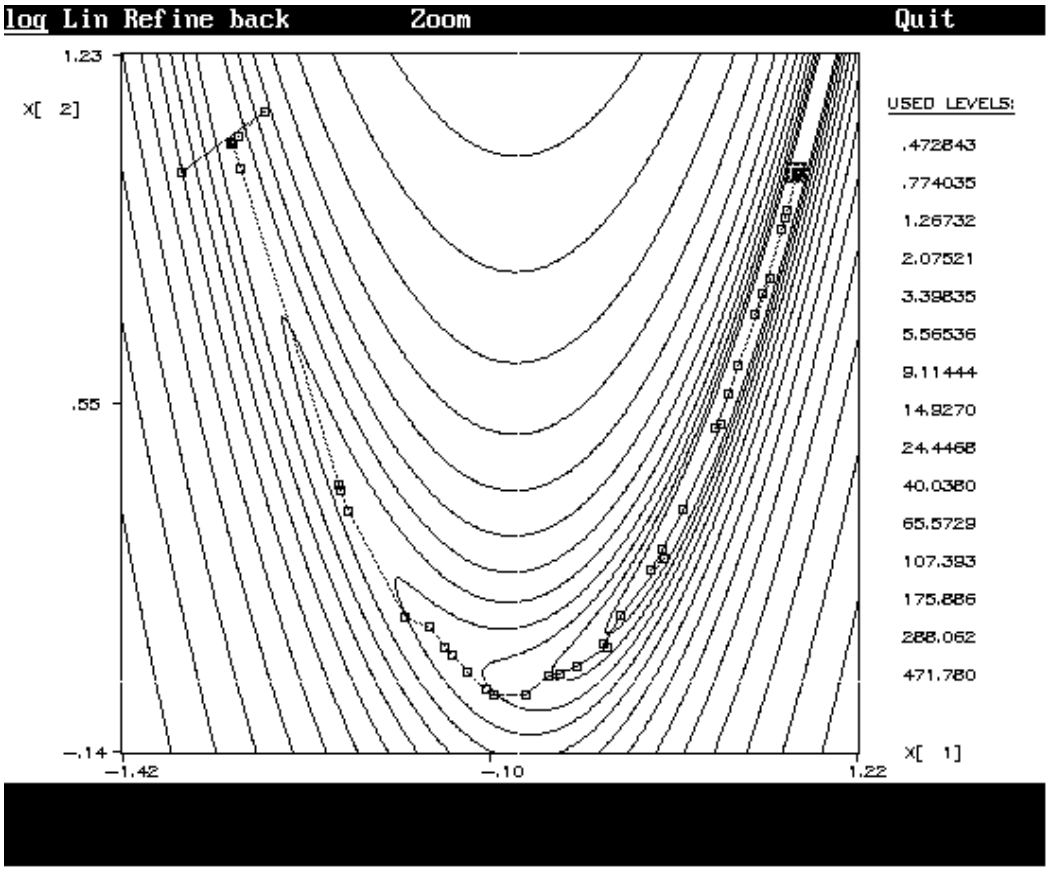
Global optimization: Isolines

D.5 Nonsmooth optimization



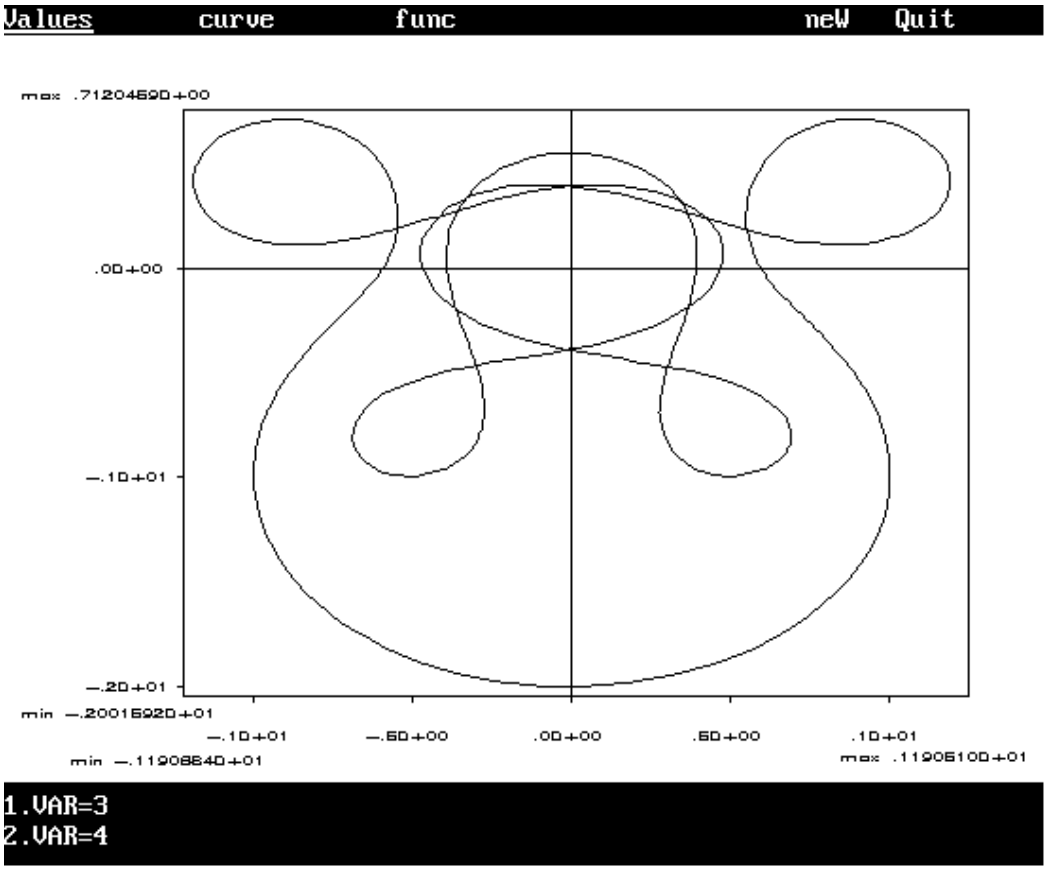
Nonsmooth optimization: Isolines

D.6 Rosenbrock function



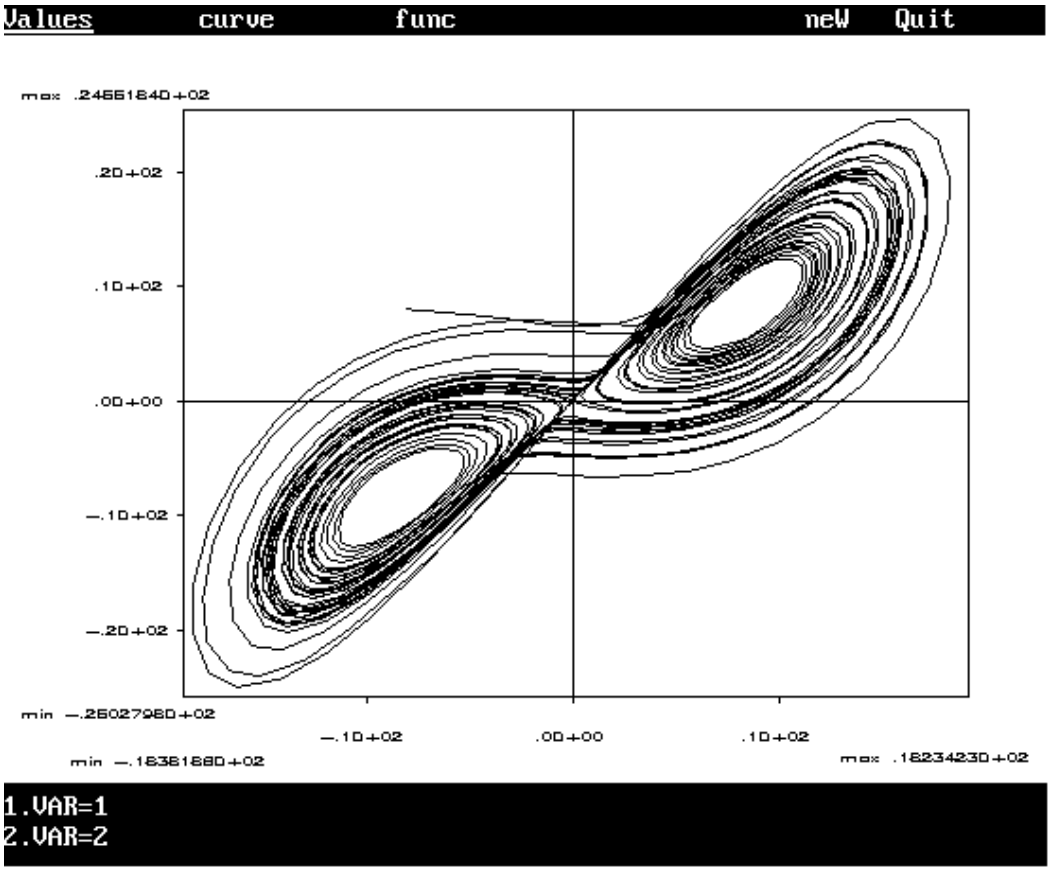
Rosenbrock function: Path of iterations

D.7 Ordinary differential equations

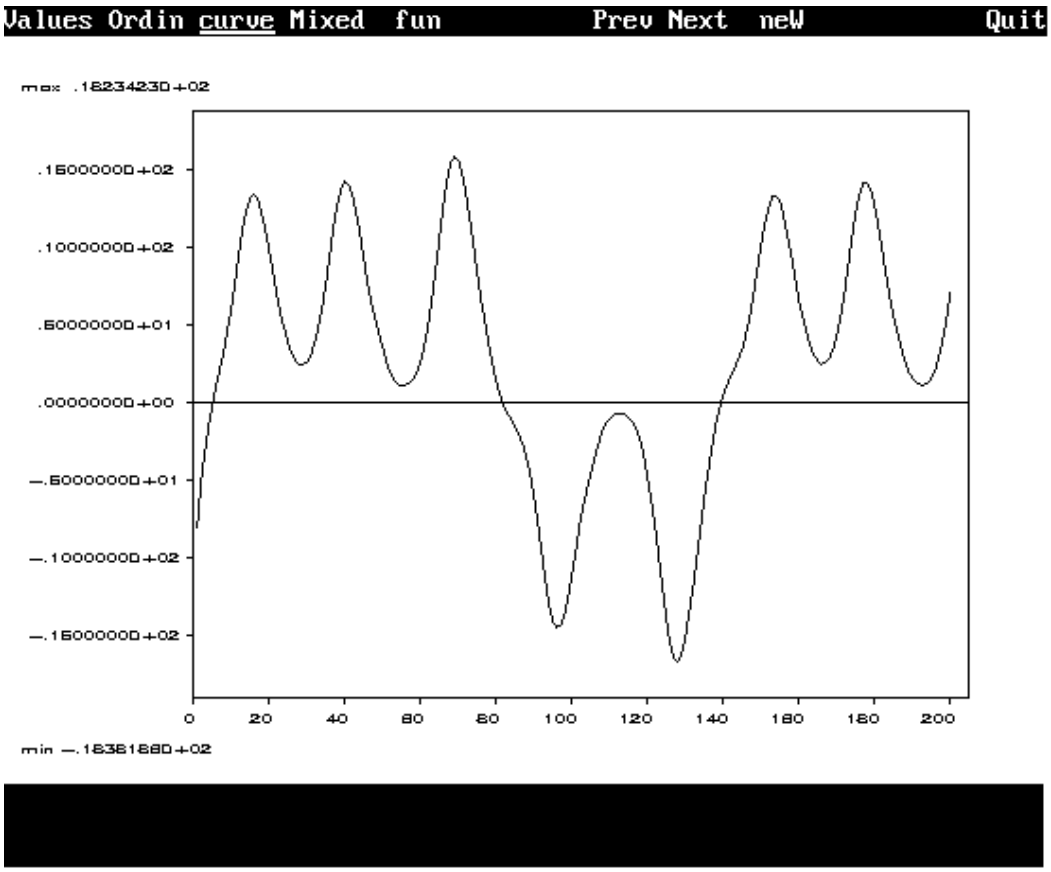


Ordinary differential equations: Orbit

D.8 The Lorenz attractor



The Lorenz attractor: Orbit

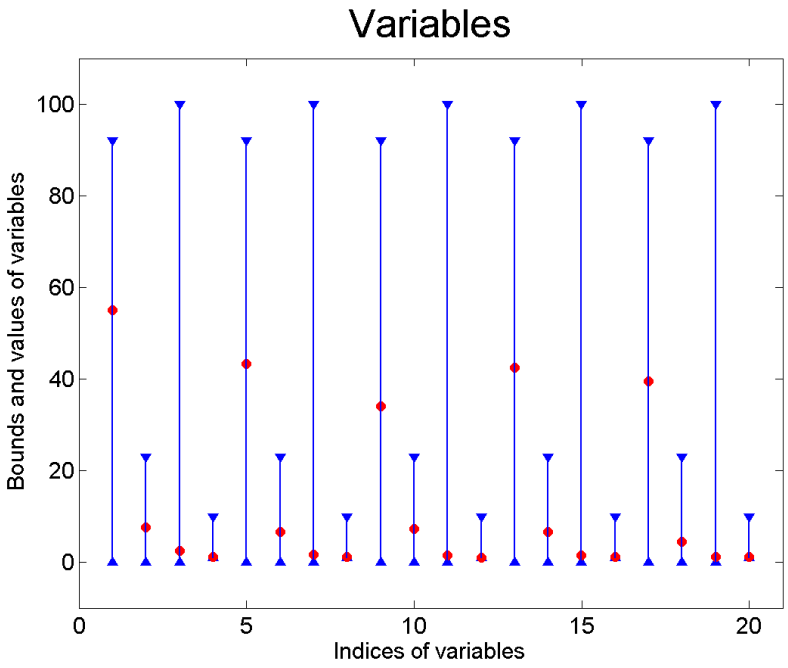


The Lorenz attractor: Trajectory

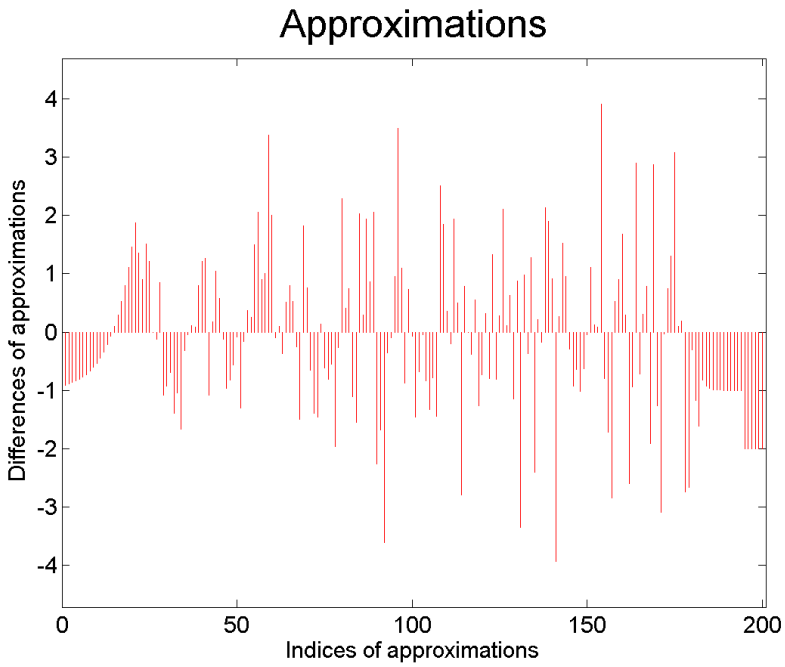
E Demonstration of external MATLAB graphic output

E.1 Nonlinear regression

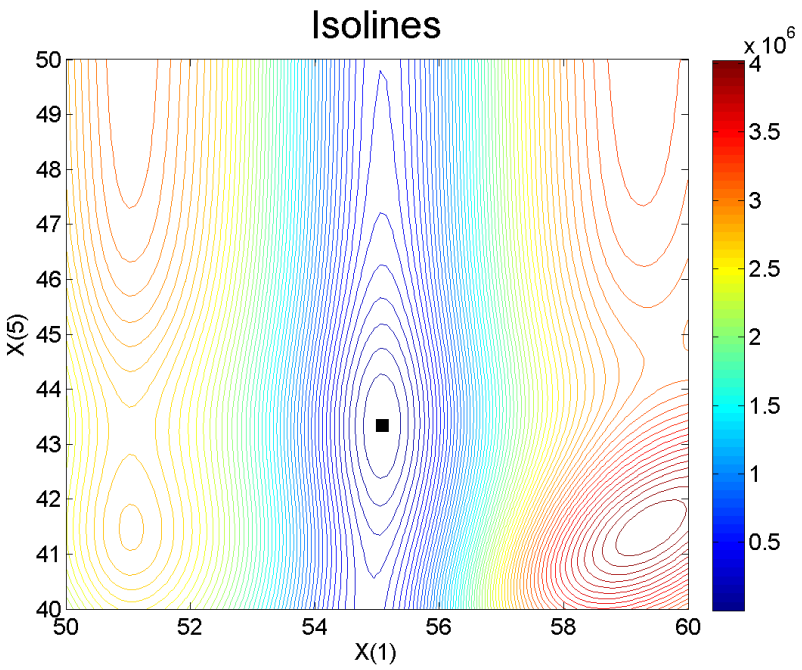
- Bounds and values of variables (final solution)



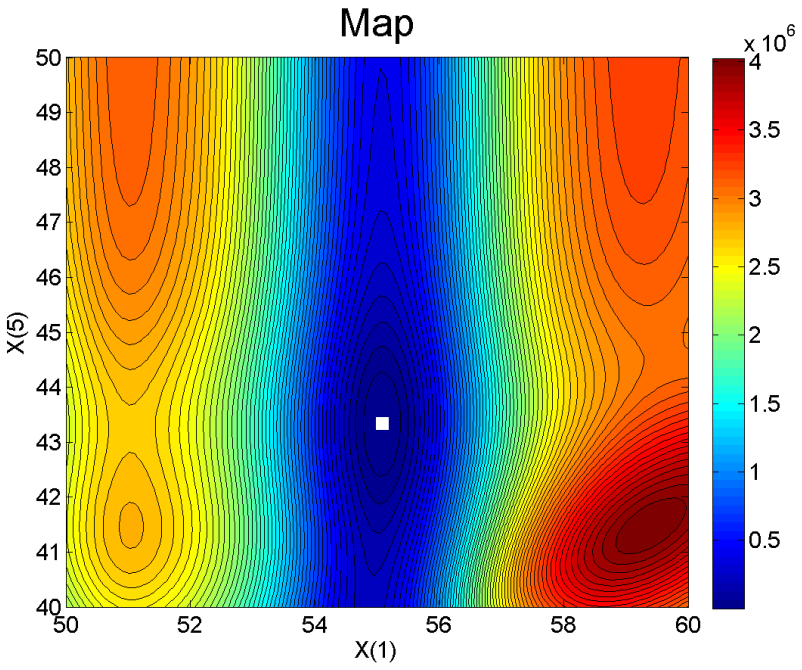
- Differences between AF and AM (final solution)



- Isolines

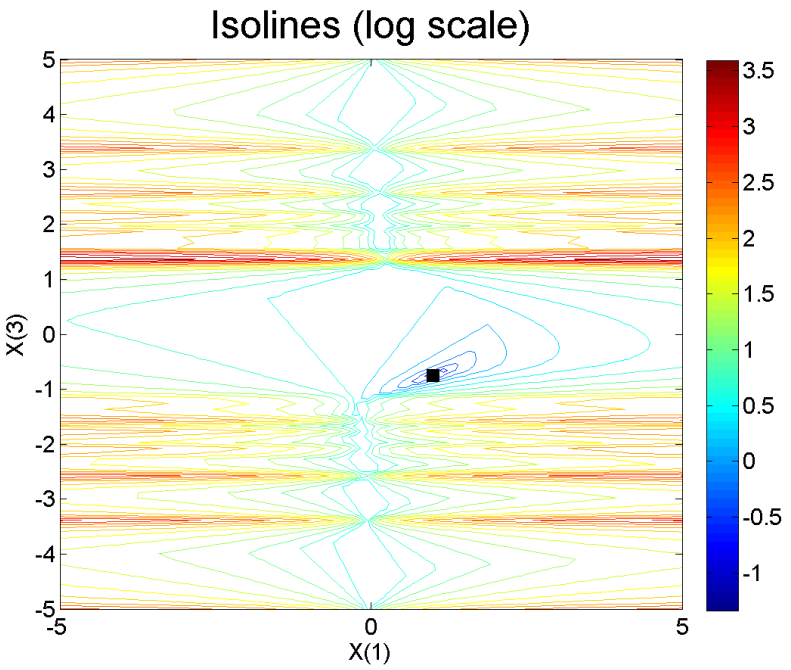


- Map

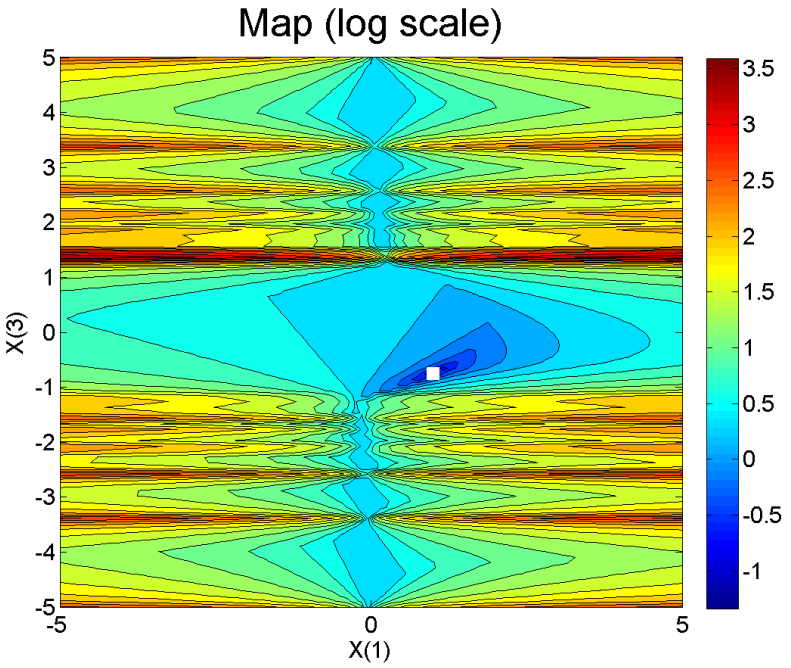


E.2 Nonlinear minimax optimization

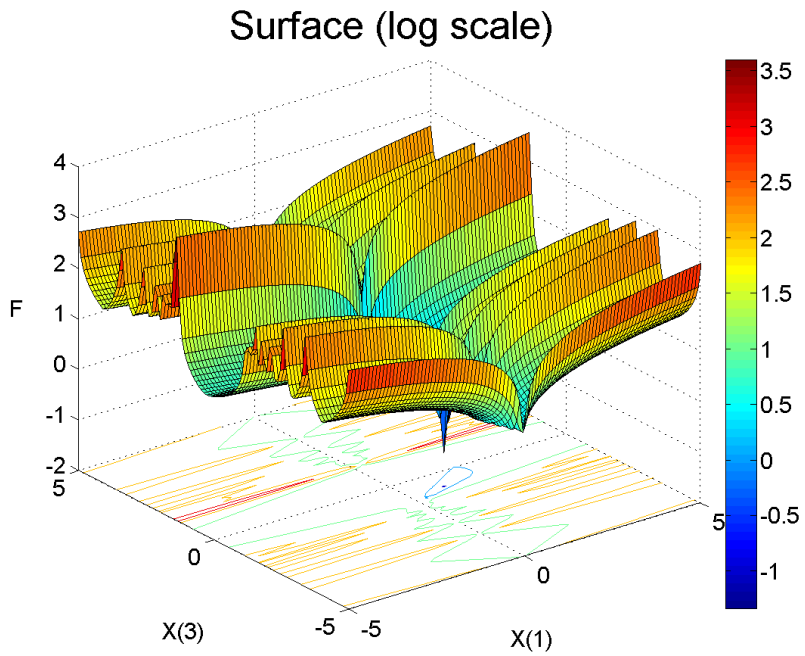
- Isolines



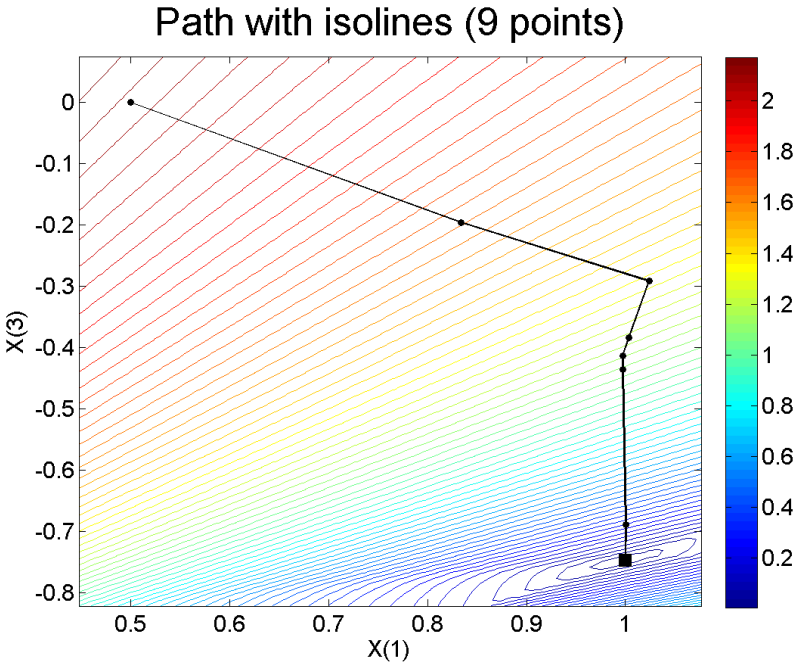
- Map



- Surface

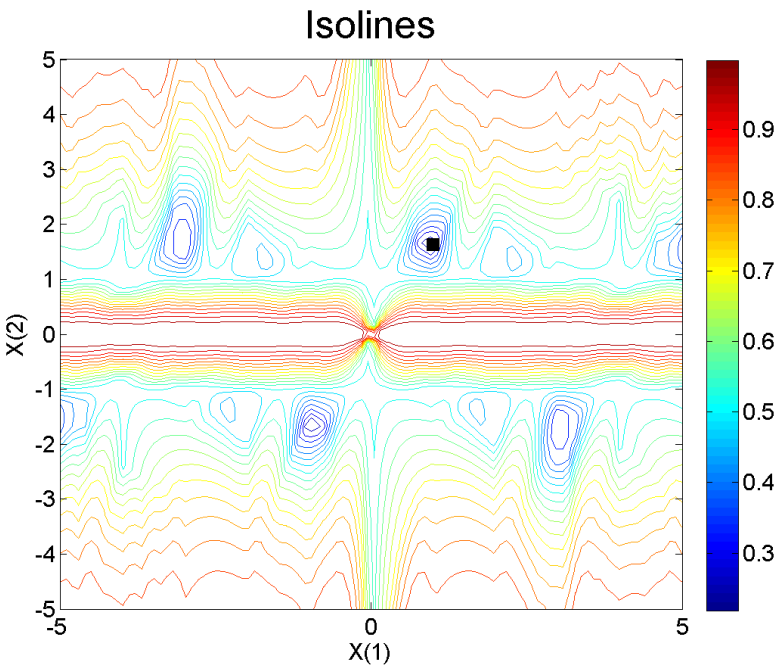


- Path with isolines

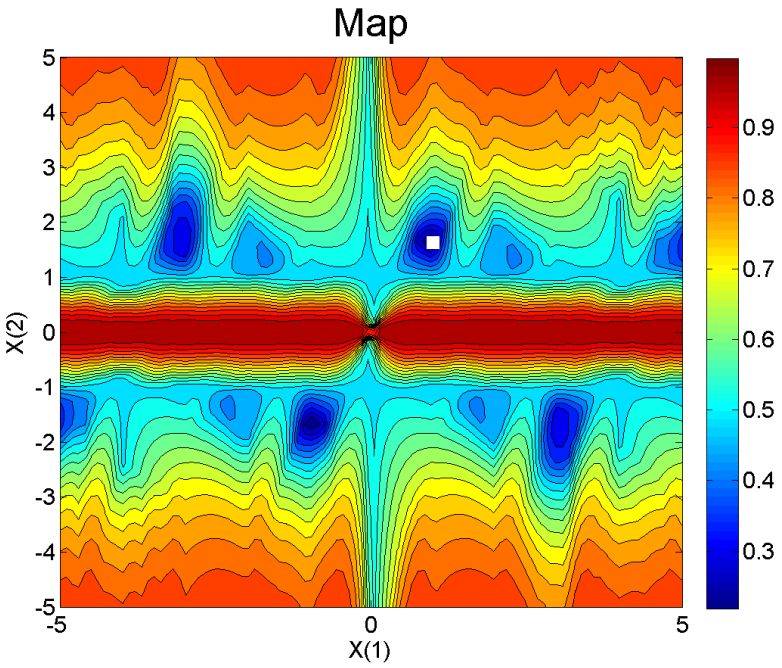


E.3 Transformer network design

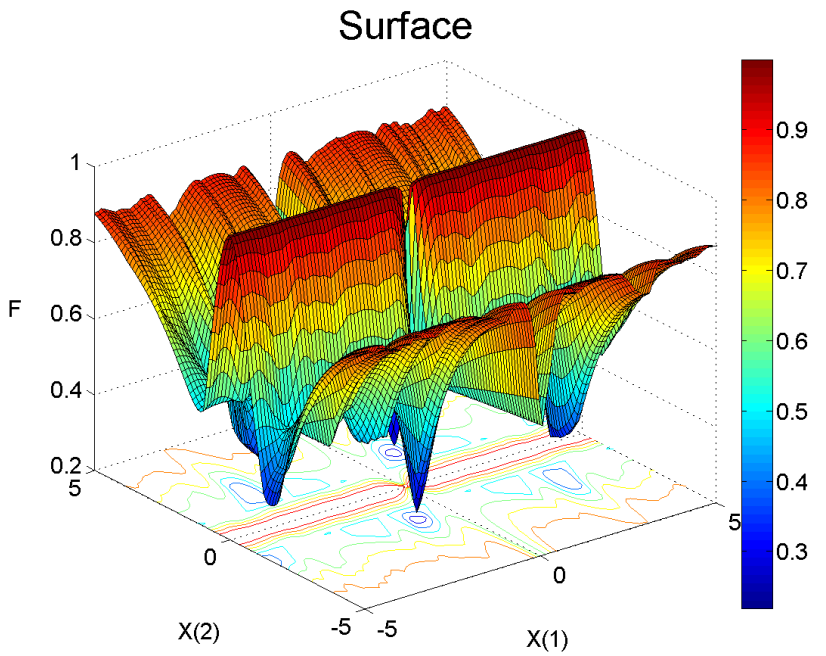
- Isolines



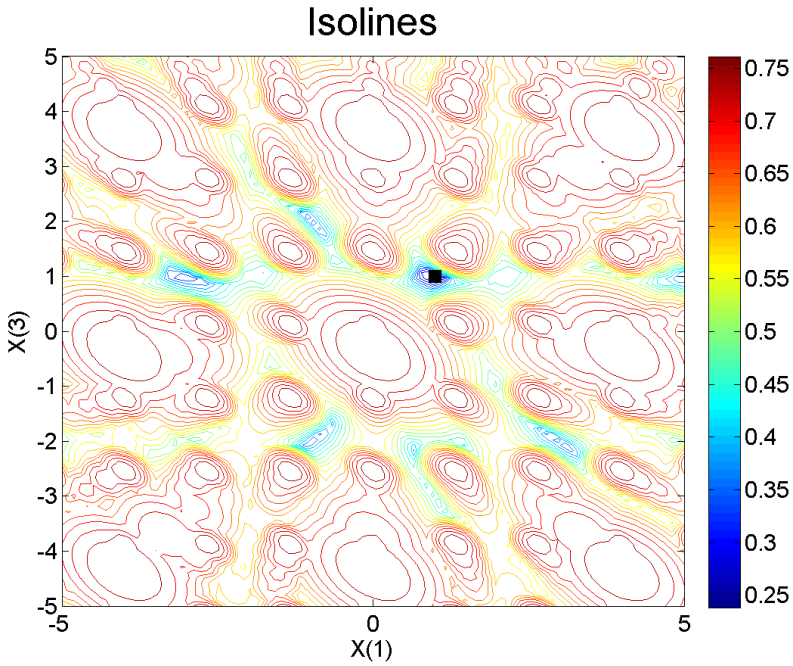
- Map



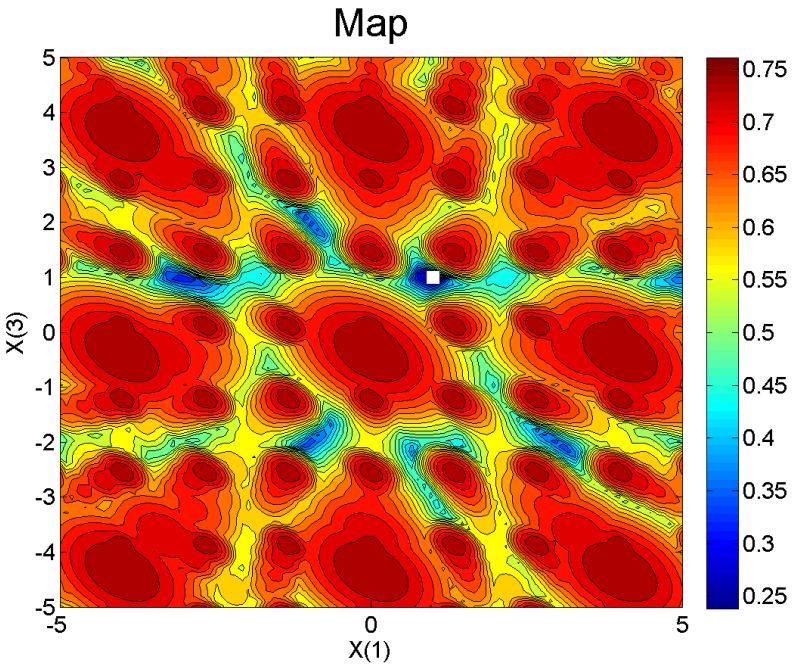
- Surface



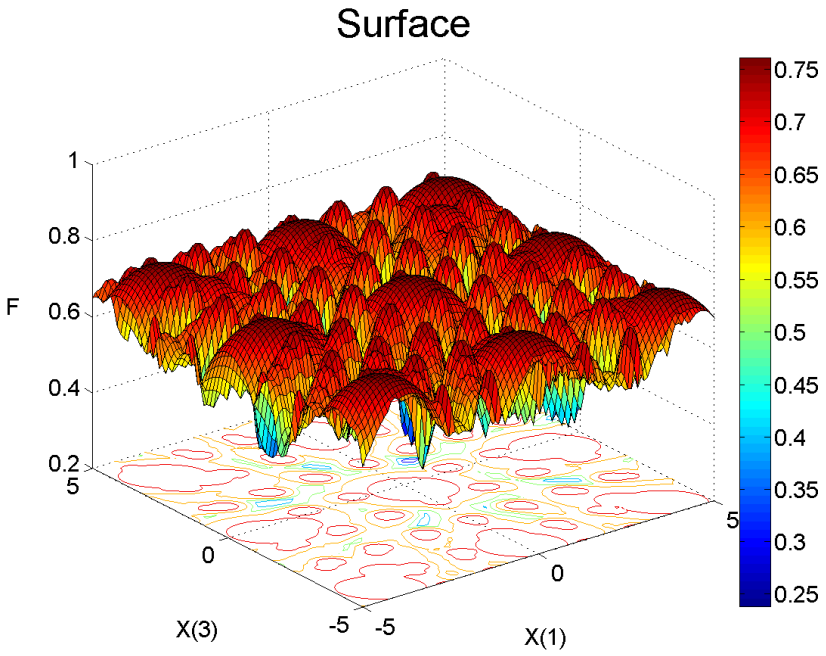
- Isolines



- Map

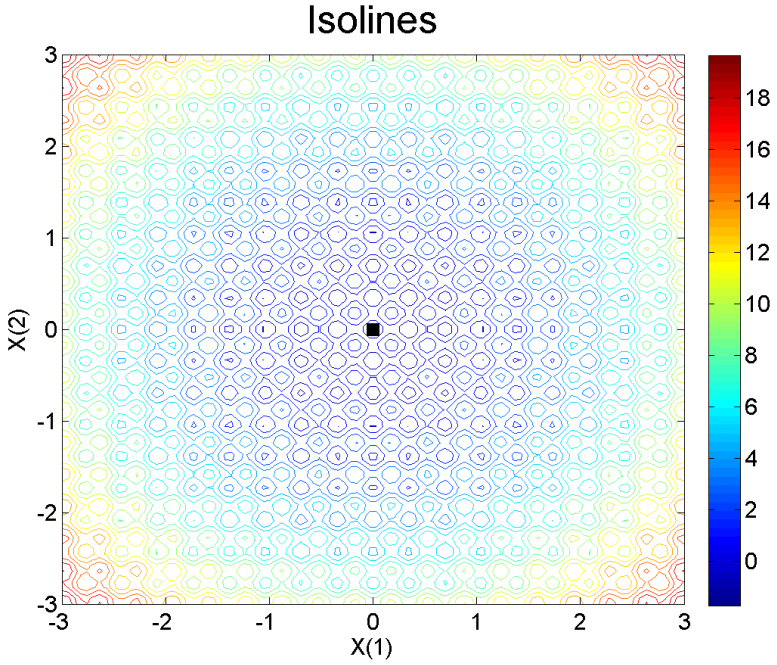

Map
X(3)
X(1)
5
4
3
2
1
0
-1
-2
-3
-4
-5
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25

- Surface

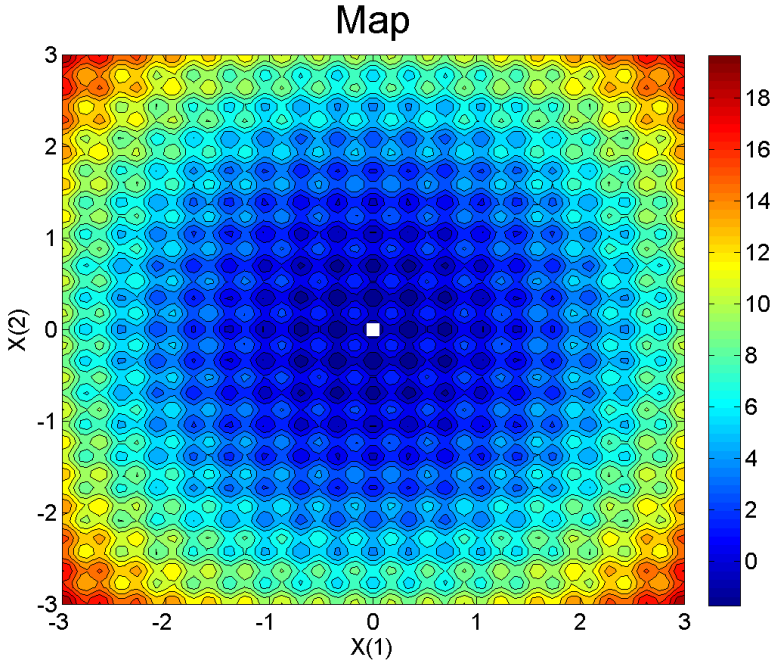

Surface
F
X(3)
X(1)
1
0.8
0.6
0.4
0.2
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.3
0.25

E.4 Global optimization

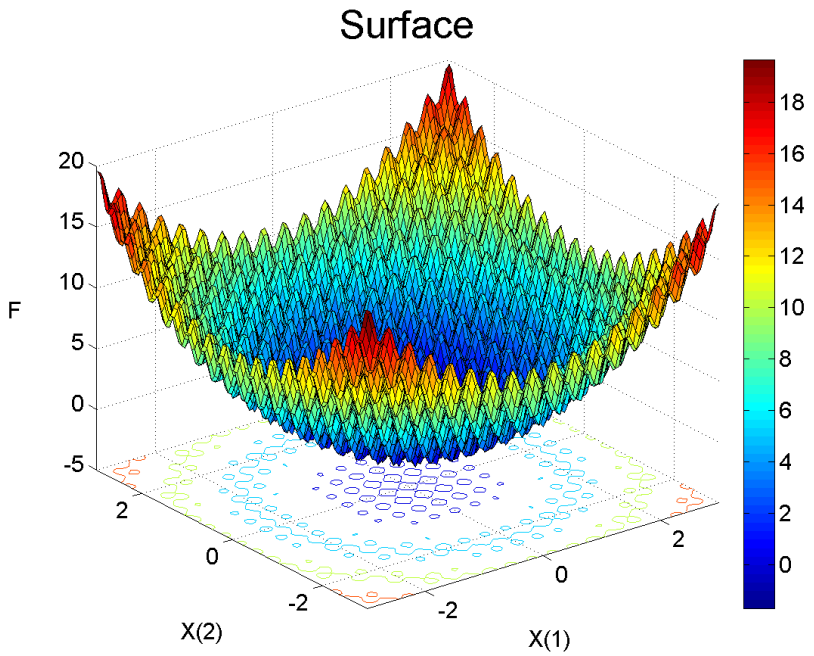
- Isolines



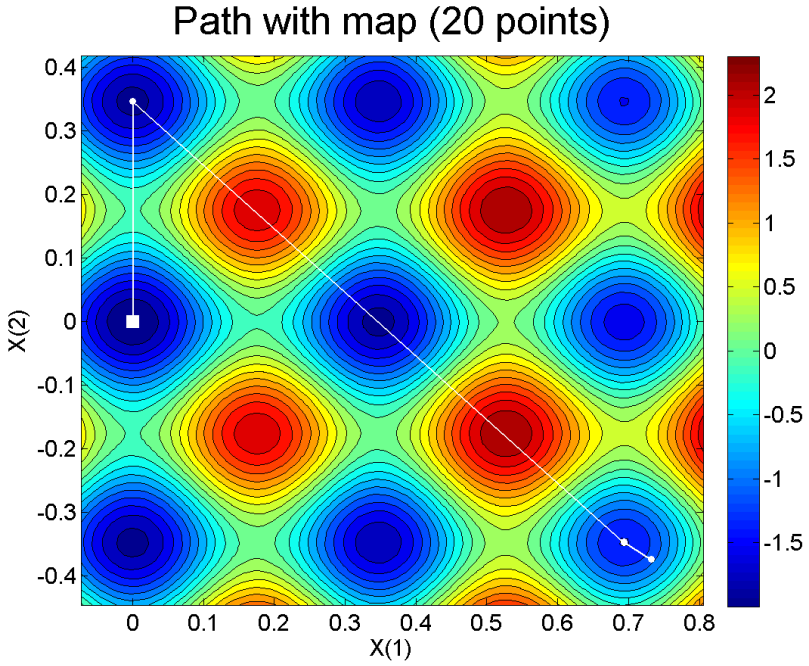
- Map



- Surface

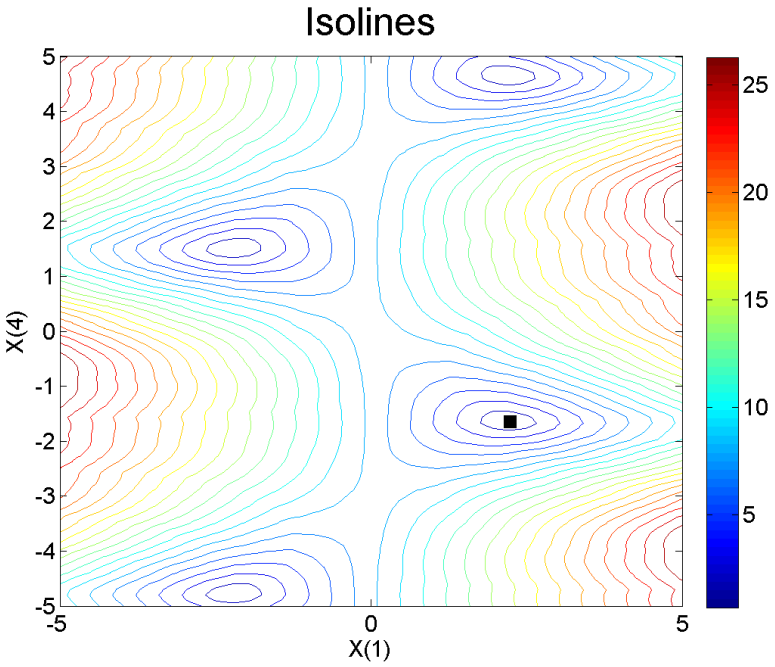


- Path with map

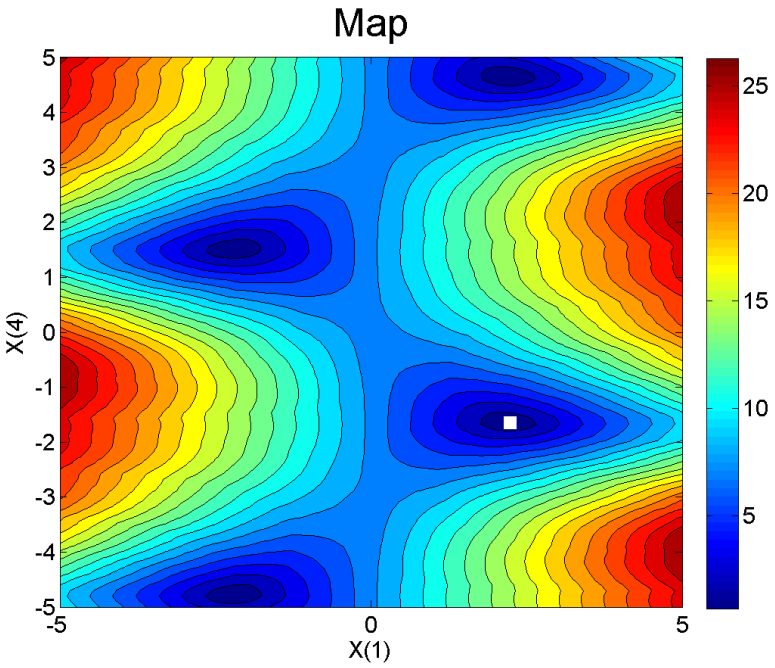


E.5 Nonsmooth optimization

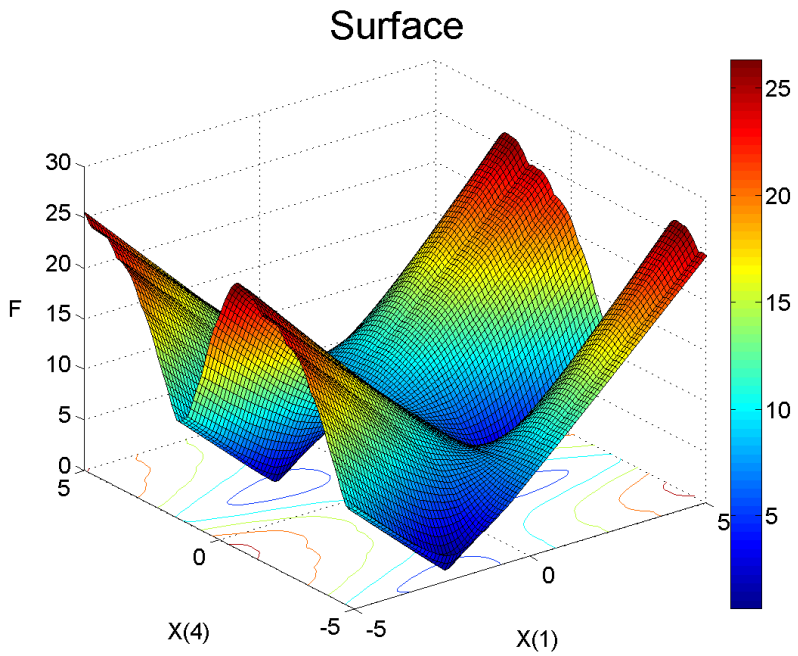
- Isolines



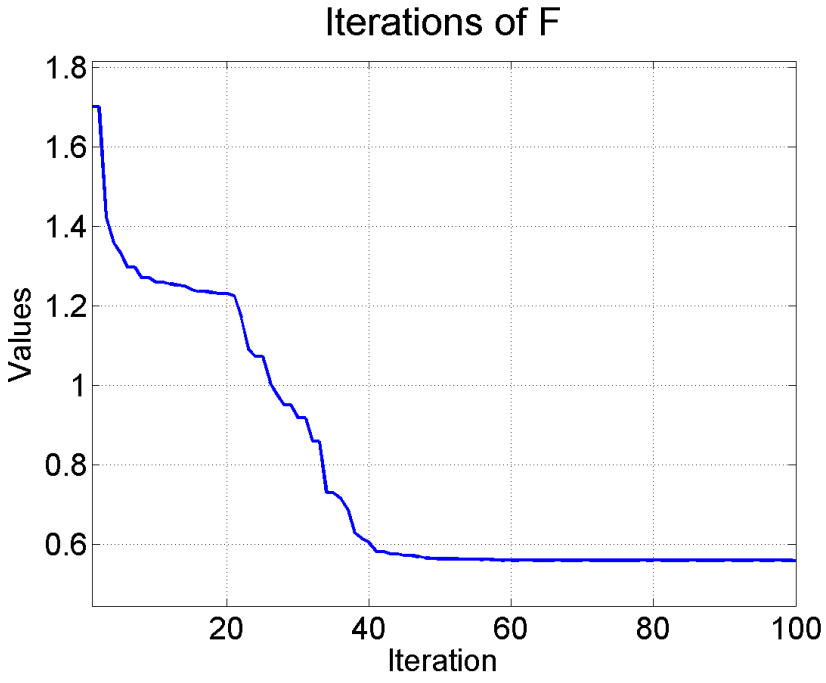
- Map



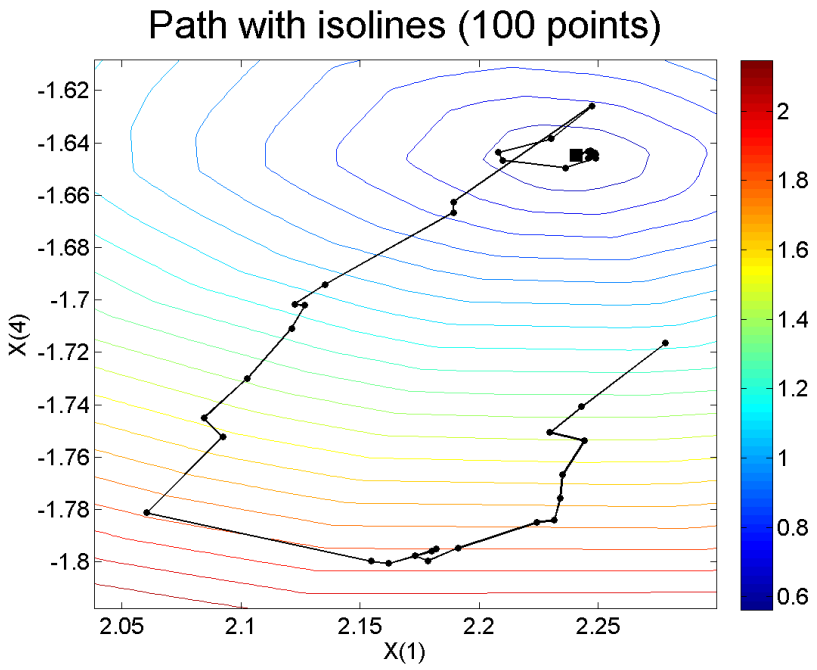
- Surface



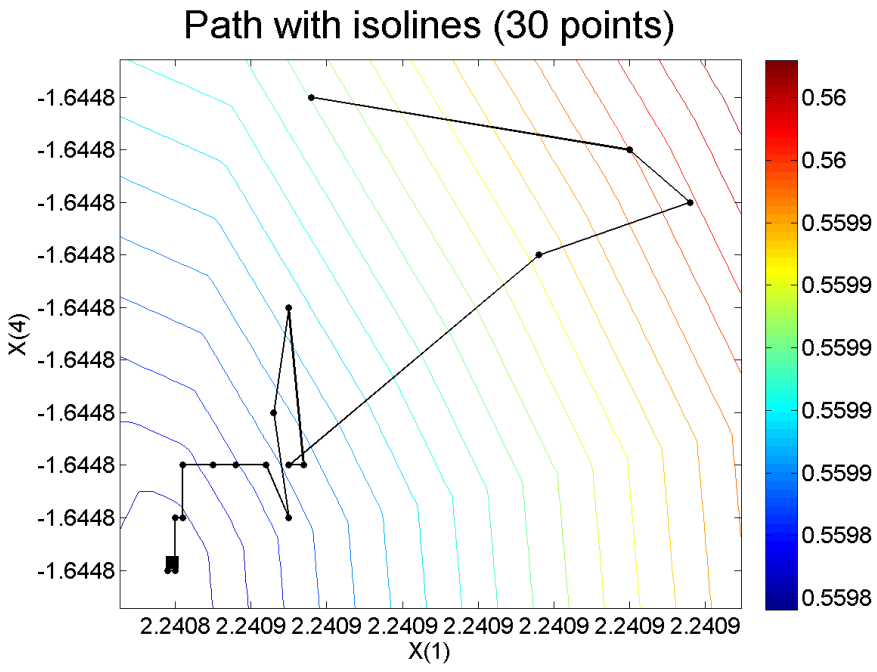
- Iterations of F



- Path with isolines (100 points)

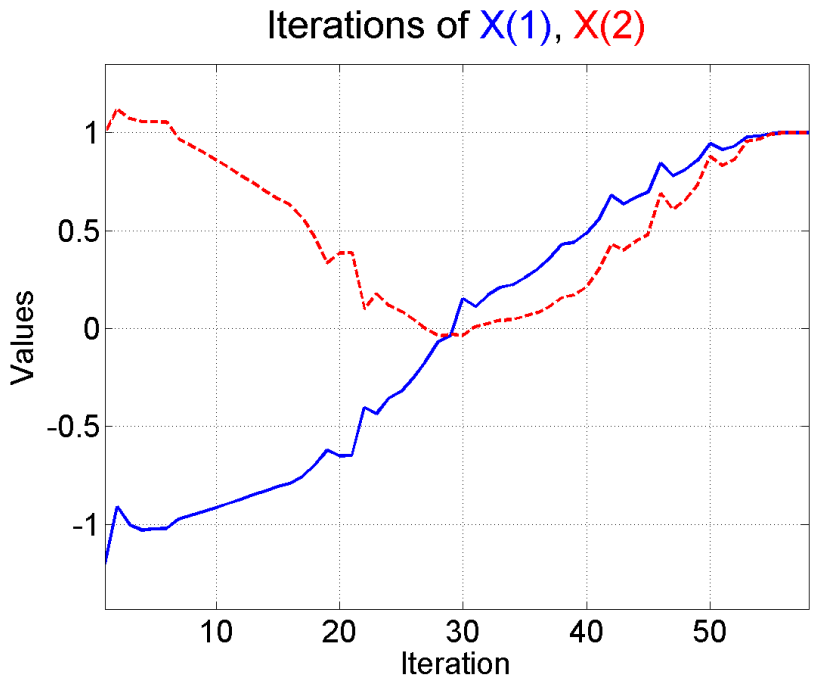


- Path with isolines (30 points)

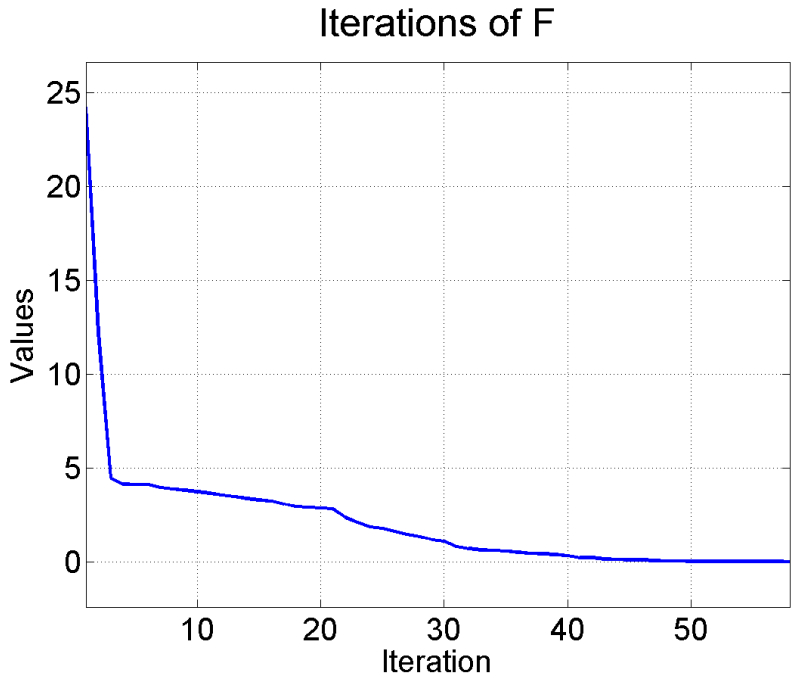


E.6 Rosenbrock function

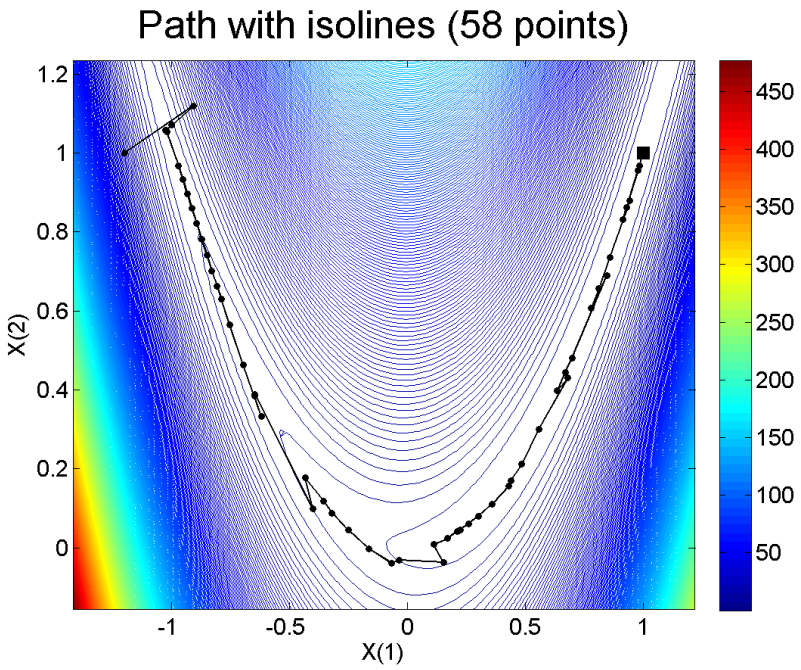
- Iterations of x_1, x_2



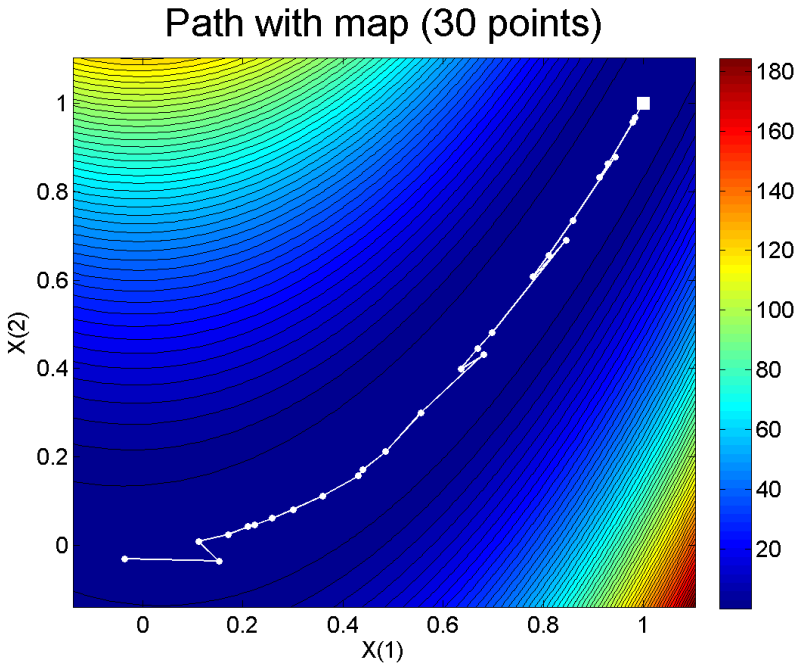
- Iterations of F



- Path with isolines

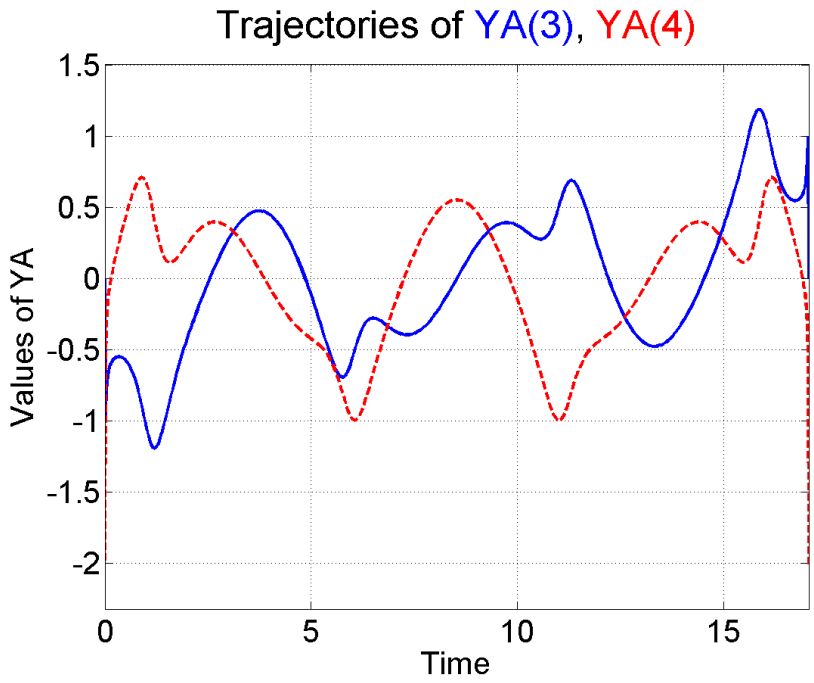


- Path with map

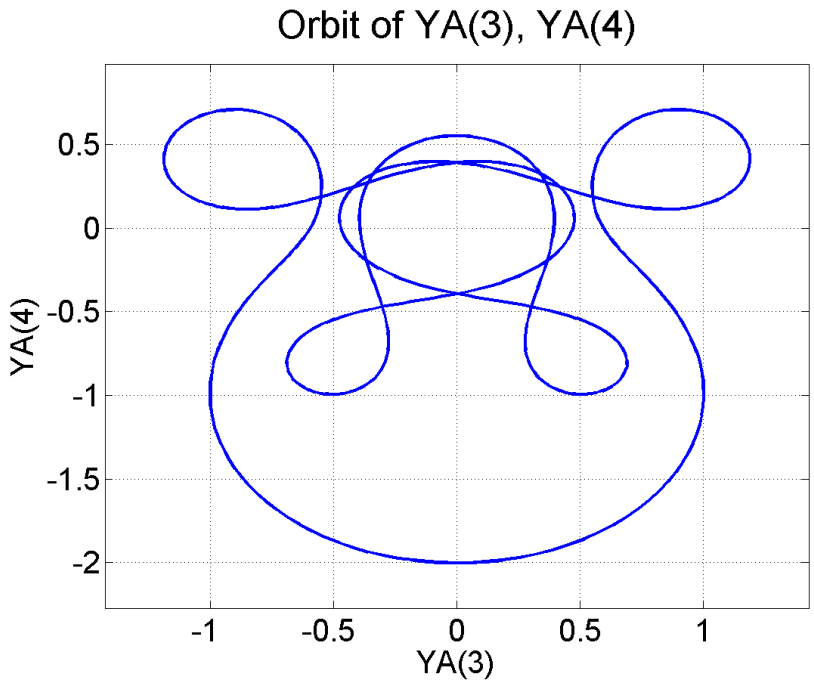


E.7 Ordinary differential equations

- Trajectories of y_3^A, y_4^A

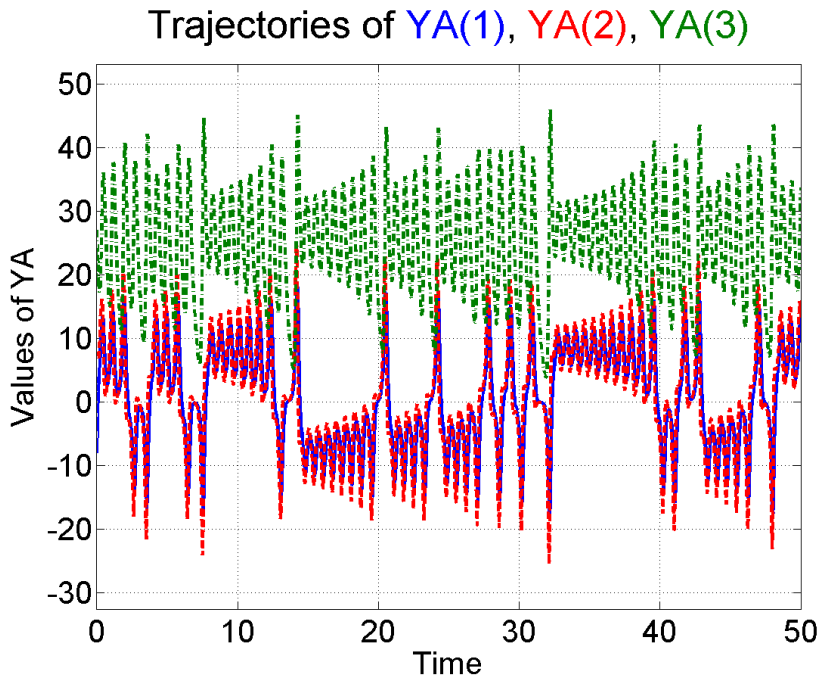


- Orbit of y_3^A, y_4^A

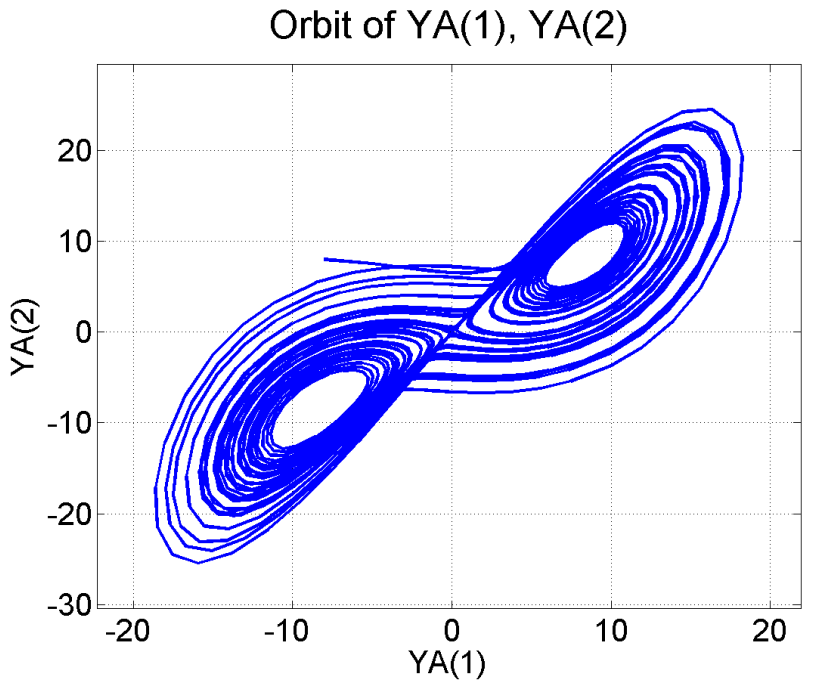


E.8 The Lorenz attractor

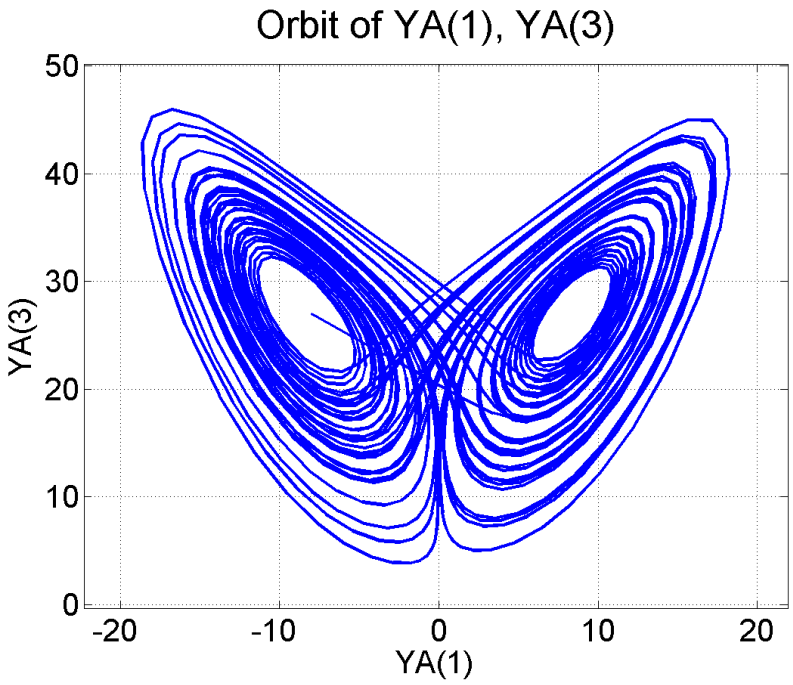
- Trajectories of y_1^A, y_2^A, y_3^A



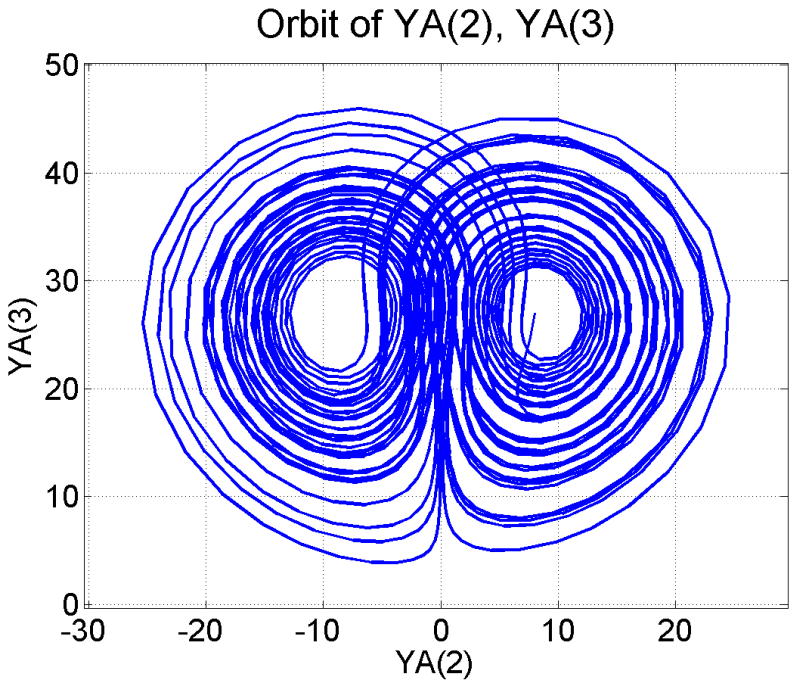
- Orbit of y_1^A, y_2^A



- Orbit of y_1^A, y_3^A



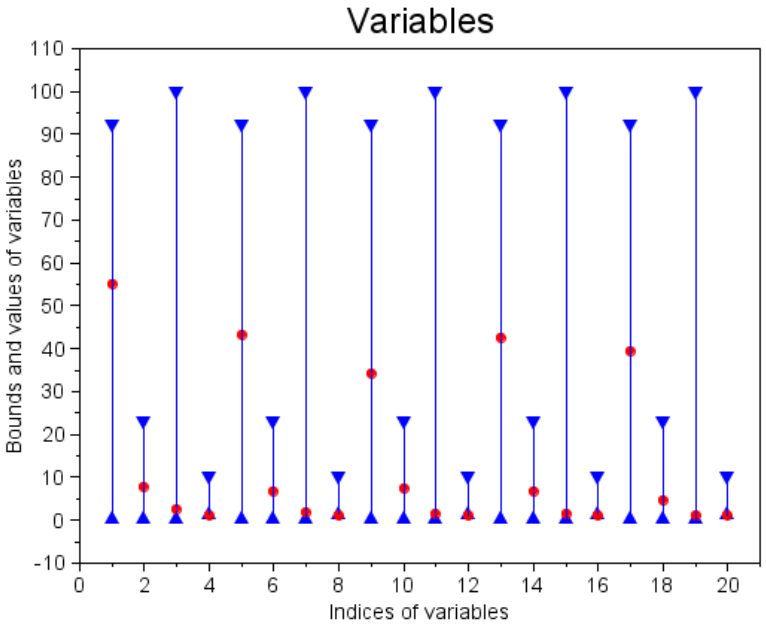
- Orbit of y_2^A, y_3^A



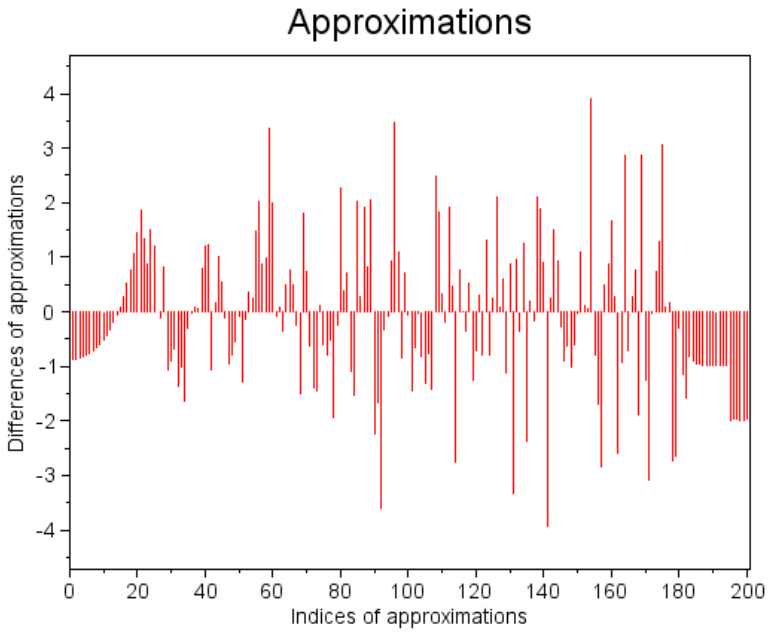
F Demonstration of external SCILAB graphic output

F.1 Nonlinear regression

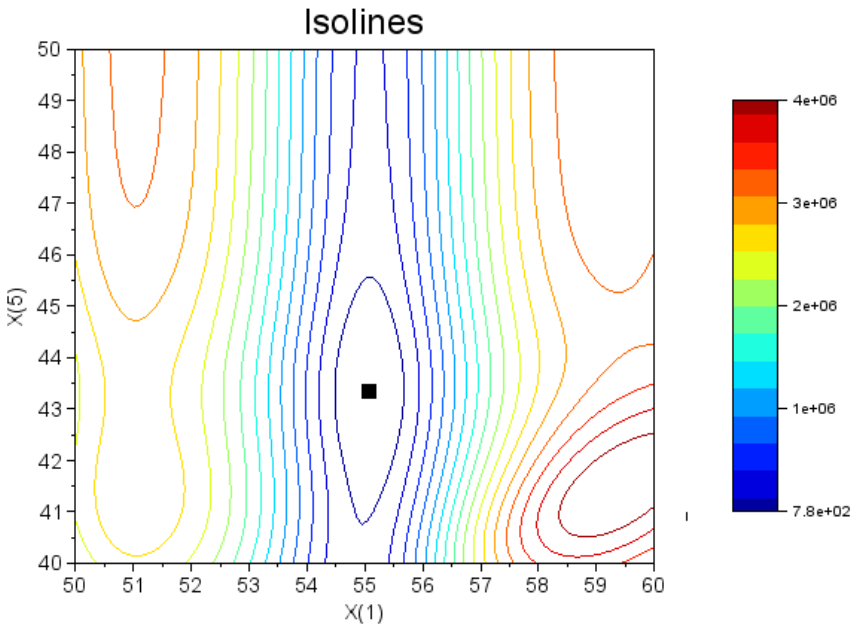
- Bounds and values of variables (final solution)



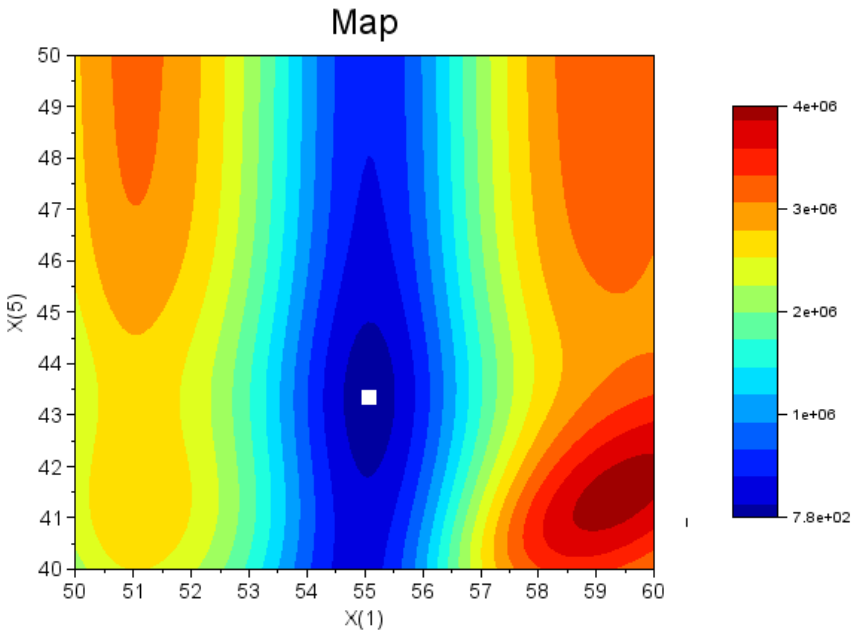
- Differences between AF and AM (final solution)



- Isolines

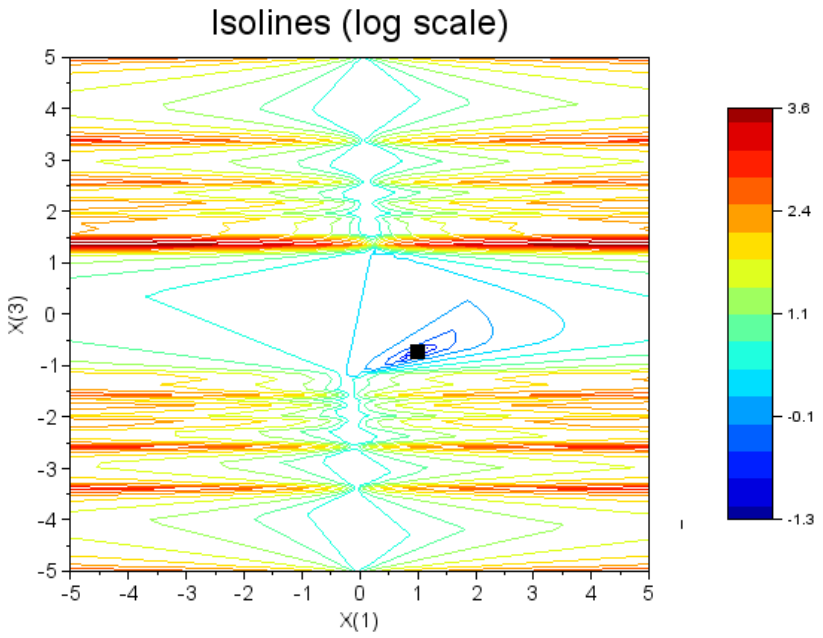


- Map

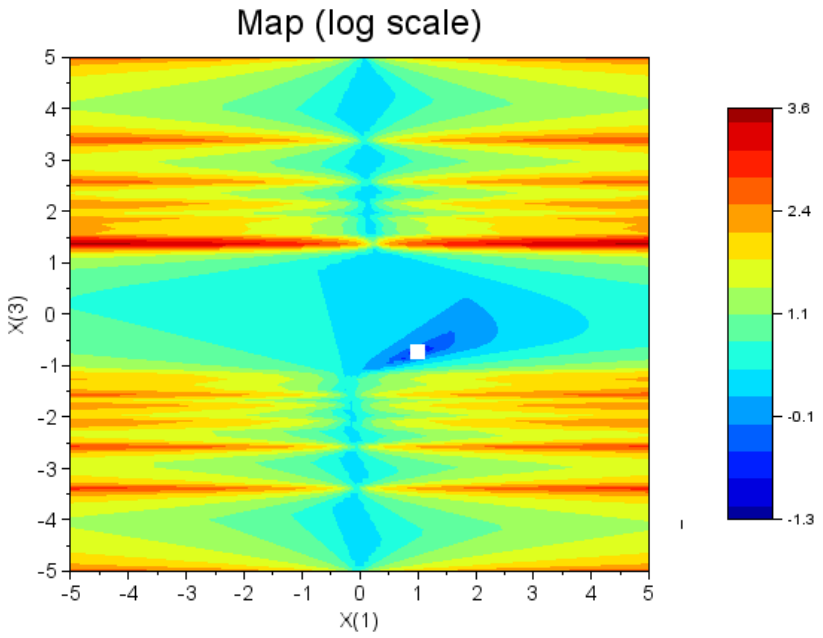


F.2 Nonlinear minimax optimization

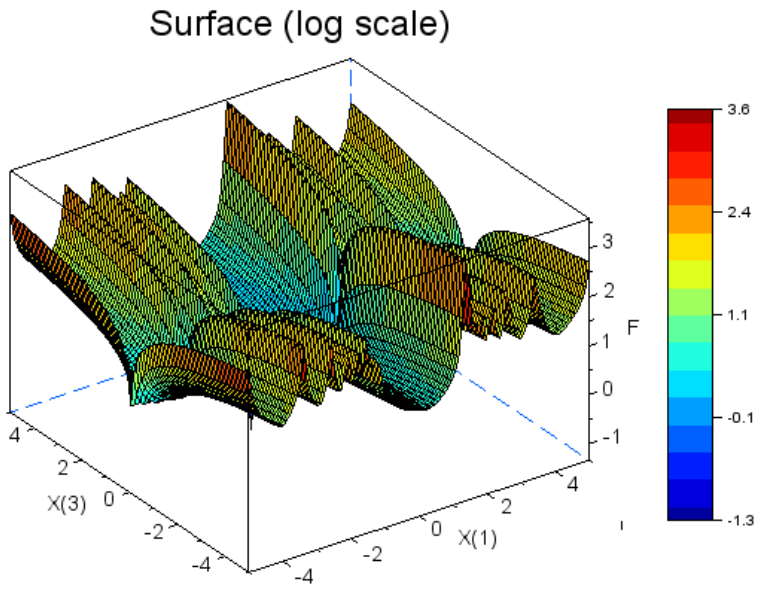
- Isolines



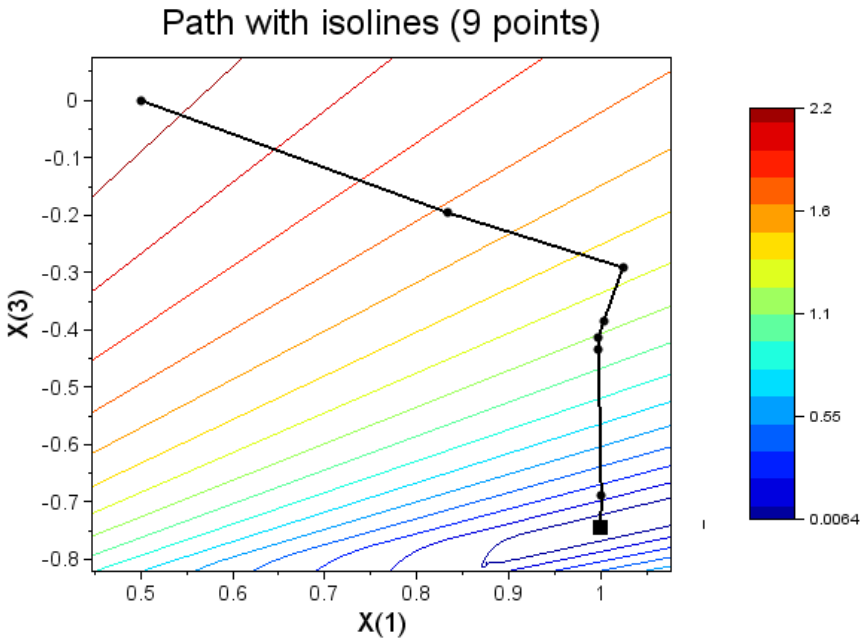
- Map



- Surface

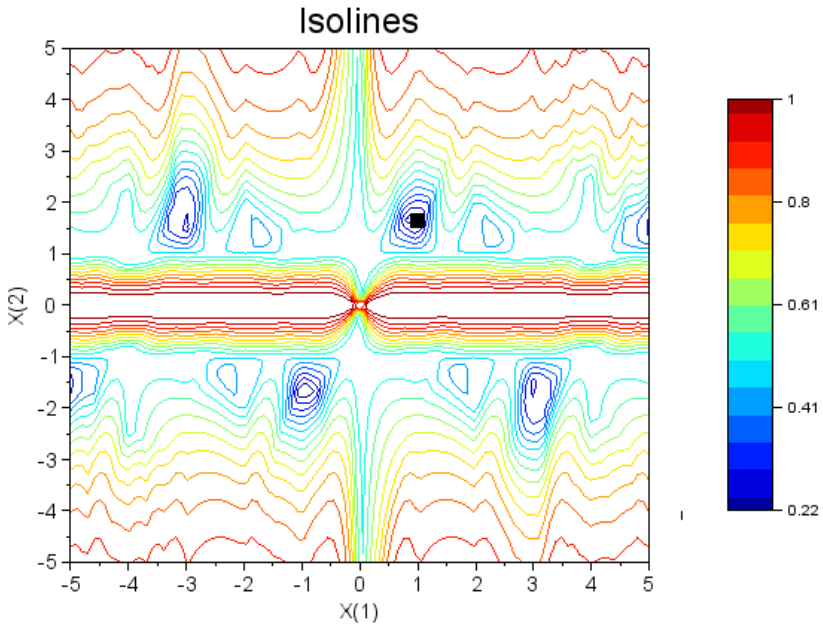


- Path with isolines

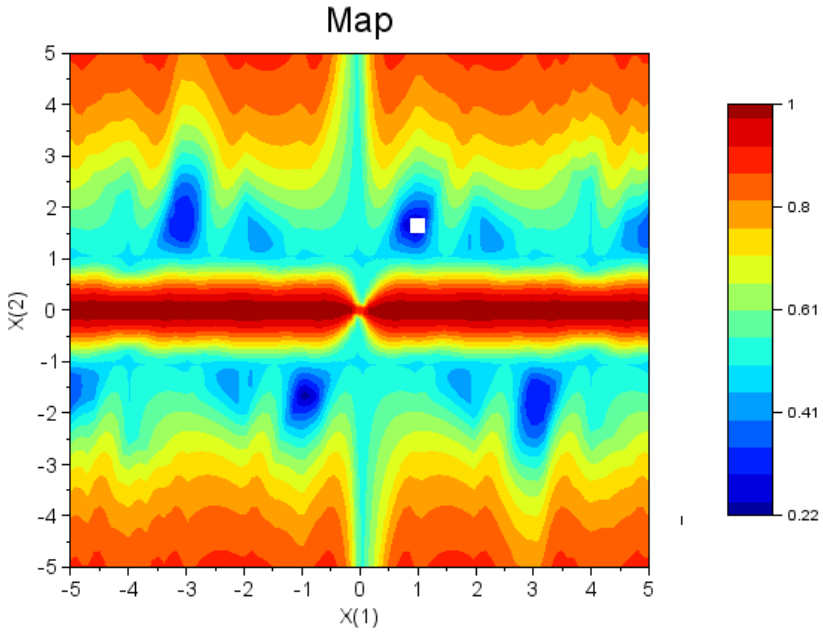


F.3 Transformer network design

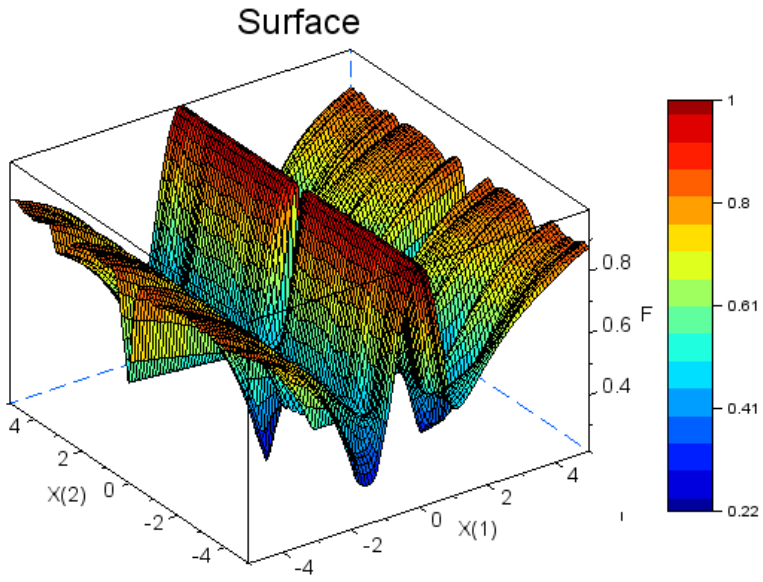
- Isolines



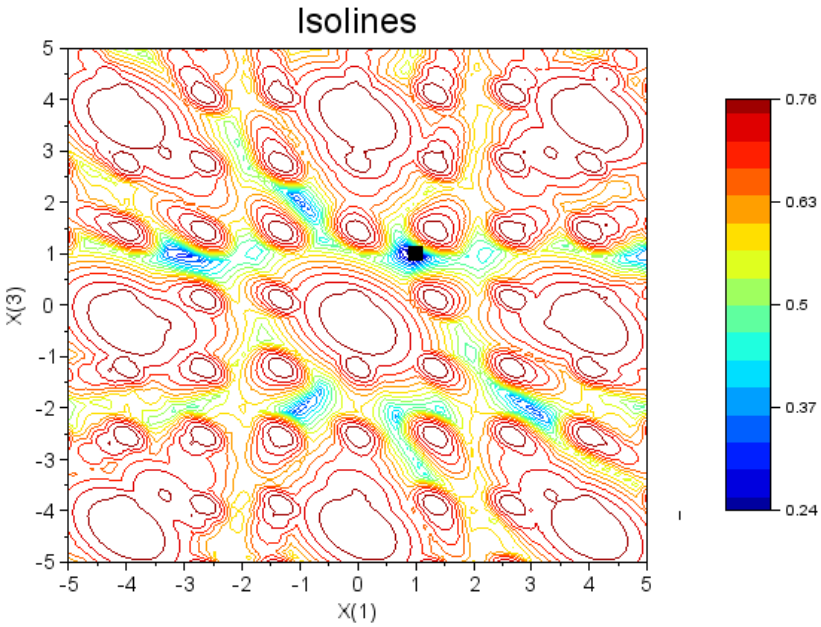
- Map



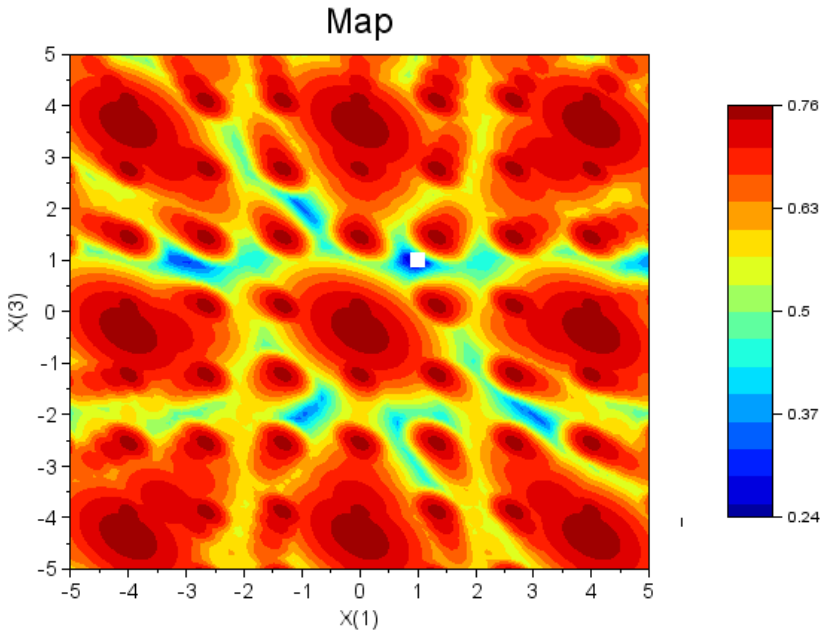
- Surface



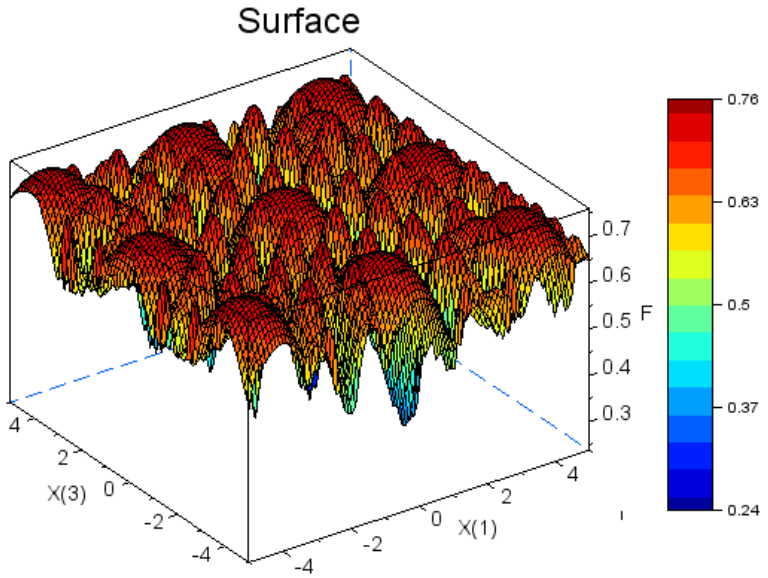
- Isolines



- Map

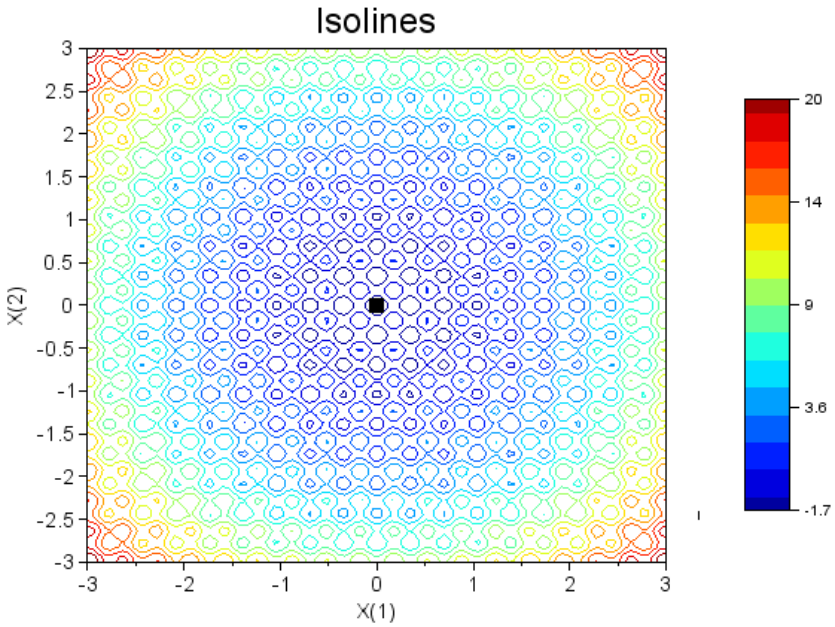

Map
X(3)
X(1)
5
4
3
2
1
0
-1
-2
-3
-4
-5
0.76
0.63
0.5
0.37
0.24

- Surface

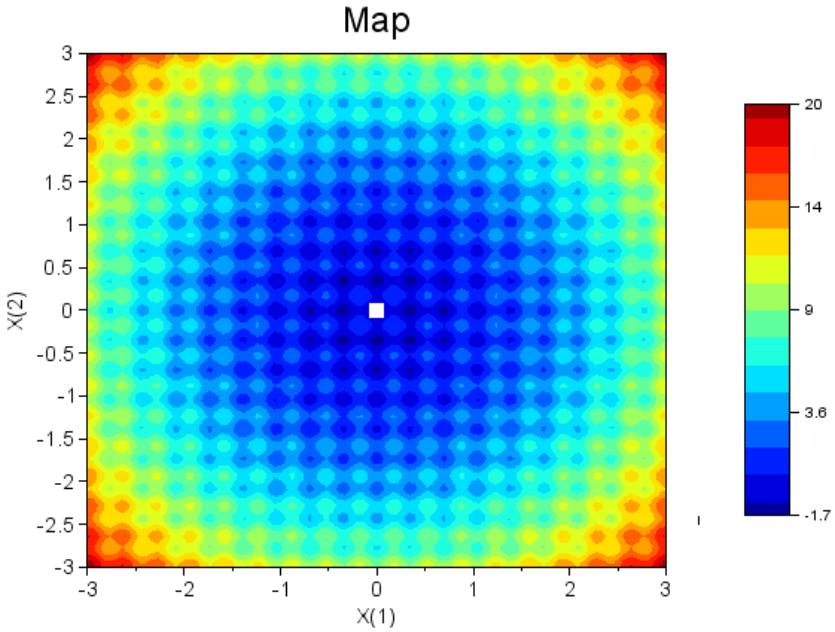

Surface
X(3)
X(1)
F
0.7
0.6
0.5
0.4
0.3
0.76
0.63
0.5
0.37
0.24

F.4 Global optimization

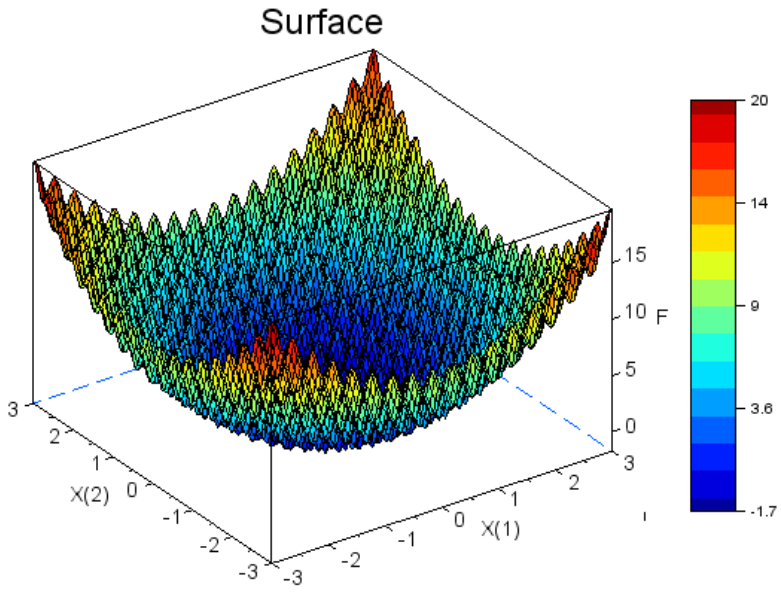
- Isolines



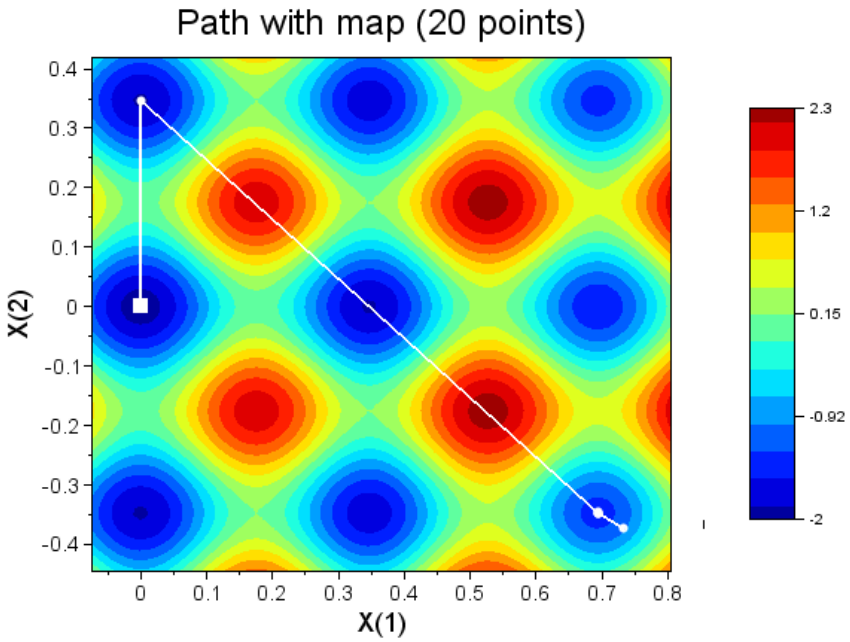
- Map



- Surface

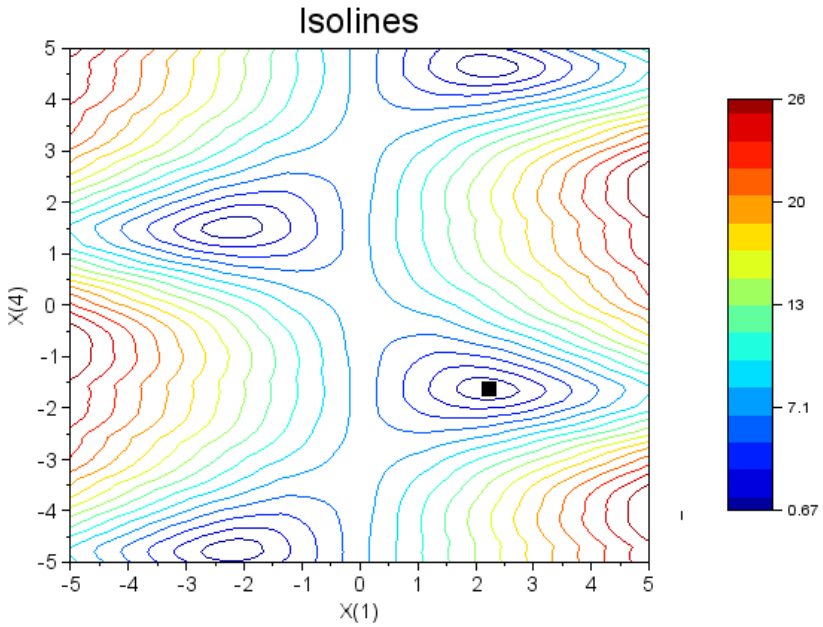


- Path with map

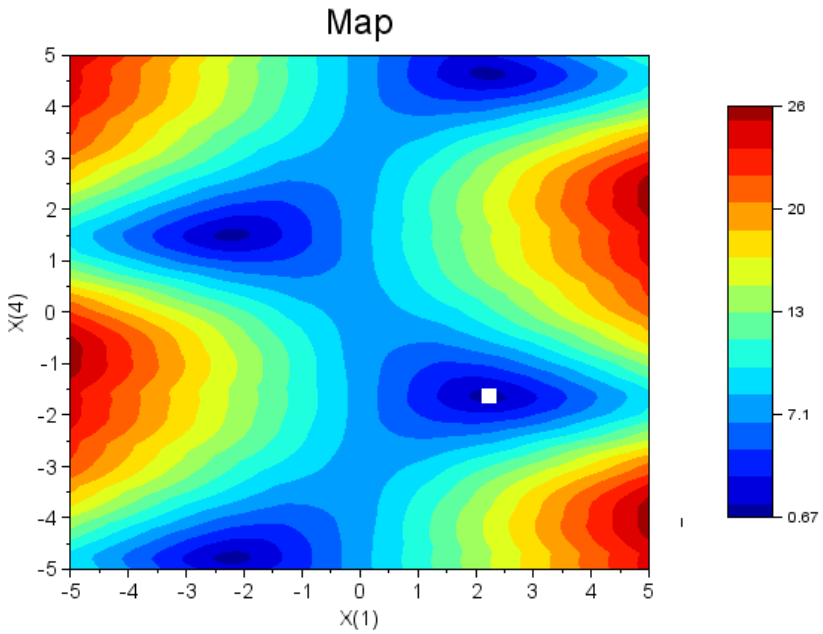


F.5 Nonsmooth optimization

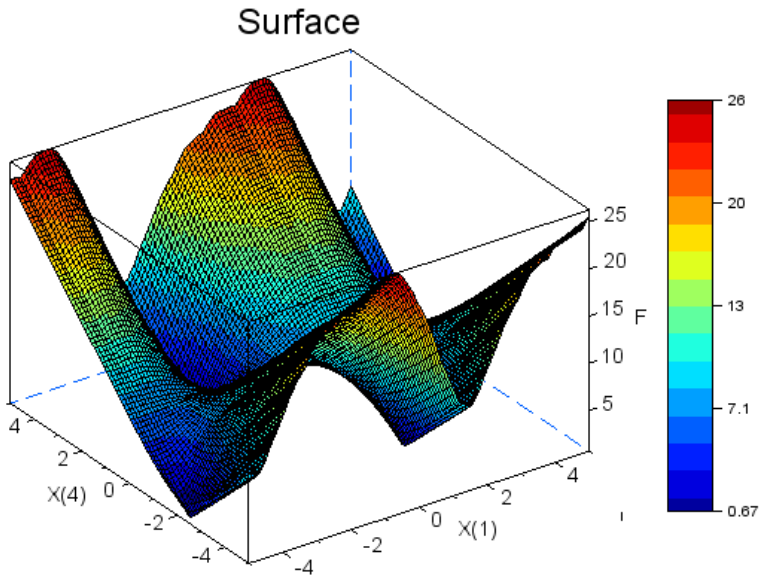
- Isolines



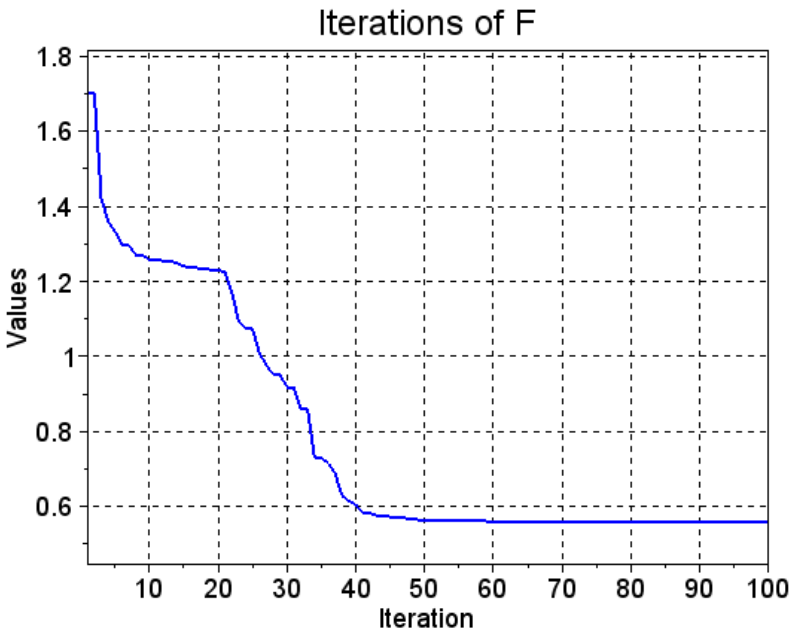
- Map



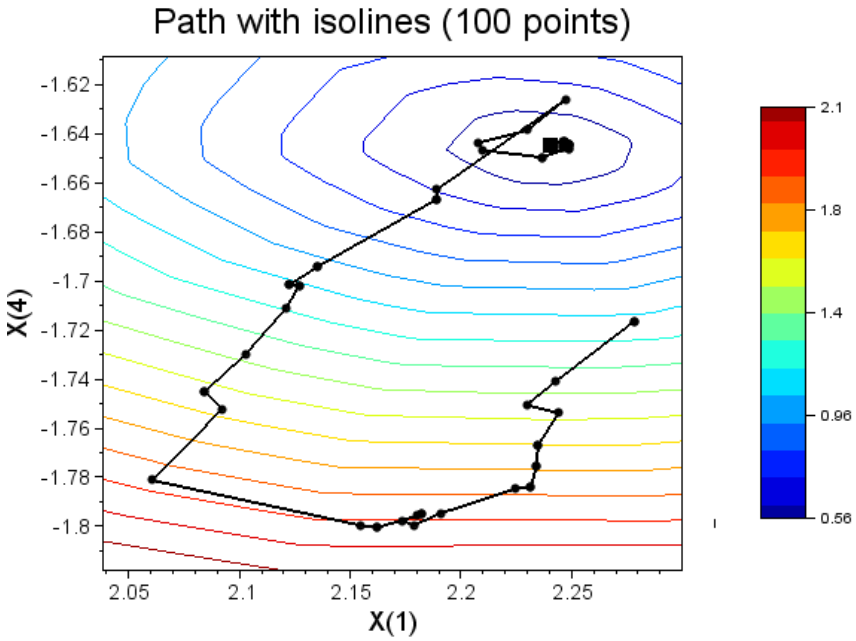
- Surface



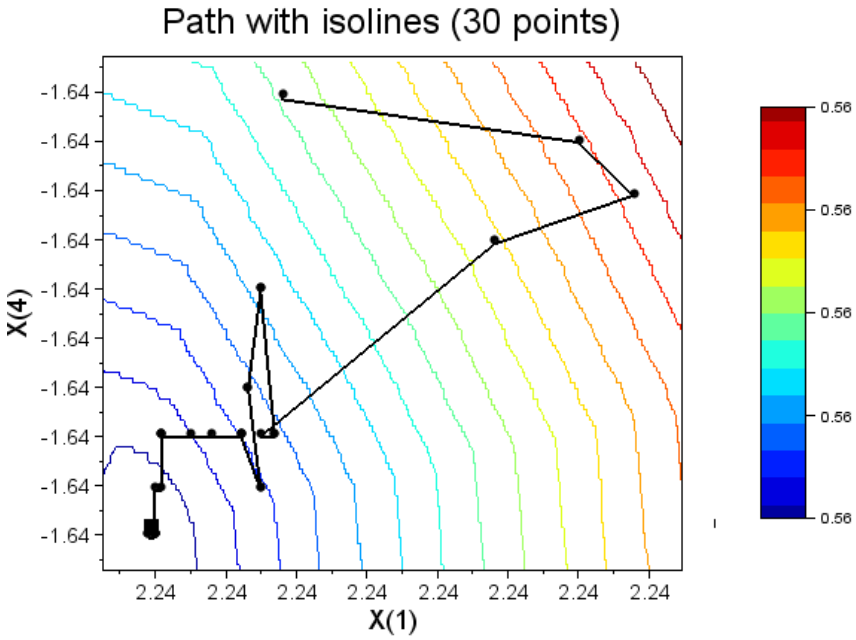
- Iterations of F



- Path with isolines (100 points)

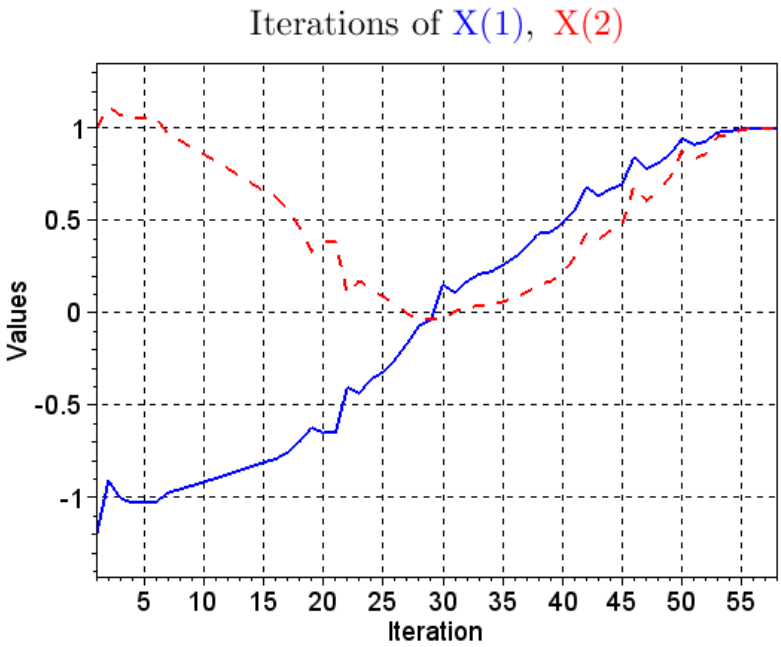


- Path with isolines (30 points)

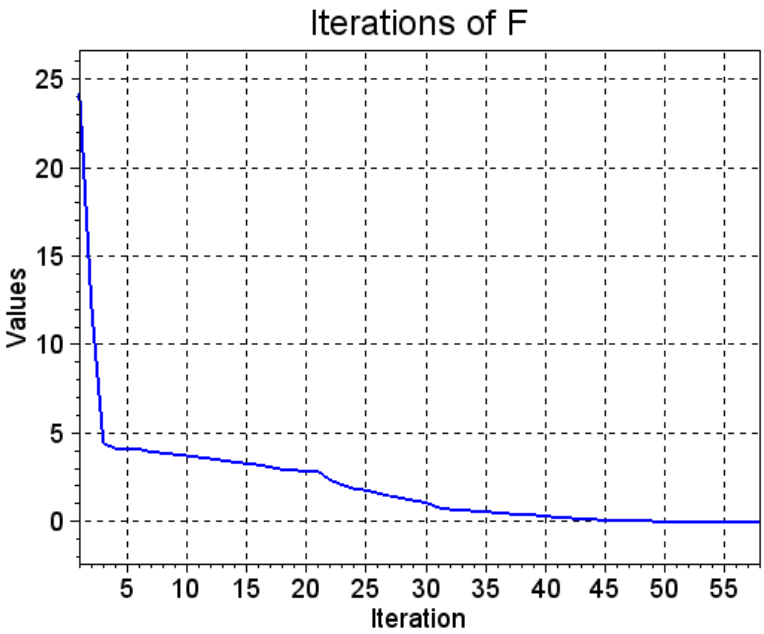


F.6 Rosenbrock function

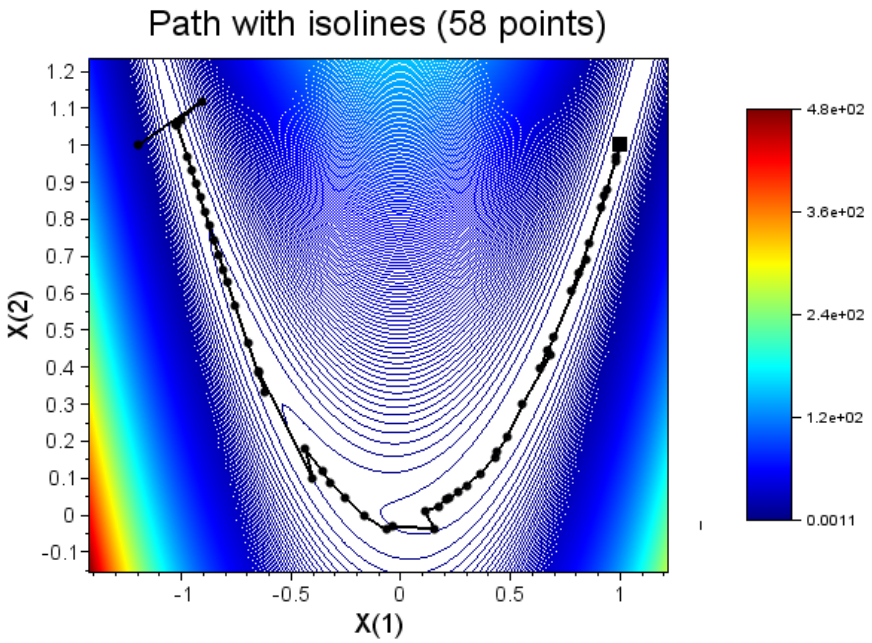
- Iterations of x_1, x_2



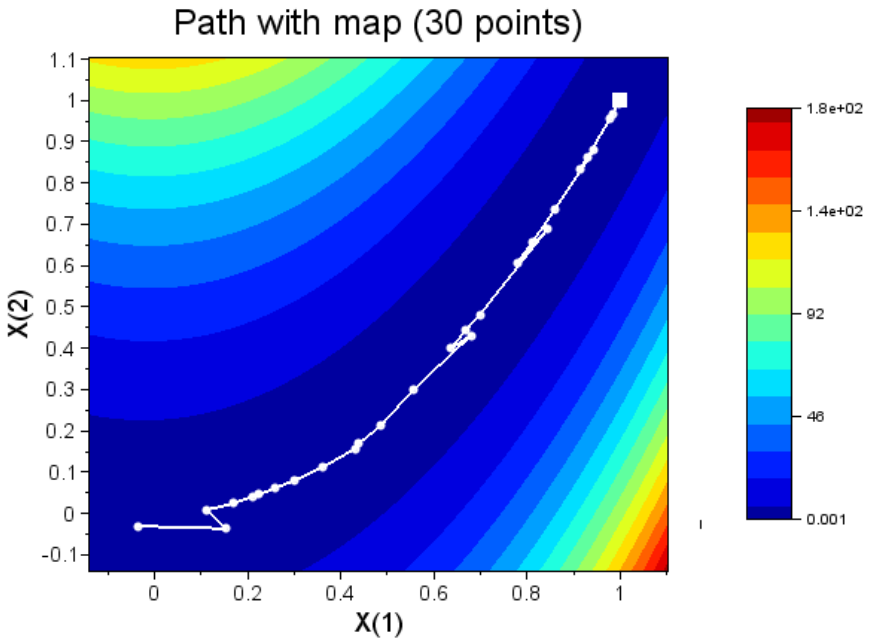
- Iterations of F



- Path with isolines

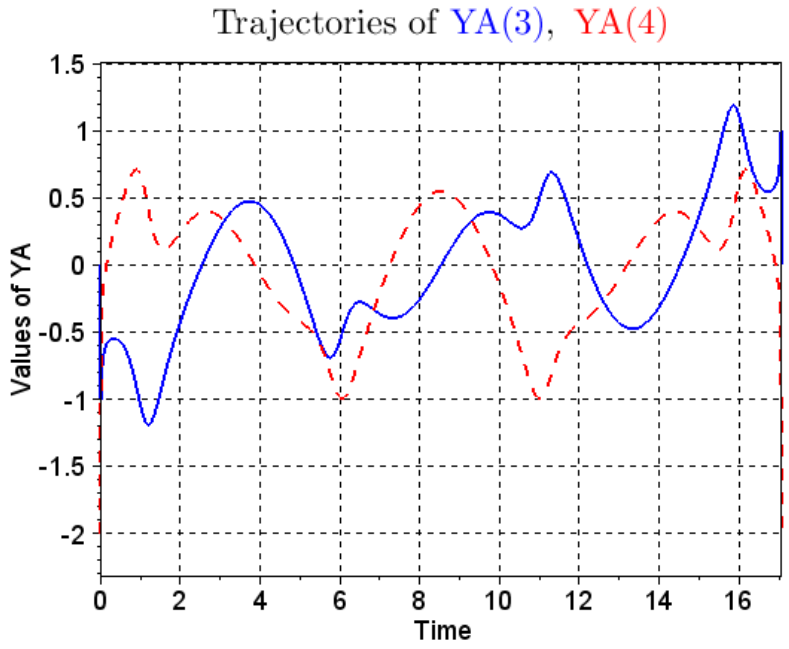


- Path with map

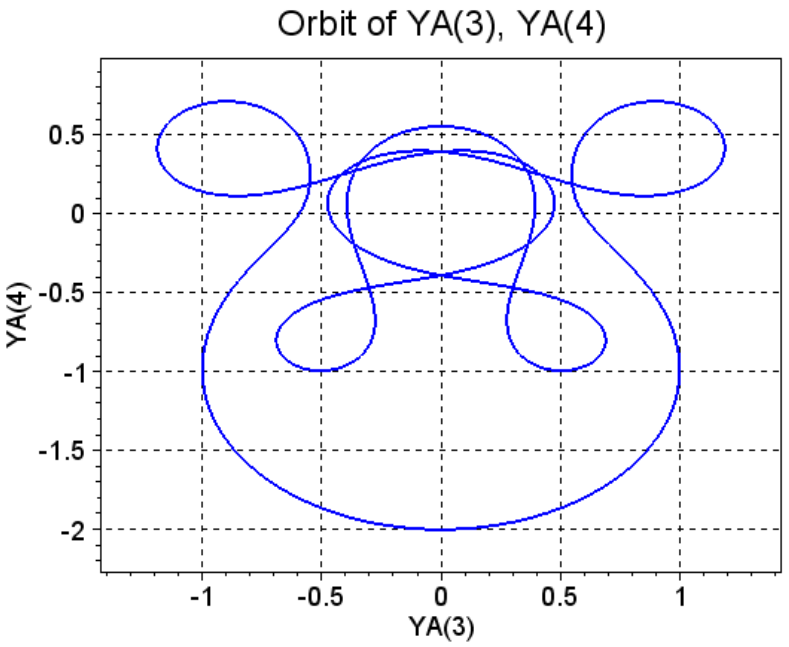


F.7 Ordinary differential equations

- Trajectories of y_3^A, y_4^A

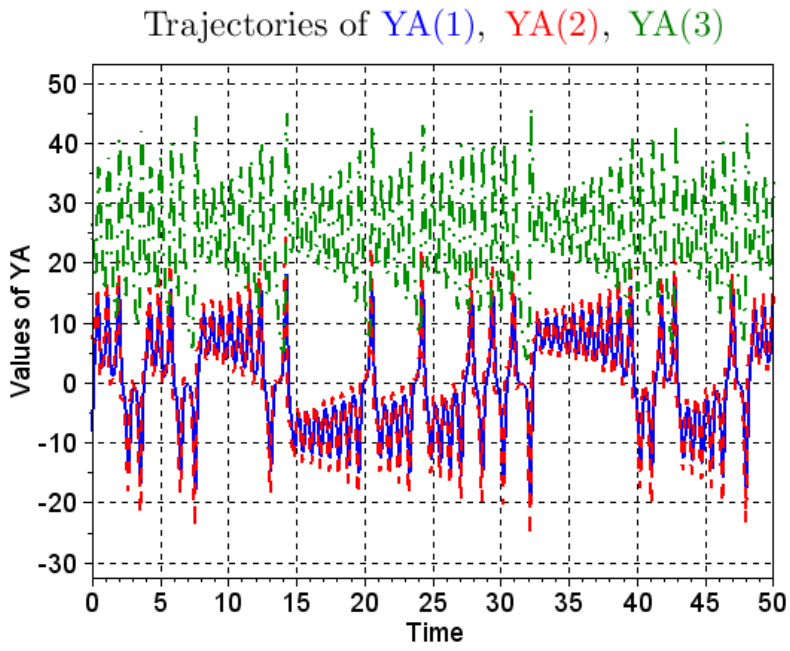


- Orbit of y_3^A, y_4^A

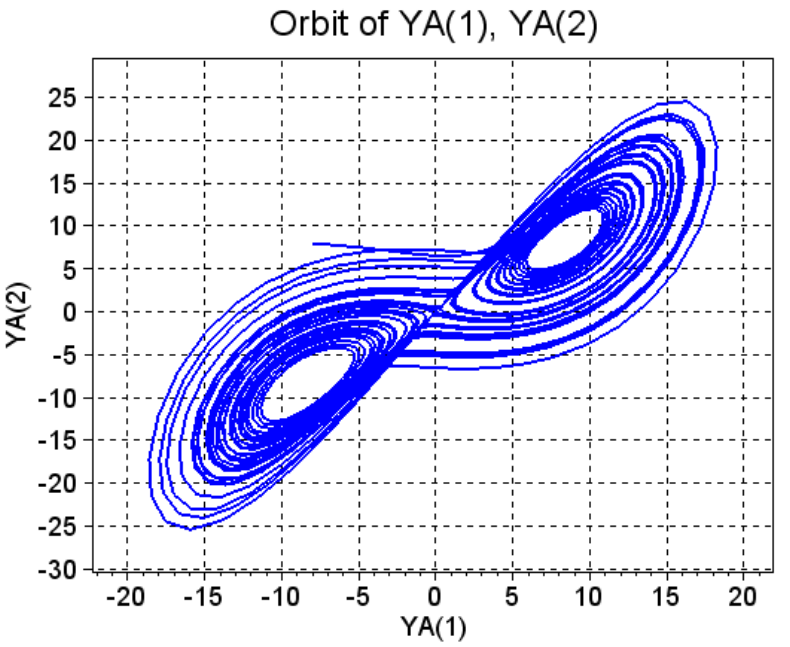


F.8 The Lorenz attractor

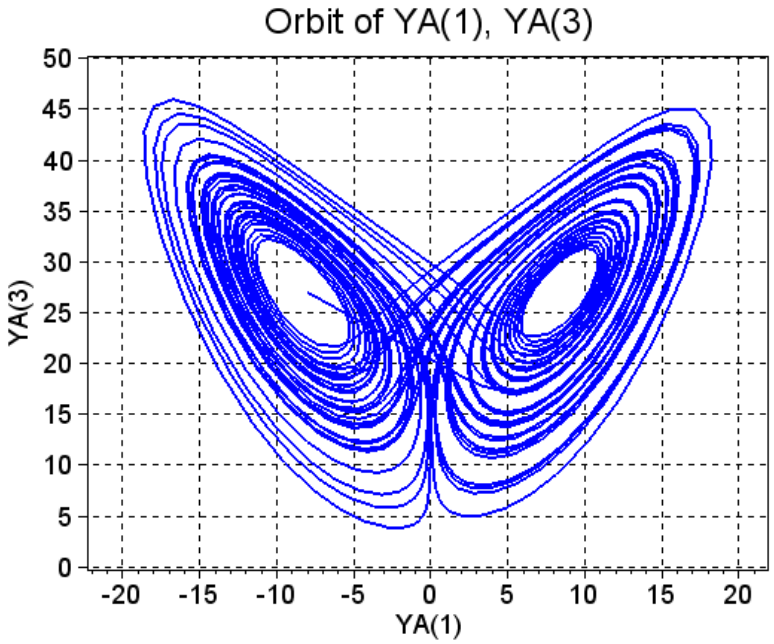
- Trajectories of y_1^A, y_2^A, y_3^A



- Orbit of y_1^A, y_2^A



- Orbit of y_1^A, y_3^A



- Orbit of y_2^A, y_3^A