Appendix C

Programs

C.1 Contents of the Floppy Disk

The floppy disk that accompanies this book contains:

- Sources of FORTRAN subroutines of the following direct optimization procedures as described in the Chapters 3, 5, and 7 of the book.

- FIBO	Coordinate strategy with Fibonacci division
fiboh (fiboh.f) calls	subroutine fibo (fibo.f)
- GOLD	Coordinate strategy with Golden section
goldh (goldh.f) calls	subroutine gold (gold.f)
- LAGR	Coordinate strategy with Lagrangian interpolation
lagrh (lagrh.f) calls	subroutine lagr (lagr.f)
- HOJE	Strategy of Hooke and Jeeves (pattern search)
hoje (hoje.f) calls	subroutine hilf (hilf.f)
- ROSE	Strategy of Rosenbrock (rotating coordinates search)
rose (rose.f) calls	subroutine grsmr (grsmr.f)
- DSCG	Strategy of Davies, Swann, and Campey
	with Gram-Schmidt orthogonalization
dscg (dscg.f) calls	subroutine lineg (lineg.f)
	subroutine grsmd (grsmd.f)
- DSCP	Strategy of Davies, Swann, and Campey
	with Palmer orthogonalization
dscp (dscp.f) calls	subroutine linep (linep.f)
	subroutine palm (palm.f)
- POWE	Powell's strategy of conjugate directions
powe (powe.f) calls	-
- DFPS	Davidon, Fletcher, Powell strategy (Variable metric)
dfps (dfps.f) calls	subroutine seth (seth.f)
	subroutine grad (grad.f)
	function updot (updot.f) calls dot (dot.f)
	function dot (dot.f)

- SIMP	Simplex strategy of Nelder and Mead
simp (simp.f) calls	-
- COMP	Complex strategy of M. J. Box
comp (comp.f) calls	-
- EVOL	Two membered evolution strategy
evol (evol.f) calls	function z (included in evol.f)
- KORR	Multimembered evolution strategy
korr2 (korr2.f) calls	function zulass (included in korr2.f)
	function gaussn (included in korr2.f)
	function bletal (included in korr2.f)
	subroutine pruefg (included in korr2.f)
	subroutine speich (included in korr2.f)
	subroutine mutati (included in korr2.f)
	subroutine umspei (included in korr2.f)
	subroutine minmax (included in korr2.f)
	subroutine gnpool (included in korr2.f)
	subroutine abscha (included in korr2.f)
	subroutine drehng (included in korr2.f)

Additionally, FORTRAN function sources of the 50 test problems are included:

- `ZIELFU(N,X)` one objective function with a computed `GOTO` for 50 entries.
- `RESTRI(J,N,X)` one constraints function with a computed `GOTO` for 50 entries and `J` as current number of the single restriction.

No runtime package is provided for this set, however.

- C sources for all strategies mentioned above and C sources for the 50 test problems (GRUP with option REKO is missing since it has become one special case within KORR).
- A set of simple interfaces to run 13 of the above mentioned optimization routines with the above mentioned 50 test problems on a PC or workstation.

C.2 About the Program Disk

The floppy disk contains both FORTRAN and C sources for each of the strategies described in the book. All test problems presented in the catalogue of problems (see appendix A) exist as C code. A set of simple interfaces, easy to understand and to expand, combines the strategies and functions to *OptimA*, a ready for use program package.

The programs are designed to run on a minimally configured PC using a math-coprocessor or having an 80486 CPU and running the DOS or LINUX operating system. To accomplish semantic equivalence with the well tested original FORTRAN codes, all strategies have been translated via `f2c`, a Fortran-to-C converter of AT&T Bell Laboratories. All C codes can be compiled and linked via `gcc` (Gnu C compiler, version 2.4). Of course,

any other ANSI C compiler such as Borland C++ that supports 4-byte-integers should produce correct results as well.
LINUX and `gcc` are freely available under the conditions of the GNU General Public License. Information about ordering the Gnu C compiler in the United States is available through the Free Software Foundation by calling 617 876 3296.

All C programs should compile and run on any UNIX workstation having `gcc` or another ANSI C compiler installed.

C.3 Running the C Programs

The following instructions are appropriate for installing and running the C programs on your PC or workstation. Installation as well as compilation and linking can be carried out automatically.

C.3.1 How to Install OptimA on a PC Using LINUX or on a UNIX Workstation

First, enter the directory where you want *OptimA* to be installed. Then copy the installation file via `mtools` by typing the command:

```
mcopy a:install.sh .
```

If you don't have `mtools`, copy `wb-1p?.tar` from floppy to workspace and untar it. The instruction

```
sh install.sh
```

will copy the whole tree of directories from the disk to your local directory. The following directories and subdirectories will be created:

```
fortran
funct
include
lib
rstrct
strat
util
```

To compile, link, and run *OptimA* go to the `workbench` directory and type

```
make
```

to start a recursive compilation and linking of all C sources.

C.3.2 How to Install OptimA on a PC Under DOS

First, enter the directory where you want *OptimA* to be installed. The instruction

`a:INSTALL`

or

`b:INSTALLB`

will copy the whole tree of directories from the disk to your local directory. The same directories and subdirectories as mentioned above will be created. To compile, link, and run *OptimA* go to the `workbench` directory and type

```
mkOptimA
```

to start a recursive compilation and linking of all C sources. This will take a while, depending on how fast your machine works.

C.3.3 Running OptimA

After the successful execution of `make` or `mkOptimA`, respectively, the executable file `OptimA` is located in the subdirectory `bin`. Here you can run the program package by issuing the command

```
OptimA
```

First, the program will list the available strategies. After choosing a strategy by typing its number, a list of test problems is displayed. Type a number or continue the listing by hitting the `return` key. Depending on the method and the problem, the program will ask for the parameters to configure the strategy. Please refer to Chapter 6 and Appendix A to choose appropriate values. Of course, you are free to define your own parameter values, but please remember that the behavior of each strategy strongly depends on its parameter settings.

Warnings during the process will inform the user of inappropriate parameter definitions or abnormal program behavior. For example, the message `timeout reached` warns the user that the strategy may find a better result if the user defined maximal time were set to a larger value. The strategies `COMP`, `EVOL`, and `KORR` will try at most five restarts after the first timeout occurred.

If a strategy that can process unrestricted problems only is applied to a restricted problem, a warning will be displayed, too. After the acknowledgement of this message by hitting the `return` key, the user can choose another function.

C.4 Description of the Programs

The following pages briefly describe the programs on which this package is based. A short description of how to incorporate self-defined problem functions to *OptimA* follows.

The directory FORTRAN lists all the original codes described in the book. The reader may write his own interfaces to these programs. For further information please refer to the C sources or to Schwefel (1980, 1981).

All C source codes of the strategies have been translated from FORTRAN to C via f2c. Some modifications in the C sources were done to gain higher portability and to achieve a homogeneous program behavior. For example, all strategies are minimizing, use standard output functions, and perform operations on the same data types. All modifications did not change the semantics of any strategy.

To each optimization method a dialogue interface has been added. Here the strategy's specific parameter definition takes place. In the comments within the program listings the meaning and usage of each parameter is briefly described. All names of the dialogue interfaces end with the suffix "_mod.c." The strategies together with the interfaces are listed in the directory named strat.

The whole catalogue of problems (see Appendix A) has been coded as C functions. They are collected in the subdirectory funct.

The problems 2.29 to 2.50 (see Appendix A) are restricted. Therefore, constraints functions to these problems were written and listed in directory rstrct. Because in some problems the number of constraints to be applied depends on the dimension of the function to be optimized, this number has to be calculated. This task is performed by the programs with prefix "rsn_." The evaluation of the constraint itself is done in the modules with prefix "rst_." A restriction holds if its value is negative.

All strategies perform operations on vectors of varying dimensions. Therefore a set of tools to allocate and to define vectors is compiled in the package vec_util which is located in the subdirectory util. The procedures from this package are used only in the dialogue interfaces. All other programs perform operations on vectors as if they would use arrays of arbitrary but fixed length.

The main program "OptimA.c" performs only initialization tasks and runs the dialogue within which the user can choose a strategy and a function number.

The strategies and functions are listed in tables, namely "func_tab.c" and "strt_tab.c." If the user wants to incorporate new problems to *OptimA* the table "func_tab.c" has to be extended. This task is relatively simple for a programmer with little C knowledge if he follows the next instructions carefully.

C.4.1 How to Incorporate New Functions

The following template is typical for every function definition:

```
#include "f2c.h"
#include "math.h"

doublereal     probl_2_18(int n,doublereal *x)
```

```
        {
                return(0.26*(x[0]*x[0] + x[1]*x[1])-0.48*x[0]*x[1] );
        }
```

Please add your own function into the directory `funct`. Here you will find the file "`func_tab.c`." Include the formal description of your problem into this table. A typical template looks like:

```
   {
        5,
        rs_nm_x_x,
        restr_x_x,
        "Problem x_x (restricted problem):\n\t x[1]+x[2]+... ",
        probl_x_x
   },
```

with the data type definition:

```
struct functions {
        long int        dim;                  /* Problem's dimension   */
        long int        (*rs_num)();          /* Calculates the number */
                                              /* of constraints        */
        doublereal      (*restrictions)();    /* Constraints function  */
        char*   name;                         /* Mathem. description   */
        doublereal      (*function)();        /* Objective function    */
};

typedef struct functions funct_t;
```

- The first item denotes the number of dimensions of the problem. A problem with variable dimension will be denoted by a -1. In this case the program should inquire the dimension from the user.
- The second entry denotes the function that calculates the numbers of constraints to be applied to the problem. If no constraints are needed a `NULL` pointer has to be inserted.
- The next line will be displayed to the user during an *OptimA* session. This string provides a short description of the problem, typically in mathematical notation.
- The last item is a function-pointer to the objective function.

Please do not add a new formal problem description into the `func_tab` behind the last table entry. The latter denotes the end of the table and should not be displaced.

To inform all problems of the new function, its prototype must be included into the header file `func_names.h`.

As a last step the `Makefile` has to be extended. The lists `FUNCTSRCS` and `FUNCTOBJS` denote the files that make up the list of problems. These lists have to be extended by the filename of your program code.

Now step back to the directory `C` and issue the command `make` or `mkOptimA`, respectively, to compile "`OptimA`."

Restrictions can be incorporated into *OptimA* like functions. Every C code from the directory `rstrct` can be taken as template. The name of the constraints function and the name of the function that calculates the number of constraints has to be included in the formal problem description.

C.5 Examples

Here two examples of how *OptimA* works in real life will be presented. The first one describes an application of the multimembered evolution strategy KORR to the corridor model (problem 2.37, function number 32). The second example demonstrates a batch run. The batch mode enables the user to apply a set of methods to a set of functions in one task.

C.5.1 An Application of the Multimembered Evolution Strategy to the Corridor Model

After calling `OptimA` and choosing problem 2.37 by typing `32`, a typical dialogue will look like:

```
  Multimembered evolution strategy applied to function:

  Problem 2.37 (Corridor model) (restricted problem):
          Sum[-x[i],{i,1,n}]


Please enter the parameters for the algorithm:
Dimension of the problem                              : 3
Number of restrictions                                : 7


Number of parents                                     : 10
Number of descendants                                 : 100
Plus (p) or the comma (c) strategy                    : c
Should the ellipsoid be able to rotate (y/n)          : y
You can choose under several recombination types:
```

```
1    No recombination
2    Discrete recombination of pairs of parents
3    Intermediary recombination of pairs of parents
4    Discrete recombination of all parents
5    Intermediary recombination of all parents in pairs
Recombination type for the parameter vector                 : 2
Recombination type for the sigma vector                     : 3
Recombination type for the alpha vector                     : 1
Check for convergence after how many generations (> 2*Dim.) : 10
Maximal computation time in sec.                            : 30
Lower bound to step sizes, absolute                         : 1e-6
Lower bound to step sizes, relative                         : 1e-7
Parameter in convergence test, absolute                     : 1e-6
Parameter in convergence test, relative                     : 1e-7
Common factor used in step-size changes (e.g. 1)            : 1
Standard deviation for the angles
of the mutation ellipsoid (degrees)                         : 5.0
Number of distinct step-sizes                               : 3
Initial values of the variables :
0
0
0
Initial step lengths :
1
1
1

Common factor used in step-size changes     : 0.408248
Individual factor used in step-size changes : 0.537285

Starting at : F(x) = 0


Time elapsed         : 18.099276

Minimum found : -300.000000
at point : 99.999992 100.000000 99.999992
Current best value of population: -300.000000
```

C.5.2 OptimA Working in Batch Mode

OptimA also supports a batch mode option. This option was introduced to enable a user to test the behavior of any strategy by varying parameter settings automatically. Of course, any function or method may be changed during a run, as well. The batch file that will be processed should contain the list of input data you would type in manually during a whole session in non-batch mode. *OptimA* in batch mode suppresses the listing of the strategies and functions. That reduces the output a lot and makes it better readable.

A typical batch run looks like:

```
  OptimA -b < bat_file > results
```

With a "`bat_file`" like:

```
8
1
100.100
0.98e-6
0.0e+0
5
5
1
1
0.8e-6
0.8e-6
0.111
0.111
y
```

the file "`results`" may look like:

```
Method #   : 8
Function # : 1

  DFPS strategy (Variable metric) applied to function:

  Problem 2.1 (Beale):
          (1.5-x*(1-y))^2 + (2.25-x*(1-y^2))^2 + (2.625-x*(1-y^3))^2
Dimension of the problem                  : 2
Maximal computation time in sec.          : 100.100000
Accuracy required                         : 9.8e-07
Expected value of the objective function
at the optimum                            : 0
Initial values of the variables :
5
5
Initial step lengths :
1
1
Lower bounds of the step lengths :
```

```
8e-07
8e-07
Initial step lengths for construction of derivatives :
0.111
0.111

Starting at : F(x) = 403069

Time elapsed         : 0.033332

Minimum found : 0.000000
at point : 3.000000 0.500000
```

Both examples have been run on a SUN SPARC S10/40 workstation.

Index