CUTE:
Constrained and unconstrained testing environment

by I. Bongartz[1], A.R. Conn[1],
Nick Gould[2], and Ph.L. Toint[3]

Report 93/10(revised) October 13, 1993

[1] IBM T.J. Watson Research Center, P.O.Box 218, Yorktown Heights, NY 10598, USA
Email : arconn@watson.ibm.com or bongart@watson.ibm.com

[2] CERFACS, 42 Avenue Gustave Coriolis, 31057 Toulouse Cedex, France, EC
Email : gould@cerfacs.fr or nimg@directory.rl.ac.uk
Current reports available by anonymous ftp from the directory
"pub/reports" on camelot.cc.rl.ac.uk (internet 130.246.8.61)

[3] Department of Mathematics, Facultés Universitaires ND de la Paix, 61, rue de Bruxelles,
B-5000 Namur, Belgium, EC
Email : pht@math.fundp.ac.be
Current reports available by anonymous ftp from the directory
"pub/reports" on thales.math.fundp.ac.be (internet 138.48.4.14)

Keywords : Large-scale, nonlinear optimization, test problems, evaluation tools.

CUTE: Constrained and unconstrained testing environment

I. Bongartz A.R. Conn Nick Gould Ph.L. Toint

October 13, 1993

Abstract

The purpose of this paper is to discuss the scope and functionality of a versatile environment for testing small and large-scale nonlinear optimization algorithms. Although many of these facilities were originally produced by the authors in conjunction with the software package LANCELOT, we believe that they will be useful in their own right and should be available to researchers for their development of optimization software. The tools are available by anonymous ftp from a number of sources and may, in many cases, be installed automatically.

The scope of a major collection of test problems written in the standard input format (SIF) used by the LANCELOT software package is described. Recognising that most software was not written with the SIF in mind, we provide tools to assist in building an interface between this input format and other optimization packages. These tools already provide a link between the SIF and an number of existing packages, including MINOS and OSL. In addition, as each problem includes a specific classification that is designed to be useful in identifying particular classes of problems, facilities are provided to build and manage a database of this information.

There is a UNIX and C-shell bias to many of the descriptions in the paper, since, for the sake of simplicity, we do not illustrate everything in its fullest generality. We trust that the majority of potential users are sufficiently familar with UNIX that these examples will not lead to undue confusion.

1 Introduction

The purpose of this paper is to describe an environment that we have devised for testing small and large-scale nonlinear optimization algorithms. It is inevitable that, during the process of developing a software package, the designers concern themselves with problems of testing. Thus, in a major project, one has to consider how to obtain and specify a suitable collection of test problems. If the collection of test problems is to be reasonably comprehensive, it will require careful management. In addition, one might presume that researchers would like to compare different algorithmic approaches to a problem. All of these situations occurred during our own researches and, indeed, many of the facilities described in this paper were originally produced and tested in conjunction with the software package LANCELOT (see Conn *et al.* (1992a)).

[0]This research was supported in part by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Air Force Office of Scientific Research under Contract No F49620-91-C-0079. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

In this paper, we discuss the scope of our constrained and unconstrained testing environment (CUTE) for nonlinear programming algorithms. We describe in some depth a collection of test problems that we have gathered from a variety of academic and real-life sources. Furthermore, we discuss a number of tools that we have produced to manage the resulting test-problem database. An additional set of Fortran tools, which are intended to facilitate the building of an interface between the test-problem collection and potential optimization software packages (interfacing, for short) is also described. These tools have to be rather general as there is, as yet, no consensus in the optimization community as to what constitutes an ideal interface between problems and algorithms.

We have tried to order the content so that it corresponds to the requirements of a potential user who might pose the following sequence of questions:

- I want to solve a (class of) problem(s). What is available?
- My problem is not available in the current test set. I would like to construct it and include it in my classification database. How do I proceed?
- I now know which problem(s) I want to solve. Which solvers are available and how do I use them?
- The solver I would like to use is not available. How can I build a suitable Fortran interface between my specific (class of) problem(s) and my solver of choice using the tools supplied?

The facilities offered are intended to be useful to the general optimization community — and especially to researchers — for the development of optimization software. All the test problems in our collection are written in the standard input format (SIF) required by the LANCELOT software package. Each problem comes with a classification that is designed to be useful in identifying particular classes of problems. We provide an automatic facility for selecting interesting subsets of the test problems in any collection conforming to this classification system. We also provide tools for maintaining a database of classified problems to allow for the introduction of new members.

We provide additional tools to allow an interface between problems, specified using the SIF, and other existing nonlinear programming packages. Since most software was not written to interface with this format, it is important to provide such tools. Furthermore, Fortran subroutines are supplied to take input written in SIF and return function and derivative values in various dense and sparse formats, thus providing a relatively easy means of building interfaces with new algorithms. Details of both the SIF and the software package LANCELOT are provided in Conn *et al.* (1992a). Of course, CUTE does not include the source, object or executable files for any external optimization packages: to obtain these files, the user should contact the appropriate authors (or, when available, transfer the files from the appropriate ftp site).

We conclude the paper by describing how the files that make up CUTE may be obtained and how the accompanying tools installed on a user's machine. Further details are provided in a number of appendices.

2 The test problem database

2.1 The scope of the collection

The provided collection of nonlinear test problems is intended to be a growing set of test problems written in the standard input format (SIF) required by LANCELOT. This collection contains a large number of nonlinear optimization problems of various sizes and difficulty, representing both 'academic' and 'real world' applications. Both constrained and unconstrained examples are included. On October 1, 1993 the database consisted of 608 *different* problems. Because many of these problems allow various choices for the number of variables and/or the number of constraints, many more instances can be defined by the user. More than a thousand such instances are suggested by appropriate comments in the problem definitions.

It is clearly impossible to describe all these examples in the present paper. It will suffice to say that the test set covers, amongst others,

- the 'Argonne test set' (Moré *et al.* (1981)), the Testpack report (Buckley (1989)), the Hock and Schittkowski collection (Hock and Schittkowski (1981)), the Dembo network problems (Dembo (1984)), the Moré-Toraldo quadratic problems (Moré and Toraldo (1991)), the Boggs-Tolle problems (Boggs and Tolle (1989)), the Toint-Tuyttens network model problems (Toint and Tuyttens (1992)), and Gould's quadratic programming problems (Gould (1991)),
- most problems from the PSPMIN collection (Toint (1983)),
- problems inspired by the orthogonal regression report by Gulliksson (Gulliksson (1990)),
- some problems from the Minpack-2 test problem collection (Averick *et al.* (1991), Averick and Moré (1991)) and from the second Schittkowski collection (Schittkowski (1987)) and
- a large number of original problems from a variety of application areas.

Moreover, not only can one assume that the collection will grow steadily, but because of the LANCELOT licensing agreement, which requires the submission of typical problems by users of the package, the new problems are likely to be even more user-oriented than the present ones. Furthermore, because the SIF format is an extension of the MPS linear programming format (see International Business Machines Corporation (1978)), access to suites of linear programming test problems (see, for example, Gay (1985)) is possible. Tables 1 and 2 are included to indicate the scope of the applications that are currently represented in the database. Also included are some other test problems of interest to optimization algorithm designers.

As can be seen from these tables, many application areas are represented in the current collection and the coverage is likely to become even wider. It can also be seen that the dimension and number of constraints (excluding simple bounds) covers a wide range. About one third of the problems in the present collection involve constraints more general than simple bounds. Both large and small-scale problems are included. The collection also contains a number of problems posed as nonlinear systems of equations. The SIF file associated with each problem contains a reference to its statement in the literature, when appropriate.

Problem	Number of variables	Number of constraints	SIF file name
Engineering			
Aircraft stability	9	9	AIRCRFTA
Clamped plate problem	4970	0	CLPLATEA
Elastic-plastic torsion problem	15625	0	TORSION1
Heat exchanger design	8	1	HS106
Journal bearing problem	15625	0	JNLBRNGA
Membrane separation	13	15	HS116
Nonlinear network gas flow problem	2734	2727	BRIDGEND
Optimization of electrical network	6	4	HS87
Oscillation problem in structural mechanics	5041	0	SVANBERG
Static power scheduling	9	6	HS107
Time optimal heat conduction	2	1	HS88
Transformer design	6	2	HS93
Three-stage launch vehicle design and costing	25	28	LAUNCH
Minimal weight rotating disc design	905	1081	ROTDISC
Airplane design	49	32	AVION2
Applied mathematics			
1-d variational problem from ODEs	1001	0	BRATU1D
2-d variational problem from PDEs	5184	0	BRATU2D
3-d variational problem from PDEs	4913	0	BRATU3D
Applied geometry	6	15	PENTAGON
Banana shaping	2	0	ROSENBR
Determination of a turning point in curve tracing	5002	0	TRIGGER
Discretized boundary value	5002	0	BDVALUE
Eigenvalue calculations	5000	0	VAREIGVL
Matrix square root	10000	0	MSQRTA
Maximum pivot growth in Gaussian Elimination	3946	3690	GAUSSELM
Minimum surface problem with nonlinear boundary conditions	15625	0	NLMSURF
Nonlinear network problem on a square grid	13284	6724	GRIDNETA
Nonlinear optimal control problem	10000	5000	HAGER4
Number theory	6	15	LEWISPOL
Obstacle problem	15625	0	OBSTCLBU
Optimal knot placement for ODE solution	649	349	GOULDQP2
Orthogonal regression problem	8197	4096	ORTHREGA
Quadrature rule determination	18	18	NYSTROM5
Rational approximation	5	502	EXPFITC
Determination of smallest crescent containing given points	6	2654	CRESC132

Figure 1: Some typical examples (1).

Problem	Number of variables	Number of constraints	SIF file name
Physics			
Quantum physics	600	1	LCH
Radiative transfer	100	100	CHANDHEQ
Seismic ray bending	246	0	RAYBENDS
Semiconductor analysis	1002	1000	SEMICON2
Thermistor modelling	3	0	MEYER3
Tide modelling	5000	10000	READING3
Vibrating beam modelling	8	60	VIBRBEAM
Chemistry and Biology			
Alkylation	10	11	HS114
Chemical equilibrium	43	14	HIMMELBJ
Chemical kinetics	8	0	PALMER1C
Chemical reaction problem	5000	5000	CHEMRCTA
Distillation column modelling	99	99	HYDCAR20
Enzyme reaction modelling	4	0	KOWOSB
Dipole model of the heart	8	8	HEART8
Fluid catalytic cracker modelling	19	8	FCCU
Economy and Operations Research			
Cattle feeding	4	3	HS73
Computer production planning	500	0	PRODPL1
Cost optimal inspection plan	4	2	HS68
Economic equilibrium	1825	730	MANNE
Economic model from Thailand	2230	1112	BIGBANK
Electric power generation	194	120	SSEBNLN
Hydro-electric reservoir management	2017	1008	HYDROELL
Optimal sample sizing	4	2	HS72
Nonlinear blending	24	14	HIMMELBK
Traffic equilibrium	540	126	STEENBRE
Transportation demand estimation	11276	0	ODNAMUR
Water distribution	906	666	DALLASL
Weapon assignment	100	12	HIMMELBI

Figure 2: Some typical examples (2).

A summary of details on the test results obtained with various of the options available with the software package LANCELOT is given in Conn *et al.* (1992b). A technical report containing the complete results (including a more detailed description of each problem's characteristics and the complete set of results obtained on nearly twenty thousand separate runs) is also available (see Conn *et al.* (1993a)).

The current size of the total uncompressed database of test problems is about fifty seven megabytes. Since this database is substantial, it is available in two separate compressed files, one containing the smaller problems (`mastsif.small.tar.Z`) and the other the larger problems (`mastsif.large.tar.Z`). Individual problems are also available. In a UNIX environment, we assume that the directory in which these problems reside is pointed to by the environment variable `MASTSIF`. We also assume that the directory where CUTE is installed (see Section 7, below, for more details) is pointed to by the environment variable `CUTEDIR`.

2.2 The classification scheme

Each of the problems in our dataset comes with a simple problem classification inspired by the scheme of Hock and Schittkowski (1981), which was itself a slight extension of a scheme adopted by Bus (1977). The scope of such a classification could be very broad indeed and we realise that a general scheme that encompasses much more than we have succeeded in doing could be very useful for large databases such as ours. However, we have consciously limited the scope of our classification as follows.

A problem is classified by the string

XXXr-XX-n-m

This string must not contain any blanks. In what follows, we state the admissible letters and integers, together with their interpretation, for each character in the classification string. Note that all letters must be given in upper case.

The first character in the string defines the type of the problem objective function. Its possible values are

N no objective function is defined,

C the objective function is constant,

L the objective function is linear,

Q the objective function is quadratic,

S the objective function is a sum of squares, and

O the objective function is none of the above.

The second character in the string defines the type of constraints of the problem. Its possible values are

U the problem is unconstrained,

X the problem's only constraints are fixed variables,

B the problem's only constraints are bounds on the variables,

N the problem's constraints represent the adjacency matrix of a (linear) network,

L the problem's constraints are linear,

Q the problem's constraints are quadratic, and

O the problem's constraints are more general than any of the above alone.

The third character in the string indicates the smoothness of the problem. There are two choices:

R the problem is regular, that is its first and second derivatives exist and are continuous everywhere, or

I the problem is irregular.

The integer (r) which corresponds to the fourth character of the string is the degree of the highest derivatives provided analytically within the problem description. It is restricted to being one of the single characters `0`, `1` or `2`.

The character immediately following the first hyphen indicates the primary origin and/or interest of the problem. Its possible values are

A the problem is academic, that is, has been constructed specifically by researchers to test one or more algorithms,

M the problem is part of a modelling exercise where the actual value of the solution is not used in a genuine practical application, and

R the problem's solution is (or has been) actually used in a real application for purposes other than testing algorithms.

The next character in the string indicates whether or not the problem description contains explicit internal variables (see the description of group partial separability in Conn *et al.* (1990) or section 8.2.2.2 of Conn *et al.* (1992a)). There are two possible values, namely

Y the problem description contains explicit internal variables, or

N the problem description does not contain any explicit internal variables.

The symbol(s) between the second and third hyphen indicate the number of variables in the problem. Possible values are

V the number of variables in the problem can be chosen by the user, or

n a positive integer giving the actual (fixed) number of problem variables.

The symbol(s) after the third hyphen indicate the number of constraints (other than fixed variables and bounds) in the problem. Note that fixed variables are not considered as general constraints here. The two possible values are

V the number of constraints in the problem can be chosen by the user, or

m a nonnegative integer giving the actual (fixed) number of problem constraints.

In the future, we intend to expand the scope of the classification to include, amongst other things, a more thorough subdivision of constraint information, structural considerations, and solution characteristics. We have refrained from extending the classification system at the present time as we wish to benefit from feedback from CUTE users. It is, of course, essential that any such extensions should be compatible with the present system."

2.3 The `select` tool

The purpose of this tool is to interrogate a file containing details on a complete list of problem classifications. In the distributed version of CUTE, which contains all the current classification details for the test set used by LANCELOT, we call this file `CLASSF.DB`. The interrogation includes a facility for choosing the file that is to be probed. Thus one is able to maintain and query several different .DB files and obtain a list of problems matching interactively defined characteristics. The list can be automatically saved in a designated file.

One might imagine having several different problem datasets in several different directories. Suppose, for example, one had two directories called, say, academic and applications. One could then maintain a .DB file in each directory corresponding to the SIF files they contained. For each, there is then the possibility of setting up an interactive dialogue at the terminal to determine such data as, for example, 'which problems are constrained and have only linear constraints', or 'which problems have more than 1000 constraints and have explicit second derivatives'? Note, however, that the user must be pointing to the directory in which the .DB file to be probed resides. (In a UNIX environment, for example, this means defining the environment variable `MASTSIF` appropriately.) If the automatic save facility is used, the listing will be saved in this directory.

The dialogue with the user is on the standard input/output. Additional information about the `select` tool is given in the document file `install.rdm` contained in the directory `$CUTEDIR/doc` (see Section 7).

Note that the upper bound on the number of variables is 99,999,999 and one is not allowed more than 99,999,999 constraints.

The output produced when running this program is meant to be self-explanatory. A typical session is given in Appendix C.

3 Adding new problems

Large-scale optimization problems (fortunately) typically contain a great deal of information in their structure. Our input format was designed to satisfy a number of objectives, including exploiting that structure. In particular, we wanted a format that would be available to anyone using a machine that has a Fortran compiler; that was capable of exploiting the problem structure; and that would be compatible with the MPS format of International Business Machines Corporation (1978). When users wish to add new problems, they will first need to write the appropriate input files. It is not our intention to provide a manual for writing such files here. Details of the required syntax and a primer for beginners are included in the book describing the LANCELOT software package (Conn *et al.* (1992a), Chapters 7 and 2 respectively).

Although this paper does not describe how to write problems in SIF, Section 3.1 outlines how the problem structure is exploited, and Appendix A gives a small illustrative example of a SIF file. The user might notice the classification string in this example. In general the classification string should occur in the SIF file before the `GROUPS` or `VARIABLES` section and must be of the form

```
* classification XXXr-XX-n-m
```

or

```
* CLASSIFICATION XXXr-XX-n-m
```

where the first character must be a `*`, any number of blanks can appear before or after the keyword `classification` or `CLASSIFICATION` as long as the entire string is no more than eighty characters, and the characters in the string `XXXr-XX-n-m` must obey the classification scheme defined in Section 2.2.

3.1 Decoding the problem-input file

A module (called the 'SIF decoder') reads the description of the problems in the standard input format. A structure called group partial separability, that we believe has many significant advantages, is accounted for in the input format.

A function $f(x)$ is said to be *group partially separable* if:

1. the function can be expressed in the form
$$f(x) = \sum_{i=1}^{n_g} g_i(\alpha_i(x)), \tag{3.1}$$
where n_g is the number of groups;

2. each of the *group functions* $g_i(\alpha)$ is a twice continuously differentiable function of the single variable α;

3. the function
$$\alpha_i(x) = \sum_{j \in \mathcal{J}_i} w_{i,j} f_j(x^{[j]}) + a_i^T x - b_i \tag{3.2}$$

is known as the i-th *group*;

4. each of the index sets $\mathcal{J}_i$ is a subset of $\{1, \ldots, n_e\}$, where n_e is the number of different nonlinear element functions;

5. each of the *nonlinear element functions* f_j is a twice continuously differentiable function of a subset $x^{[j]}$ of the variables x. Each function is assumed to have a large invariant subspace. Usually, this is manifested by $x^{[j]}$ comprising a small fraction of the variables x;

6. the gradient a_i of each of the *linear element functions* $a_i^T x - b_i$ is, in general, sparse; and

7. the $w_{i,j}$ are known as element *weights*.

This structure is extremely general. Indeed, any function with a continuous, sparse Hessian matrix may be written in this form (see Griewank and Toint (1982)). A more thorough introduction to group partial separability is given by Conn *et al.* (1990). The SIF decoder assumes that the objective and general constraint functions are of this form, with the proviso that each constraint uses only a single group (that is, $n_g = 1$ for the constraints).

Thus for each problem in the SIF test set there is a file, named for example EXAMPLE.SIF. The SIF decoder interprets the statements found in the file and produces several Fortran subroutines and a data file as given below:

`ELFUNS.f` contains a Fortran subroutine that evaluates the numerical functions corresponding to the nonlinear element function types occurring in the problem, as well as their derivatives;

`GROUPS.f` contains a Fortran subroutine that evaluates the numerical functions corresponding to the group function types occurring in the problem, as well as their derivatives;

`RANGES.f` contains a Fortran subroutine that computes the transformations from elemental to internal variables (see Conn *et al.* (1992a) or Conn *et al.* (1990)) for all element types which use a nontrivial transformation;

`SETTYP.f` contains a Fortran subroutine that assigns the correct type to the nonlinear elements for the problem;

`EXTERN.f` (if present) contains any user supplied Fortran functions found at the end of the problem SIF file; and

`OUTSDIF.d` contains data on the problem structure (variable and constraint names, scalings, starting point and the like).

These files are subsequently used by the tools that interface with other optimization codes (see Sections 5 and 6) and of course, they are required by the LANCELOT optimizer.

4 Management of classification databases

Once the problem is described in SIF, the next step is its inclusion in the problems database. This raises the issue of the database management. We now decribe the tools `classify` and `classall` required for this management: the purpose of these tools is to create, maintain and update .DB files that contain the classification database that the `select` tool interrogates. The dialogue with the user is on the standard input/output. We now illustrate how the classification commands can be used.

Suppose the directory pointed to by the environment variable `MASTSIF` (assuming one is in a UNIX environment) contains the SIF files

```
BRYBND.SIF     EQC.SIF        LEAKNET.SIF     NET3.SIF
CLUSTER.SIF    FLETCHBV.SIF   MINMAXBD.SIF    QC.SIF
CORE1.SIF      GENROSE.SIF    MOREBV.SIF      S268.SIF
CORE2.SIF      HS25.SIF       NET1.SIF        S368.SIF
COSHFUN.SIF    HS35.SIF       NET2.SIF        STANCMIN.SIF
```

that all contain suitable classification lines. Issuing the command

```
classify LEAKNET
```

(note that the filename is case sensitive) results in the following output at the screen

```
Your current classification file is : CLASSF.DB
Your current SIF filename file is : LEAKNET.SIF
```

If the file `CLASSF.DB` in the directory pointed to by the environment variable `MASTSIF` does not already exist, then it is created and contains the line

```
LEAKNET  LOR2-RN-  156-  153
```

Otherwise, if `CLASSF.DB` exists but there is no entry for problem LEAKNET, the classification line is inserted in the file in its correct ASCII ordered position. If LEAKNET already appears in `CLASSF.DB`, the user is asked if he wishes to overwrite the old entry, if it differs from the new one.

To refer to a directory other than the directory indicated by `$MASTSIF`, one must issue the command

`classify directory name`,

where `name` is the problem name with or without the .SIF appended and `directory` is the name of the directory where this problem resides. In this case, `CLASSF.DB` is created in the directory `directory`. Note that when the `directory` argument is not given, `classify` assumes that the SIF file `name`.SIF is in the `$MASTSIF` directory. It is recommended that the user not assign filenames with multiple .SIF endings.

Finally issuing the command

```
classall
```

produces the file CLASSF.DB in the appropriate directory (pointed to by the environment variable MASTSIF in a UNIX environment), together with screen messages similar to that produced by classify for each SIF file processed. The process is completed by a listing of the .DB file on the screen. For our example, classall produces a CLASSF.DB file containing the following lines:

```
BRYBND   SUR2-AN-    V-    V
CLUSTER  NOR2-AN-    2-    2
CORE1    LQI2-RN-   65-   59
CORE2    LQI2-RN-  157-  134
COSHFUN  LOR2-AN-    V-    V
EQC      OUR2-MY-    9-    0
FLETCHBV OUR2-AN-    V-    V
GENROSE  SUR2-AN-    V-    V
HS25     SUR2-AN-    3-    0
HS35     QLR2-AN-    3-    1
LEAKNET  LOR2-RN-  156-  153
MINMAXBD LOR2-AN-    5-   20
MOREBV   SUR2-MY-    V-    V
NET1     OOI2-RN-   67-   57
NET2     OOI2-RN-  181-  160
NET3     OOI2-RN-  591-  521
QC       OUR2-MY-    9-    0
S268     QLR2-AN-    5-    5
S368     OBR2-MY-    V-    V
STANCMIN OLI2-AY-    3-    2
```

To override the environment variable MASTSIF, one must issue the command

```
classall directory
```

where directory is the name of the directory containing the SIF files to be classified. In this case, CLASSF.DB is created in the directory directory.

5 The interface with existing optimization packages

So far we have described what problems exist in SIF, how they are classified and how to add to a supplied or new database. This section lists the optimization packages for which interfaces have been developed. The purpose of these interfaces is to deal with the situation, presumably not uncommon amongst researchers and practioners alike, that one wants to run a given problem on a range of algorithms to assess which algorithm is likely to be the most suitable for solving classes of related problems.

5.1 Scope

At the present time, we have available interfaces with the following optimization packages:

- **MINOS of Murtagh and Saunders (1987)**
  MINOS solves problems of the form

$$
\begin{array}{lll}
\underset{x \in \mathbf{R}^n, y \in \mathbf{R}^m}{\text{minimize}} & F(x) + c^T x + d^T y & \\
\text{subject to} & f(x) + A_1 y = b_1 & \\
 & A_2 x \ + A_3 y = b_2 & \quad (5.3) \\
\text{and} & l_x \le x \le u_x & \\
 & l_y \le y \le u_y. &
\end{array}
$$

The nonlinear contributions to the constraints are linearized so that linear programming technology can be exploited. MINOS allows matrices to be stored in either dense or sparse format, and is therefore suitable for large sparse problems. Details are given in Murtagh and Saunders (1978) and Murtagh and Saunders (1987). We currently have interfaces for MINOS 5.3 and MINOS 5.4. MINOS is distributed by the Office of Technology Licensing (OTL) at Stanford University and is subject to certain license agreements. MINOS is copyrighted by Stanford University. Readers interested in more details should contact Michael Saunders (email address: mike@sol-michael.stanford.edu, postal address: Department of Operations Research, Stanford, CA 94305-4022, USA).

- **NPSOL of Gill *et al.* (1986)**
  This package is designed to minimize smooth functions subject to constraints, which may include simple bounds, linear constraints, and smooth nonlinear constraints. The software uses a sequential quadratic programming algorithm, where bounds, linear constraints and nonlinear constraints are treated separately. Unlike MINOS, NPSOL stores all matrices in dense format, and is therefore not intended for large sparse problems. NPSOL is available from the Office of Technology Licensing at Stanford University.

- **OSL of International Business Machines Corporation (1990)**
  This package obtains solutions to quadratic programming problems where the Hessian matrix is assumed positive-semidefinite. It is intended to be suitable for large-scale problems. OSL is distributed by International Business Machines, subject to certain license agreements, and is copyrighted by IBM Corporation.

- **TENMIN of Schnabel and Chow (1991)**
  This package is intended for problems where the cost of storing one n by n matrix (where n is the number of variables), and factoring it at each iteration, is acceptable. The software allows the user to choose between a tensor method for unconstrained optimization, and an analogous standard method based upon a quadratic model. The tensor method bases each iteration upon a specially constructed fourth-order model of the objective function that is not significantly more expensive to form, store, or solve than the standard quadratic model. TENMIN is available via anonymous ftp from ftp.cs.colorado.edu, in the directory pub/cs/distribs/tensor. Any questions about this software should be addressed to: eskow@cs.colorado.edu

- **UNCMIN of Koontz *et al.* (1985) that corresponds closely to the pseudocode in Dennis and Schnabel (1983)**

This package is designed for unconstrained minimization and has options that include both line search and trust region approaches. The provided options include analytic gradients or difference approximations with analytic Hessians or finite difference Hessians (from analytic or finite difference gradients) or secant methods (BFGS).

- **VA15 of Liu and Nocedal (1989)**
  This package solves general nonlinear unconstrained problems using a limited memory BFGS method. It is intended for large-scale problems.

- **VE09 of Gould (1991)**
  This package obtains local solutions to general, non-convex quadratic programming problems, using an active set method, and is intended to be suitable for large-scale problems.

- **VE14 of Conn *et al.* (1993b)**
  This package solves bound-constrained quadratic programming problems using a barrier function method and is again intended to be suitable for large-scale problems.

- **VF13 of Powell (1982)**
  This package solves general nonlinearly constrained problems using a sequential quadratic programming technique.

  VA15, VE09, VE14 and VF13 are part of the Harwell Subroutine Library (1993). They are distributed by the United Kingdom Atomic Energy Authority, Harwell, subject to certain license agreements. They are all copyrighted jointly by the UKAEA and SERC (Science and Engineering Research Council).

Of course, LANCELOT also solves problems in SIF, but it does not require an interface using the CUTE tools. It is worth noting that LANCELOT exploits much more structure than that exploited by the interface tools.

The natural next question is how does one use the supplied interfaces. We begin with one of the simplest.

5.2 Using the UNCMIN interface

After installation of CUTE (see Section 7 below), for example in a UNIX environment, there will be an empty subdirectory named `uncmin`. It is the user's responsibility to ensure that the UNCMIN source codes are compiled into a single object code (`uncmins.o`) in the `uncmin` subdirectory. (Although UNCMIN is available in single precision only, the user may wish to create a double precision version. The double precision object code should be stored as `uncmind.o` in the `uncmin` subdirectory. To run this double precision version, the user will have to edit the scripts `sdunc` and `unc`. These scripts include comments describing the required changes.)

For illustrative purposes, suppose one is in a UNIX environment. Assume that the environment variables `CUTEDIR` and `MASTSIF` are set appropriately, and that `uncmins.o` is in the directory `$CUTEDIR/uncmin`. Now suppose further that one wishes to solve the problem MOREBV whose SIF file MOREBV.SIF is in the directory `$MASTSIF`. The user should either set an alias to the command `sdunc` using

```
alias sdunc '$CUTEDIR/interfaces/sdunc'
```

(typically, this alias would be set automatically by the user's shell) or add `$CUTEDIR/interfaces` to his/her path. The user should then enter the directory `$MASTSIF` and issue the command

```
sdunc -s MOREBV
```

(The `-s` option is explained in Section 5.3.) This command results in the following output:

```
 Problem name: MOREBV

 Single precision version will be formed.

 The objective function uses     10 nonlinear groups

 There are     10 free variables

 #FCN EVAL   =         13
 GRAD EVAL   =         13
 HSN  EVAL   =          2
 ITRMCD      =          1
 FINAL F     =   6.5557D-14
 FINAL NORMG =   1.3728D-07

                 X          G
X1          -3.5088D-02 -1.5189D-08
X2          -6.5312D-02 -4.8354D-09
X3          -8.9785D-02  2.4506D-08
X4          -1.0742D-01  9.2885D-08
X5          -1.1686D-01 -1.4974D-09
X6          -1.1641D-01  7.4553D-08
X7          -1.0390D-01 -9.9286D-09
X8          -7.6520D-02 -4.0877D-08
X9          -3.0551D-02  1.3728D-07
X10          3.9023D-02  1.2241D-07

 Set up time =         0.05
  Solve time =         0.02
  Total time =         0.07 seconds
```

Further details for both this interface and the others provided with the package are given in the document file `install.rdm` contained in the directory `$CUTEDIR/doc` and in Section 5.3. A summary of the available packages and the corresponding commands is given in Figure 3. LANCELOT uses the commands `sdlan` and `lan`, which perform the same functions and have the same syntax (described in Section 5.3) as the commands listed in Figure 3. The `sdlan` and `lan` scripts are distributed as part of LANCELOT and not as part of CUTE.

A few SIF files, as indicated in Figure 4, are provided in the directory `$CUTEDIR/problems`. These are sufficient to verify that the packages are correctly installed.

Package	Decode and optimize	Optimize only
MINOS	`sdmns`	`mns`
NPSOL	`sdnps`	`nps`
OSL	`sdosl`	`osl`
TENMIN	`sdten`	`ten`
UNCMIN	`sdunc`	`unc`
VA15	`sdlmq`	`lmq`
VE09	`sdqp`	`qp`
VF13	`sdcon`	`con`
VE14	`sdbqp`	`bqp`

Figure 3: Currently available interfaces.

Package	Appropriate SIF file
MINOS	HS11
NPSOL	HS11
OSL	HS21
TENMIN	ROSENBR
UNCMIN	ROSENBR
VA15	ROSENBR
VE09	HS21
VF13	HS11
VE14	HS3

Figure 4: Interface and appropriate sample SIF file

5.3 The generic interface

All of the shell scripts for the interfaces listed in Figure 3 execute the same general steps and have the same syntax. Thus, we now give details on the usage of these scripts by describing the `sdgen` and `gen` commands for a generic interface. The characters `gen` in `sdgen` and `gen` may be replaced by `mns`, `unc`, or any of the other interface names given in the third column in Figure 3.

Besides offering a convenient means to describe details on the usage of the current interfaces, the `sdgen` and `gen` shell scripts serve another important purpose: they provide a template for the shell scripts required for new interfaces with optimization packages other than those listed in Figure 3. Details on the construction of a new interface are given in the document file `interface.rdm` contained in the directory `$CUTEDIR/doc`.

5.3.1 The `sdgen` and `gen` commands

Suppose the scripts `sdgen` and `gen`, residing in the directory `$CUTEDIR/interfaces`, provide an interface with the optimization package GEN (for GENeric). The object module(s) for the GEN package reside in the directory `$CUTEDIR/gen`. The single precision object is named `gens.o`, while the double precision object is named `gend.o`. (Because some optimization packages consist of only one precision source code, the user may wish to modify this source code to create an appropriate precision version.) Besides the shell scripts, the interface also includes a Fortran driver `genma.f` which interleaves calls to the optimization subroutines in GEN with calls to the appropriate CUTE evaluation tools (described in Section 6 and Appendix B). The Fortran source file `genma.f` resides in `$CUTEDIR/tools/sources`, while the compiled object modules reside in `$CUTEDIR/tools/objects/single` and `$CUTEDIR/tools/objects/double`, for the single and double precision versions, respectively.

The main steps executed by the `sdgen` command are as follows:

1. Check the argument of the command for inconsistencies and interpret them. Also check that the problem specified has an associated SIF file.

2. Apply the SIF decoder to the problem SIF file, in order to produce the `OUTSDIF.d` file and the problem dependent Fortran subroutines. Stop the process if any error is uncovered in the SIF file.

3. Call the `gen` command (described next) in order to continue the process.

The main steps executed by the `gen` command are as follows:

1. Check the arguments of the command for inconsistencies and interpret them.

2. If a new executable module is to be built, compile the problem dependent Fortran subroutines.

3. Load together the compiled problem dependent subroutines, the appropriate evaluation tool objects, the Fortran driver object `genma.o`, and the appropriate GEN object module (`gens.o` or `gend.o`, depending on whether the single or double precision version is selected) to produce the executable file `genmin`.

4. Execute `genmin` in order to solve the problem.

5.3.2 Syntax and options for the sdgen command

The format of the command is

```
sdgen [-s] [-h] [-k] [-o j] [-l secs] probname
```

where optional arguments are within square brackets. The command arguments have the following meaning:

`[-s]` runs the single precision version (default: run the double precision version),

`[-h]` prints a simple description of the possible options for the `sdgen` command,

`[-k]` does not delete the executable module after execution (default: the module is deleted),

`[-o j]` if j=0, the tool runs in silent mode, while brief descriptions of the stages executed are printed if j = 1 (default: silent mode),

[-l secs] an upper limit of secs seconds is set on the CPU time used by GEN in the numerical solution of the problem (default: 99,999,999 seconds),

probname is the name (without extension) of the file containing the SIF description of the problem to solve.

5.3.3 Syntax and options for the gen command

The format of the command is

```
gen [-n] [-h] [-s] [-k] [-o i] [-l secs]
```

where optional arguments are within square brackets. The command arguments have the following meaning:

[-n] reconstructs the executable module genmin from the files output at the SIF decoding stage (default: run the current executable module without reconstructing it),

[-h] prints a simple description of the possible options for the gen command,

[-s] runs the single precision version (default: run the double precision version),

[-k] does not delete the executable module after execution (default: the module is deleted),

[-o j] if j=0, the tool runs in silent mode, while brief descriptions of the stages executed are printed if j = 1 (default: silent mode),

[-l secs] an upper limit of secs seconds is set on the CPU time used by GEN in the numerical solution of the problem (default: 99,999,999 seconds).

5.3.4 Running GEN on a problem in standard input format

In order to be able to run sdgen and gen from any directory, the user should add the lines

```
alias sdgen '$CUTEDIR/interfaces/sdgen'
alias gen '$CUTEDIR/interfaces/gen'
```

to his/her .cshrc file if running the C shell under AIX, UNICOS, SUN SunOS or Ultrix systems. Alternatively, the user could add $CUTEDIR/interfaces to his/her PATH.

To run GEN on a problem in SIF, the user should move into the directory in which the SIF file for the problem, probname.SIF, resides and issue the command

sdgen probname

This command will decode the SIF file, print a short summary of the problem characteristics, and attempt to solve the problem using GEN.

If the SIF file has already been decoded but the executable module genmin has been deleted, the appropriate command is

```
gen -n
```

while it is

```
gen
```

if the SIF file has already been decoded and the executable module `genmin` still exists. (To keep the executable module after solving the problem, the `-k` option should be used with either the `sdgen` or `gen` command.)

6 Evaluation tools

6.1 Overview

We also provide a collection of tools that enable one to manipulate data once it has been decoded from a SIF file. This is important for two main reasons. Firstly, the data structure is able to represent structure that is more general than sparsity (see Section 3.1 above). Naturally this capability imposes some overhead with respect to the complexity of the data. The evaluation tools enable one to reverse the process — in other words, the tools take a group partially separable format (that one might think of as a 'scatter' operation) and 'gather' the information to produce the function value, the gradient or the second derivatives for the entire function. In addition we include a routine that forms the matrix vector product with the Hessian matrix and we provide utilities to obtain the names of a problem, its variables and possibly its constraints. Secondly, our testing of LANCELOT required the collection of a substantial number of large and non-trivial optimization problems (see Section 2.1 above). With the aid of the supplied tools one can use the same input files to interface with other optimization software that is not designed to accept SIF (see below). As a corollary, we are hoping that practitioners will be encouraged to write test problems in the SIF. Of course, although not strictly speaking essential, (see Chapter 8 of Conn *et al.* (1992a)), we would assume that most users of LANCELOT will write their input using the SIF.

Several parameters have values that may be changed by the user to suit a particular problem or system configuration. If any of these are too small, an error message is produced telling which parameter(s) to increase. Typically they concern the size of the workspace for what we consider the default values for big, medium and small problems. Details on how to do this are given in the document file `install.rdm` contained in the directory `$CUTEDIR/doc`.

The next two sections indicate the problem classes for which tools are available for users to manipulate the decoded data.

6.2 The unconstrained and bound constrained problem

We provide tools that specifically relate to the bound constrained problem:

$$\text{optimize } f(x) \tag{6.4}$$

subject to the simple bounds

$$l_i \leq x_i \leq u_i, \quad (i = 1, \ldots, n), \tag{6.5}$$

where $x \in \mathbf{R}^n$, f is assumed to be a smooth function from $\mathbf{R}^n$ into $\mathbf{R}$ and is group partially separable. Of course, if all the bounds are infinite, then the problem is unconstrained.

A summary of the available subroutines and their purpose is given in Figure 5. A more detailed description is given in Appendix B.

6.3 The general constrained problem

We also provide tools that specifically relate to the general constrained problem

$$\text{optimize } f(x) \tag{6.6}$$

subject to the general equations

$$c_i(x) = 0, \quad i \in E, \tag{6.7}$$

the general inequality constraints

$$(c_l)_i \leq c_i(x) \leq (c_u)_i, \quad i \in I, \tag{6.8}$$

and the simple bounds

$$l_i \leq x_i \leq u_i, \quad (i = 1, \ldots, n). \tag{6.9}$$

Here $x \in \mathbf{R}^n$, f, c_i are assumed to be smooth functions from $\mathbf{R}^n$ into $\mathbf{R}$ and are group partially separable. Furthermore $I \cup E = 1, 2, ..., m$, with I and E disjoint.

Associated with the above problem is the so-called Lagrangian function

$$L(x, \lambda) = f(x) + \lambda^T c(x), \tag{6.10}$$

where $c(x)$ is the vector whose i-th component is $c_i(x)$ and the components of the vector λ are known as Lagrange multipliers.

The various subroutines, many of which are similar to those supplied for the problem with simple bounds above, are now summarized in Figure 5. A more detailed description is given in Appendix B.

7 Installing CUTE on your system

The `CUTE` distribution consists of a set of files. All these files should be placed in a single directory, from which the installation will be performed. This directory will henceforth be called the 'CUTE directory' and we assume that in a UNIX environment this is pointed to by the `CUTEDIR` environment variable. The installation will, amongst other things, create a subdirectory structure suitable for the machine on which the package is being installed. This structure is shown in Figure 6.

In addition, there are directories created at the same level as the tools which are meant to contain the single and double precision object modules required for the various interfaces. Since these depend upon the user having the right to use these particular modules they are not, of course, included in the distribution. Currently the directories are `uncmin`, `conmin`, `minos`, `npsol`, `qp`, and `tenmin`. The directories `minos` and `npsol` contain default specifications files for MINOS

Purpose	Tool name	Sparse, Dense version (when applicable)
Tools for unconstrained problems		
Set up the correct data structure	USETUP	
Evaluate the objective function value	UFN	
Evaluate the objective function and possibly its gradient	UOFG	
Evaluate the objective function gradient	UGR	Dense
Evaluate the objective function Hessian	UDH	Dense
Evaluate the objective function gradient and Hessian	UGRDH	Dense
Evaluate the objective function Hessian	USH	Sparse
Evaluate the objective function gradient and Hessian	UGRSH	Sparse
Evaluate the matrix-vector product of a vector with the Hessian	UPROD	
Obtain the elements of the Hessian matrix which lie within a band of given semi-bandwidth	UBANDH	
Obtain the names of the problem and variables	UNAMES	
Tools for constrained problems		
Set up the correct data structure	CSETUP	
Evaluate the objective function and general constraint values	CFN	
Evaluate the objective function and possibly its gradient	COFG	
Evaluate the general constraint functions and possibly their gradients	CCFG	Dense
Evaluate the general constraint functions and possibly their gradients	CSCFG	Sparse
Evaluate the general constraint gradients and either the objective function or Lagrangian gradient	CGR	Dense
Evaluate the general constraint gradients and either the objective function or Lagrangian gradient	CSGR	Sparse
Evaluate the Lagrangian function Hessian	CDH	Dense
Evaluate the objective function or Lagrangian gradient, gradients of general constraints and Lagrangian Hessian	CGRDH	Dense
Evaluate the Lagrangian function Hessian	CSH	Sparse
Evaluate the objective function or Lagrangian gradient, gradients of general constraints and Lagrangian Hessian	CSGRSH	Sparse
Evaluate the matrix-vector product of a vector with the Hessian	CPROD	Dense
Obtain the names of the problem, its variables and general constraints	CNAMES	

Figure 5: Available tools.

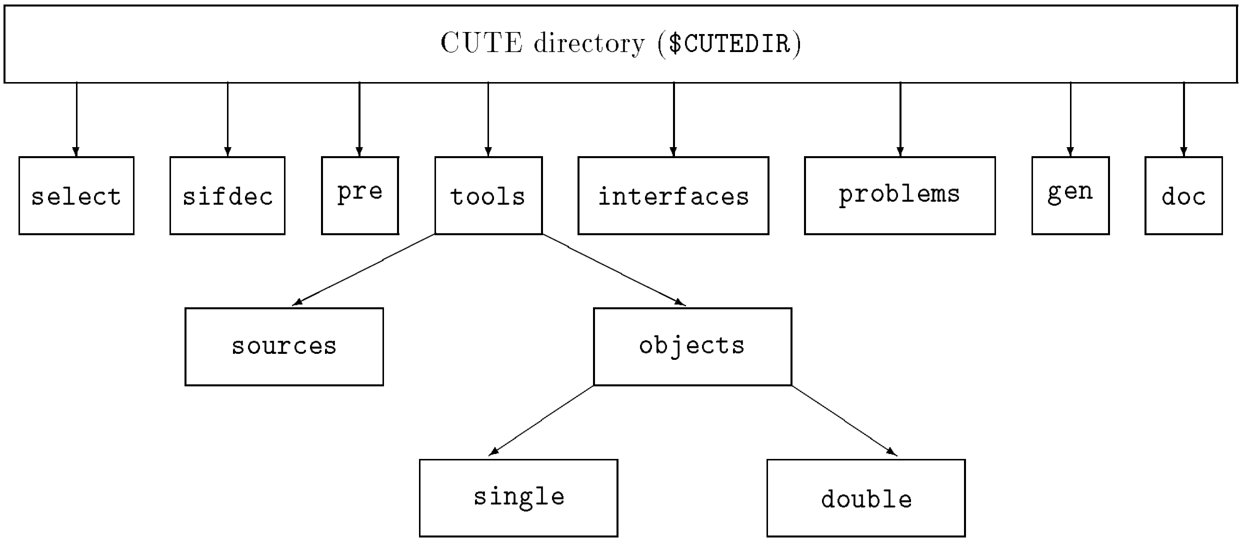

Figure 6: Organization of the CUTE directories

and NPSOL, named `MINOS.SPC` and `NPSOL.SPC`, respectively. Each of the directories `minos`, `npsol`, and `tenmin` also contains a `READ.ME` to explain how to create the required object module.

The CUTE package is available with automated installation procedures for the following range of operating systems:

- Cray: UNICOS,
- DEC: Ultrix, VMS (using "G-floating" double precision) and OSF/1,
- IBM: AIX,
- SUN: SunOS.

If your system is different from those listed above and if you nevertheless wish to install CUTE, we include some details that should be useful in the document file `install.rdm` contained in the directory `$CUTEDIR/doc`. Further details, including installation on several preassigned systems, preparation of the environment, running of scripts and individual tailoring and installation of unsupported systems are given in the document files `install.rdm` and `interface.rdm` in the directory `$CUTEDIR/doc`.

8 Obtaining the CUTE distribution

CUTE is written is standard ANSI Fortran 77. Single and double precision versions are available. Machine dependencies are carefully isolated and easily adaptable. Automatic installation procedures are available for DEC Ultrix, VMS, and OSF/1; SUN SunOS; CRAY UNICOS; and IBM AIX (see Section 7).

The package may be obtained in one of two ways. Firstly, the reader can obtain CUTE electronically via an anonymous ftp call to the account at the Rutherford Appleton Laboratory *camelot.cc.rl.ac.uk* (Internet i.d. 130.246.8.61, in directory pub/cute), or at Facultés Universitaires Notre-Dame de la Paix (Namur) *thales.math.fundp.ac.be* (Internet i.d. 138.48.4.14, in directory cute). We request that the userid be given as the password. This will serve to identify those who have obtained a copy via ftp.

Secondly, the package can be obtained at minimal cost on floppy disk or magnetic tape. The price covers the costs of media, packaging, preparation and courier delivery. To get a magnetic media order form, the reader should contact Ph. L. Toint (see title page for addresses).

Updates to the test problem set, since the initial release of the package, are given in the file `mastsif.updates`. Corrections and additions to the tools are documented in the file `updates.rdm` in the directory `$CUTEDIR/doc`.

9 Conclusions

We have presented in this paper the scope of the Constrained and Unconstrained Testing Environment (CUTE), and the variety of tools to maintain it, as well as the facility to interface with existing and future optimization packages. The main purpose of CUTE is to

- provide a way to explore an extensive collection of problems,
- provide a way to compare existing packages,
- provide a way to use a large test problem collection with new packages,
- provide motivation for building a meaningful set of new interesting test problems,
- provide ways to manage and update the system efficiently, and
- do all the above on a variety of popular platforms.

The only way to really judge the effectiveness of the environment is for the reader to use it. We are most interested in learning of others' experiences as this will undoubtedly be helpful in making improvements.

10 Acknowledgements

Finally, we would like to acknowledge the support we have received from the granting agencies DARPA, FNRS and NATO and to thank Jorge Nocedal and one anonymous referee for their comments on this paper. We are indebted to our colleagues D. Jensen, A. Sartenaer and M. Bierlaire, to Ali Bouaricha for his work on the TENMIN interface, to Peihuang Chen for her work on the NPSOL interface, and to the problem contributors, past, present and future.

Appendix

A Illustrative SIF file

A SIF file for the simple problem

$$\min_{x,y \in \Re} e^{(x-3y)} \tag{A.11}$$

subject to the constraint

$$\sin(y - x - 1) = 0 \tag{A.12}$$

and the bounds

$$-2 \leq x \leq 2 \;\text{ and }\; -1.5 \leq y \leq 1.5. \tag{A.13}$$

could be specified as follows:

```
NAME          EXAMPLE
*   A simple constrained  problem
*   classification OOR2-AN-2-1
VARIABLES
    x
    y
GROUPS
*   Linear terms for the objective function
 XN Object    x          1.0            y         -3.0
*   Linear terms for the general constraint
 XE Constr    y          1.0            x         -1.0
CONSTANTS
*   Constant term for the linear part of the general constraint
 X  EXAMPLE   Constr     1.0
BOUNDS
*   Lower and upper bounds on the variable x
 LO EXAMPLE   x         -2.0
 UP EXAMPLE   x          2.0
*   Lower and upper bounds on the variable y
 LO EXAMPLE   y         -1.5
 UP EXAMPLE   y          1.5
GROUP TYPE
*   Create an 'expontential' group type for the objective function
 GV EXPN      ALPHA
*   Create an 'sine' group type for the constraint function
 GV SINE      ALPHA
GROUP USES
*   Ensure that the exponential group is associated with the objective function
 XT Object    EXPN
*   Ensure that the sine group is associated with the constraint function
 XT Constr    SINE
ENDATA
```

```
GROUPS        EXAMPLE
TEMPORARIES
*   Define temporary variables
 R  EXPA
 R  SINA
*   Declare the machine dependent functions that are used
 M  EXP
 M  SIN
 M  COS
INDIVIDUALS
*   Exponential group type
 T  EXPN
 A  EXPA                EXP( ALPHA )
 F                      EXPA
 G                      EXPA
 H                      EXPA
*   Sine group type
 T  SINE
 A  SINA                SIN( ALPHA )
 F                      SINA
 G                      COS( ALPHA )
 H                      - SINA
ENDATA
```

B Details on the provided evaluation tools

We now give the complete argument lists for the subroutines summarized in Figure 5. There are two sets of tools: one set for unconstrained and bound constrained problems, and one set for generally constrained problems. Note that these two sets of tools cannot be mixed.

The superscript i on an argument means that the argument must be set on input. A superscript o means that the argument is set by the subroutine.

B.1 Unconstrained and bound constrained problems

- Set up the correct data structures for subsequent computations

  CALL USETUP (INPUT^i, IOUT^i, N^o, X^o, BL^o, BU^o, NMAX^i)

  Note that a call to USETUP must precede calls to other evaluation tools for unconstrained and bound constrained problems.

- Obtain the names of the problem and its variables

  CALL UNAMES (N^i, PNAME^o, XNAMES^o)

- Evaluate the function value of the objective function

  CALL UFN (N^i, X^i, F^o)

- Evaluate the gradient of the objective function
  CALL UGR (N^i, X^i, G^o)

- Evaluate the value of the objective function and possibly its gradient
  CALL UOFG (N^i, X^i, F^o, G^o, GRAD^i)

  Note that calling UOFG is more efficient than separate calls to UFN and UGR.

- Evaluate the Hessian matrix of a objective function (when stored as a dense matrix)
  CALL UDH (N^i, X^i, LH1^i, H^o)

- Evaluate both the gradient and Hessian matrix of the objective function (when stored as a dense matrix)
  CALL UGRDH (N^i, X^i, G^o, LH1^i, H^o)

  Note that calling UGRDH is more efficient than separate calls to UGR and UDH.

- Evaluate the Hessian matrix of the objective function (when stored as a sparse matrix)
  CALL USH (N^i, X^i, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o)

- Evaluate both the gradient and Hessian matrix of the objective function
  CALL UGRSH (N^i, X^i, G^o, NNZSH^o, LSH^i, SH^o, IRNSH^o, ICNSH^o)

  Note that calling UGRSH is more efficient than separate calls to UGR and USH.

- Form the product of a vector with the Hessian matrix
  CALL UPROD (N^i, GOTH^i, X^i, P^i, Q^o)

- Obtain the elements of the Hessian that lie within a given semi-bandwidth of its diagonal
  CALL UBANDH (N^i, GOTH^i, X^i, NSEMIB^i, BANDH^o, LBANDH^i)

B.2 Generally constrained problems

- Set up the correct data structures for subsequent computations
  CALL CSETUP (INPUT^i, IOUT^i, N^o, M^o, X^o, BL^o, BU^o, NMAX^i, EQUATN^o, LINEAR^o,
  V^o, CL^o, CU^o, MMAX^i, EFIRST^i, LFIRST^i, NVFRST^i)

  Note that a call to CSETUP must precede calls to other evaluation tools for generally constrained problems.

- Obtain the names of the problem, its variables and general constraints
  CALL CNAMES (N^i, M^i, PNAME^o, XNAMES^o, GNAMES^o)

- Evaluate the objective and general constraint function values
  CALL CFN (N^i, M^i, X^i, F^o, LC^i, C^o)

- Evaluate the gradients of the general constraint functions
  CALL CGR (N^i, M^i, X^i, GRLAGF^i, LV^i, V^i, G^o, JTRANS^i, LCJAC1^i, LCJAC2^i, CJAC^o)

- Evaluate the value of the objective function and possibly its gradient
  `CALL COFG ( N`^{i}`, X`^{i}`, F`^{o}`, G`^{o}`, GRAD`^{i}` )`

  Note that calling `COFG` is more efficient than separate calls to `CFN` and `CGR`.

- Evaluate the gradients of the general constraint functions (when these are stored in a sparse format)
  `CALL CSGR ( N`^{i}`, M`^{i}`, GRLAGF`^{i}`, LV`^{i}`, V`^{i}`, X`^{i}`, NNZSCJ`^{o}`, LSCJAC`^{i}`, SCJAC`^{o}`, INDVAR`^{o}`, INDFUN`^{o}` )`

- Evaluate the values of the constraints functions and possibly their gradients
  `CALL CCFG ( N`^{i}`, M`^{i}`, X`^{i}`, LC`^{i}`, C`^{o}`, JTRANS`^{i}`, LCJAC1`^{i}`, LCJAC2`^{i}`, CJAC`^{o}`, GRAD`^{i}` )`

- Evaluate the values of the constraint functions and possibly their gradients (when these are stored in a sparse format)
  `CALL CSCFG ( N`^{i}`, M`^{i}`, X`^{i}`, LC`^{i}`, C`^{o}`, NNZSCJ`^{o}`, LSCJAC`^{i}`, SCJAC`^{o}`, INDVAR`^{o}`, INDFUN`^{o}`, GRAD`^{i}` )`

- Evaluate the Hessian matrix of the Lagrangian (when stored as a dense matrix)
  `CALL CDH ( N`^{i}`, M`^{i}`, X`^{i}`, LV`^{i}`, V`^{i}`, LH1`^{i}`, H`^{o}` )`

- Evaluate both the gradients of the general constraint functions and the Hessian matrix of the Lagrangian
  `CALL CGRDH ( N`^{i}`, M`^{i}`, X`^{i}`, GRLAGF`^{i}`, LV`^{i}`, V`^{i}`, G`^{o}`, JTRANS`^{i}`, LCJAC1`^{i}`, LCJAC2`^{i}`, CJAC`^{o}`, LH1`^{i}`, H`^{o}` )`

  Note that calling `CGRDH` is more efficient than separate calls to `CGR` and `CDH`.

- Evaluate the Hessian matrix of the Lagrangian (when stored as a sparse matrix)
  `CALL CSH ( N`^{i}`, M`^{i}`, X`^{i}`, LV`^{i}`, V`^{i}`, NNZSH`^{o}`, LSH`^{i}`, SH`^{o}`, IRNSH`^{o}`, ICNSH`^{o}` )`

- Evaluate both the gradients of the general constraint functions and the Hessian matrix of the Lagrangian
  `CALL CSGRSH ( N`^{i}`, M`^{i}`, X`^{i}`, GRLAGF`^{i}`, LV`^{i}`, V`^{i}`, NNZSCJ`^{o}`, LSCJAC`^{i}`, SCJAC`^{o}`, INDVAR`^{o}`, INDFUN`^{o}`, NNZSH`^{o}`, LSH`^{i}`, SH`^{o}`, IRNSH`^{o}`, ICNSH`^{o}` )`

  Note that calling `CSGRSH` is more efficient than separate calls to `CSGR` and `CSH`.

- Form the product of a vector with the Hessian matrix of the Lagrangian
  `CALL CPROD ( N`^{i}`, M`^{i}`, GOTH`^{i}`, X`^{i}`, LV`^{i}`, V`^{i}`, P`^{i}`, Q`^{o}` )`

B.3 Meaning of the arguments

The arguments in the above calling sequences have the following meanings:

`BANDH` is a two-dimensional array of dimension (0:`LBANDH`,`N`) which gives the lower triangular part of the band segment of the Hessian of the objective function. The diagonal entry in column i is returned in location `BANDH`$(0, i)$, while the entry j places below the diagonal in column i may be found in location `BANDH`(j, i).

`BL` is an array which gives lower bounds on the variables.

BU is an array which gives upper bounds on the variables.

C is an array which gives the values of the general constraint functions evaluated at X. The i-th component of C will contain the value of $c_i(x)$.

CJAC is a two-dimensional array of dimension (LCJAC1, LCJAC2) which gives the value of the Jacobian matrix of the constraint functions, or its transpose, evaluated at X. If JTRANS is .TRUE., the i, j-th component of the array will contain the i-th derivative of the j-th constraint function. Otherwise, if JTRANS is .FALSE., the i, j-th component of the array will contain the j-th derivative of the i-th constraint function.

CL is an array which gives lower bounds on the inequality constraints.

CU is an array which gives upper bounds on the inequality constraints.

EFIRST is a logical variable which should be set .TRUE. if the user wishes the general equations to occur before the general inequalities in the list of constraints. If the order is unimportant, EFIRST should be set .FALSE..

EQUATN is a logical array whose i-th component is .TRUE. if the i-th constraint is an equation ($i \in E$), and .FALSE. if the constraint is an inequality ($i \in I$).

F gives the value of the objective function evaluated at X.

G is an array which gives the value of the gradient of the objective function (for unconstrained problems, or for constrained problems when GRLAGF = .FALSE.) or of the Lagrangian (for constrained problems when GRLAGF = .TRUE.), evaluated at X.

GNAMES is an array of 10-character names which gives the names of the general constraints.

GOTH is a logical variable which specifies whether the second derivatives of the groups and elements have already been set (GOTH = .TRUE.) or if they should be computed (GOTH = .FALSE.). GOTH should be set to .TRUE. whenever either a call has been made either to UDH, USH, UGRDH or UGRSH or to CDH, CSH, CGRDH or CSGRSH at the current point, or whenever a previous call, with GOTH = .FALSE., has been made to UPROD, UBANDH or CPROD at the current point. Otherwise, it should be set .FALSE.

GRAD is a logical variable which should be set .TRUE. if the gradient of the objective function (in UOFG) or of the constraint functions (in CCFG and CSCFG) is required and .FALSE. otherwise.

GRLAGF is a logical variable which should be set .TRUE. if the gradient of the Lagrangian is required and .FALSE. if the gradient of the objective function is sought.

H is a two-dimensional array which contains the value of the Hessian matrix of the objective function (unconstrained problems) or of the Lagrangian (constrained problems) evaluated at X (and V, for constrained problems).

IRNSH is an array which gives the row indices of the nonzeros of the Hessian matrix SH.

ICNSH is an array which gives the column indices of the nonzeros of the Hessian matrix SH.

INPUT is the unit number for the decoded data, i.e. the unit from which OUTSDIF.d is read.

INDFUN is an array whose i-th component is the index of the problem function of which SCJAC(i) is the derivative. INDFUN(i) $= 0$ indicates the objective function whenever GRLAGF is .FALSE. or the Lagrangian when GRLAGF is .TRUE., while INDFUN(i) $= j > 0$ indicates the j^{th} general constraint function.

INDVAR is an array whose i-th component is the index of the variable with respect to which SCJAC(i) is the derivative.

IOUT is the unit number for any error messages.

JTRANS is a logical variable which should be set .TRUE. if the transpose of the constraint Jacobian is required and .FALSE. if the Jacobian itself is wanted. The Jacobian matrix is the matrix whose i-th row is the gradient of the i-th constraint function.

LBANDH is the actual declared size of the leading dimension of BANDH (with LBANDH no smaller than NSEMIB). Note that the leading component of BANDH includes the index 0 so strictly, the size of the leading dimension is LBANDH + 1.

LC is the actual declared dimension of C, with LC no smaller than M.

LCJAC1 is the actual declared size of the leading dimension of CJAC, with LCJAC1 no smaller than N if JTRANS is .TRUE. or M if JTRANS is .FALSE..

LCJAC2 is the actual declared size of the second dimension of CJAC, with LCJAC2 no smaller than M if JTRANS is .TRUE. or N if JTRANS is .FALSE..

LFIRST is a logical variable which should be set .TRUE. if the user wishes the general linear (or affine) constraints to occur before the general nonlinear ones in the list of constraints. If the order is unimportant, LFIRST should be set .FALSE. If both EFIRST and LFIRST are set .TRUE., the linear constraints will occur before the nonlinear ones. The linear constraints will be ordered so that the linear equations occur before the linear inequalities. Likewise, the nonlinear equations will appear before the nonlinear inequalities in the list of nonlinear constraints.

LH1 is the actual declared size of the leading dimension of H, with LH1 no smaller than N.

LINEAR is a logical array whose i-th component is .TRUE. if the i-th constraint is linear or affine and .FALSE. otherwise.

LSH is the actual declared dimension of SH, IRNSH and ICNSH.

LSCJAC is the actual declared dimension of SCJAC, INDVAR and INDFUN.

LV is the actual declared dimension of V.

M gives the total number of general constraints.

MMAX is the actual declared dimension of EQUATN, LINEAR, CL and CU.

N is the number of variables for the problem.

NMAX is the actual declared dimension of BL, BU, and X.

NNZSH is the number of nonzeros in SH.

NNZSCJ is the number of nonzeros in SCJAC.

NSEMIB is the required semi-bandwidth of the matrix BANDH, i.e., the number of bands directly below the diagonal of the Hessian.

NVFRST is a logical variable which should be set .TRUE. if the user wishes the nonlinear variables to occur before the linear variables. (Any variable which belongs to a nontrivial group or to a nonlinear element in a trivial group is treated as a nonlinear variable.) If the number of variables which appear nonlinearly in the objective function (say n_1) is different from the number of variables which appear nonlinearly in the constraints (say m_1), then the nonlinear variables are ordered so that the smaller set occurs first. For example, if $n_1 < m_1$, the n_1 nonlinear objective variables occur first, followed by the nonlinear Jacobian variables not belonging to the first n_1 variables, followed by the linear variables.

P is an array which contains the vector whose product with the Hessian of the objective (unconstrained problems) or of the Lagrangian (constrained problems) is required.

PNAME is a 10-character name for the problem.

Q is an array which gives the result of multiplying the Hessian of the objective (unconstrained problems) or of the Lagrangian (constrained problems) by P.

SCJAC is an array which gives the values of the nonzeros of the gradients of the objective, or Lagrangian, and general constraint functions evaluated at X and V. The i-th entry of SCJAC gives the value of the derivative with respect to variable INDVAR(i) of function INDFUN(i).

SH is an array which gives the value of the Hessian matrix of the objective function (unconstrained problems) or of the Lagrangian (constrained problems) evaluated at X. The i-th entry of SH gives the value of the nonzero in row IRNSH(i) and column ICNSH(i). Only the upper triangular part of the Hessian is stored.

V is an array which gives the current estimate of the Lagrange multipliers. V is not used if GRLAGF = .FALSE. or if GOTH = .TRUE..

X is an array which gives the current estimate of the solution of the problem. X is not used by UPROD, UBANDH and CPROD if GOTH = .TRUE..

XNAMES is an array of 10-character names which gives the names of the variables.

C A typical select session

Suppose the user is interested in finding all problems in the database in the directory pointed to via the environment variable MASTSIF for which the objective function is omitted or the objective function is a sum of squares. This is not an unrealistic situation since this may correspond to either a system of equations that the user wants to solve or a system of equalities/inequalities for which a feasible point is desired. In addition, least squares problems frequently arises from trying to solve systems of equations. Suppose further that the user has a particular collection of test-problems in mind that contain ten, one hundred or a thousand constraints. This motivates the following session (note that input can be in upper or lower case):

```
select


      *************************************************
      *                                               *
      *        CONSTRAINED AND UNCONSTRAINED          *
      *             TESTING ENVIRONMENT               *
      *                                               *
      *                  ( CUTE )                     *
      *                                               *
      *        INTERACTIVE PROBLEM SELECTION          *
      *                                               *
      *               CGT PRODUCTIONS                 *
      *                                               *
      *************************************************


   Your current classification file is : CLASSF.DB

   Do you wish to change this [<CR> = N] ? (N/Y)
<CR>

   Your current problem selection key is:
     ( * = anything goes )

     Objective function type         : *
     Constraints type                : *
     Regularity                      : *
     Degree of available derivatives : *
     Problem interest                : *
     Explicit internal variables     : *
     Number of variables             : *
     Number of constraints           : *

   CHOOSE A PROBLEM CHARACTERISTIC THAT YOU WANT TO SPECIFY :
   ----------------------------------------------------------
```

```
     O   : Objective type          C : Constraint type
     R   : Regularity              I : Problem interest
     N   : Number of variables     M : Number of constraints
     D   : Degree of available analytic derivatives
     S   : Presence of explicit internal variables
    <CR> : No further characteristic, perform selection

   Your choice :
o

   OBJECTIVE FUNCTION TYPE :
   -------------------------

     C   : Constant                L : Linear
     Q   : Quadratic               S : Sum of squares
     N   : No objective
     O   : Other (that is none of the above)
    <CR> : Any of the above (*)

   Your choice :
s

     C   : Constant                L : Linear
     Q   : Quadratic               S : Sum of squares
     N   : No objective
     O   : Other (that is none of the above)
    <CR> : No further type

   Your choice :
N

     C   : Constant                L : Linear
     Q   : Quadratic               S : Sum of squares
     N   : No objective
     O   : Other (that is none of the above)
    <CR> : No further type

   Your choice :
<CR>

   You have specified objective of type(s): S N
   Do you wish to reconsider your choice [<CR> = N] ? (N/Y)
N

   Your current problem selection key is:
     ( * = anything goes )
```

```
     Objective function type         : S N
     Constraints type                : *
     Regularity                      : *
     Degree of available derivatives : *
     Problem interest                : *
     Explicit internal variables     : *
     Number of variables             : *
     Number of constraints           : *

   CHOOSE A PROBLEM CHARACTERISTIC THAT YOU WANT TO SPECIFY :
   ----------------------------------------------------------

     O   : Objective type          C : Constraint type
     R   : Regularity              I : Problem interest
     N   : Number of variables     M : Number of constraints
     D   : Degree of available analytic derivatives
     S   : Presence of explicit internal variables
    <CR> : No further characteristic, perform selection

   Your choice :
m

   NUMBER OF CONSTRAINTS :
   -----------------------

     F   : Fixed                   V : Variable
     I   : In an interval
    <CR> : Any number of variables (*)

   Your choice :
f

   You have specified a fixed number of constraints.
   Do you wish to reconsider your choice [<CR> = N] ? (N/Y)
<CR>

   SELECT A NUMBER OF CONSTRAINTS:
   -------------------------------

    (INT) : Select only problems with (INT) constraints
            (minimum 0, maximum 99999999, multiple choices are allowed)
    <CR>  : Any fixed number of variables (*)

   Your choice :
10

   SELECT A NUMBER OF CONSTRAINTS:
```

```
   ------------------------------

    (INT) : Select only problems with (INT) constraints
            (minimum 0, maximum 99999999, multiple choices are allowed)
     *    : Any fixed number of variables
    <CR>  : No further selection

   Your choice :
100

   SELECT A NUMBER OF CONSTRAINTS:
   ------------------------------

    (INT) : Select only problems with (INT) constraints
            (minimum 0, maximum 99999999, multiple choices are allowed)
     *    : Any fixed number of variables
    <CR>  : No further selection

   Your choice :
1000

   SELECT A NUMBER OF CONSTRAINTS:
   ------------------------------

    (INT) : Select only problems with (INT) constraints
            (minimum 0, maximum 99999999, multiple choices are allowed)
     *    : Any fixed number of variables
    <CR>  : No further selection

   Your choice :
<CR>

   You have specified a number of constraints in the set:
          10   100  1000
   Do you wish to reconsider your choice [<CR> = N] ? (N/Y)
<CR>

   Your current problem selection key is:
     ( * = anything goes )

     Objective function type         : S N
     Constraints type                : *
     Regularity                      : *
     Degree of available derivatives : *
     Problem interest                : *
     Explicit internal variables     : *
     Number of variables             : *
```

```
     Number of constraints           :    10   100  1000

   CHOOSE A PROBLEM CHARACTERISTIC THAT YOU WANT TO SPECIFY :
   ----------------------------------------------------------

     O   : Objective type          C : Constraint type
     R   : Regularity              I : Problem interest
     N   : Number of variables     M : Number of constraints
     D   : Degree of available analytic derivatives
     S   : Presence of explicit internal variables
    <CR> : No further characteristic, perform selection

   Your choice :
<CR>

   MATCHING PROBLEMS :
   -------------------

     ARGLINA    ARGLINB    ARGLINC    ARGTRIG    ARTIF
     BDVALUE    BRATU2D    BRATU2DT   BRATU3D    BROYDN3D
     BROYDNBD   CBRATU2D   CBRATU3D   CHANDHEQ   CHEMRCTA
     CHEMRCTB   CORKSCRW   DIXCHLNV   HAGER1     HAGER3
     INTEGREQ   LIARWHD    MSQRTA     MSQRTB     SEMICON1
     SEMICON2   SPMSQRT    ZIGZAG

      28 Problem(s) match(es) the specification.


   Do you wish to save the problem names to a file [<CR> = N] ? (N/Y)
y
   Input the filename you want (up to 32 characters):

listing

   Do you wish to make another selection [<CR> = N] ? (N/Y)
<CR>
```

References

[Averick and Moré, 1991] B. M. Averick and J. J. Moré. The Minpack-2 test problem collection. Technical Report ANL/MCS-TM-157, Argonne National Laboratory, Argonne, USA, 1991.

[Averick *et al.*, 1991] B. M. Averick, R. G. Carter, and J. J. Moré. The Minpack-2 test problem collection (preliminary version). Technical Report ANL/MCS-TM-150, Argonne National Laboratory, Argonne, USA, 1991.

[Boggs and Tolle, 1989] P. T. Boggs and J. W. Tolle. A strategy for global convergence in a sequential quadratic programming algorithm. *SIAM Journal on Numerical Analysis*, 26(3):600–623, 1989.

[Buckley, 1989] A. G. Buckley. Test functions for unconstrained minimization. Technical Report CS-3, Computing Science Division, Dalhousie University, Dalhousie, Canada, 1989.

[Bus, 1977] J.C.P. Bus. A proposal for the classification and documentation of test problems in the field of nonlinear programming. Technical report, Mathematisch Centrum,Amsterdam, 1977.

[Conn *et al.*, 1990] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. In R. Glowinski and A. Lichnewsky, editors, *Computing Methods in Applied Sciences and Engineering*, pages 42–54, Philadelphia, USA, 1990. SIAM.

[Conn *et al.*, 1992a] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[Conn *et al.*, 1992b] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Numerical experiments with the LANCELOT package (Release A) for large-scale nonlinear optimization. Research Report RC 18434, IBM T. J. Watson Research Center, Yorktown, USA, 1992. Also issued as Research Report 92-075, RAL, Chilton, England and Technical Report 92/16, Department of Mathematics, FUNDP, Namur, Belgium.

[Conn *et al.*, 1993a] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Complete numerical results for LANCELOT Release A. Research Report RC 18750, IBM T. J. Watson Research Center, Yorktown, USA, 1993. Also issued as Technical Report 92/15, Department of Mathematics, FUNDP, Namur, Belgium.

[Conn *et al.*, 1993b] A. R. Conn, Nick Gould, and Ph. L. Toint. A note on using alternative second-order models for the subproblems arising in barrier function methods for minimization. Research Report RC18898, IBM T. J. Watson Research Center, Yorktown, USA, 1993. Also issued as Research Report TR/PA/93/17, CERFACS, Toulouse, France, and Technical Report 93/16, Department of Mathematics, FUNDP, Namur, Belgium.

[Dembo, 1984] R. S. Dembo. A primal truncated-Newton algorithm with application to large-scale nonlinear network optimization. Technical Report 72, Yale School of Management, Yale University, New Haven, USA, 1984.

[Dennis and Schnabel, 1983] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* Prentice-Hall, Englewood Cliffs, USA, 1983.

[Gay, 1985] D. M. Gay. Electronic mail distribution of linear programming test problems. Mathematical Programming Society COAL Newsletter, December 1985.

[Gill *et al.*, 1986] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for NPSOL (version 4.0): A Fortran package for nonlinear programming. Technical Report SOL86-2, Department of Operations Research, Stanford University, Stanford, California 94305, USA, 1986.

[Gould, 1991] N. I. M. Gould. An algorithm for large-scale quadratic programming. *IMA Journal of Numerical Analysis*, 11(3):299–324, 1991.

[Griewank and Toint, 1982] A. Griewank and Ph. L. Toint. On the unconstrained optimization of partially separable functions. In M. J. D. Powell, editor, *Nonlinear Optimization 1981*, pages 301–312, London and New York, 1982. Academic Press.

[Gulliksson, 1990] M. Gulliksson. *Algorithms for Nonlinear Least Squares with Applications to Orthogonal Regression.* PhD thesis, Institute of Information Processing, University of Umeå, S-901 87 Umeå, Sweden, 1990.

[Harwell Subroutine Library, 1993] Harwell Subroutine Library. *A catalogue of subroutines (release 11).* Advanced Computing Department, Harwell Laboratory, Harwell, UK, 1993.

[Hock and Schittkowski, 1981] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes.* Springer Verlag, Berlin, 1981. Lectures Notes in Economics and Mathematical Systems 187.

[International Business Machines Corporation, 1978] International Business Machines Corporation. Mathematical programming system extended (MPSX) and generalized upper bounding (GUB). Technical Report SH20-0968-1, IBM Corporation, 1978. MPSX Standard.

[International Business Machines Corporation, 1990] International Business Machines Corporation. *Optimization Subroutine Library : Guide and Reference*, second edition, 1990.

[Koontz *et al.*, 1985] J.E. Koontz, R.B. Schnabel, and B.E. Weiss. A modular system of algorithms for unconstrained minimization. *ACM Transactions on Mathematical Software*, 11:419–440, 1985. Also available as Technical Report CU-CS-240-82, Department of Computer Science, University of Colorado, Boulder, CO.

[Liu and Nocedal, 1989] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming, Series B*, 45:503–528, 1989.

[Moré and Toraldo, 1991] J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[Moré *et al.*, 1981] J. J. Moré, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, 1981.

[Murtagh and Saunders, 1978] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14:41–72, 1978.

[Murtagh and Saunders, 1987] B. A. Murtagh and M. A. Saunders. MINOS 5.1 USER'S GUIDE. Technical Report SOL83-20R, Department of Operations Research, Stanford University, Stanford, USA, 1987.

[Powell, 1982] M.J.D. Powell. Extensions to subroutine VF02. In R.F. Drenick and F. Kozin, editors, *Systems Modelling and Optimization. Lecture notes in control and Information sciences 38*, pages 529 – 538, Berlin, 1982. Springer-Verlag.

[Schittkowski, 1987] K. Schittkowski. *More Test Examples for Nonlinear Programming Codes.* Springer Verlag, Berlin, 1987. Lecture notes in economics and mathematical systems, volume 282.

[Schnabel and Chow, 1991] R.B. Schnabel and T.-T. Chow. Tensor methods for unconstrained optimization using second derivatives. *SIAM Journal on Optimization*, 1(3):293–315, 1991.

[Toint and Tuyttens, 1992] Ph. L. Toint and D. Tuyttens. LSNNO: a Fortran subroutine for solving large scale nonlinear network optimization problems. *ACM Transactions on Mathematical Software*, 18(3):308–328, 1992.

[Toint, 1983] Ph. L. Toint. Test problems for partially separable optimization and results for the routine PSPMIN. Technical Report 83/4, Department of Mathematics, FUNDP, Namur, Belgium, 1983.