Vol. 00, No. 00, Month 200x, 1–21

USER GUIDE

GlobSol User Guide

R. Baker Kearfott, Department of Mathematics, University of Louisiana, U.L. Box 4-1010, Lafayette, Louisiana, 70504-1010, USA (`rbk@louisiana.edu`).

(*Received 00 Month 200x; in final form 00 Month 200x*)

We explain installation and use of the GlobSol package for mathematically rigorous bounds on all solutions to constrained and unconstrained global optimization problems, as well as nonlinear systems of equations. This document should be of use both to people with optimization problems to solve and to people incorporating GlobSol's components into other systems or providing interfaces to GlobSol.

Keywords: nonconvex optimization; global optimization; GlobSol; constraint propagation

2000 AMS Subject Classifications: 90C30; 65K05; 90C26; 65G20

1. Introduction and Purpose

GlobSol is an open-source, self-contained package for finding validated solutions to unconstrained and constrained global optimization problems, and for finding rigorous enclosures to all solutions to systems of nonlinear equations. GlobSol is distributed under the Boost license [3]. GlobSol uses a branch and bound algorithm for this purpose. Thus, the structure of GlobSol's underlying algorithm is somewhat similar to that of many optimization packages, such as BARON [18]. However, GlobSol's goal is to provide *rigorous bounds on all possible solutions.* That is, GlobSol's output consists of lists of small intervals or interval vectors within which GlobSol has proven with the rigor of a mathematical proof that any possible solutions of the problem posed must lie[1]. (This is done by careful design of algorithms and use of directed rounding.) GlobSol is thus well-suited to solving problems in which such bounds and guarantees are desired or important; for problems where such guarantees are not necessary, other software may solve the problems more efficiently, or may solve a wider range of problems.

This document focusses on:

- installation of GlobSol,
- quick use of GlobSol once installed,
- interpretation of GlobSol's results,
- details on how to configure GlobSol,
- GlobSol's components and package organization,
- improvements in the works.

There are the following overall goals:

(1) to help potential users decide the appropriateness of GlobSol for their application,

[1]This forces the GlobSol algorithms to differ in various important respects from algorithms that do not claim to provide solutions with mathematical rigor; see, for example, [8].

ISSN: 1055-6788 print/ISSN 1029-4937 online
© 200x Taylor & Francis
DOI: 10.1080/03081080xxxxxxxxx
http://www.informaworld.com

(2) to guide users through the production and interpretation of results, and
(3) to give developer-users information and pointers they need to incorporate GlobSol and its components into wider systems (such as input to GlobSol using AMPL [5] or GAMS [16], or providing a MATLAB interface to GlobSol.)

This document does not trace the history of GlobSol, nor does this document fully explain the underlying algorithms, their testing, and mathematical justifications. For those topics, see the series of talks and publications listed at `http://interval.louisiana.edu/preprints.html`. Example references are [7] and [13]. In particular, [7] contains a description of various components the package `INTOPT_90`, a very early version of GlobSol. However, although [7] still contains relevant background, the usage instructions in [7] are now partially obsolete.

2. Installation of GlobSol

Not only is GlobSol written in Fortran 95, but the description of the problem to be solved is given to GlobSol is as a Fortran 95 program. For this reason, a Fortran 95 compiler must be resident on the system where GlobSol is to be installed and used, and this should be the same compiler with which GlobSol was built. (However, see §9 for interfacing GlobSol to other systems so that Fortran 95 input is not needed.) Fortunately, two high-quality open-source free Fortran compilers have become available: the `g95` compiler (see `http://g95.sourceforge.net/` and `http://en.wikipedia.org/wiki/G95`) and the `gfortran` compiler. Both of these compilers are licensed under the GNU general public license, so can be distributed freely with GlobSol itself. `gfortran` is desirable because it is slated to be an integral part of the Gnu compiler suite (including `gcc`, etc.), but `g95` was somewhat farther in its development when the latest packaging of GlobSol was done.

2.1. *Windows*

For Windows environments, we have provided an executable installation package, produced with the open-source 7-zip package. This executable installation comes with an executable copy of `g95`. To install this version of GlobSol, simply perform the following steps.

(1) Create a directory to be the root directory of the GlobSol installation. Henceforth, we will refer to this directory as <`GlobSol root dir`>.
(2) Visit the registration page `http://interval.louisiana.edu/GlobSol/download_GlobSol.html`.
(3) Once you have given instructions and have displayed the download page click on the link to obtain one of the two installation executables (`GlobSol-win-gnu.correct_IO.exe`) and store it in <`GlobSol root dir`>.
(4) Run `GlobSol-win-gnu.correct_IO.exe` from[1] <`GlobSol root dir`>.
(5) Follow the remaining instructions on the GlobSol download page.

Executing `GlobSol-win-gnu.correct_IO.exe` and following subsequent instructions installs a "production" version of GlobSol. The file `GlobSol-win-gnu.correct_IO.exe` installs a version of GlobSol that includes correctly rounded output, based on David Gay's "`gdtoa`" library[2]. The distribution

[1]It can be assured that `GlobSol-win-gnu.correct_IO.exe` is run from ⟨GlobSol root dir> by running it from a command prompt in that directory.
[2]Included in this GlobSol distribution, and often included in Linux distributions.

also includes all of GlobSol's source files and external (freely distributable) source, and the GNU "make" program.

With this installation, the path should be set to include `C:\GlobSol`, `C:\GlobSol\executables` and `C:\g95`, and the environment variable `LIBRARY_PATH` should contain `C:\g95\lib` and `C:\GlobSol\executables`. This path is initially set in the command window during installation, but is not set globally within Windows. This can be set in subsequently opened command windows by executing `setup_path.bat` in `<GlobSol root dir>`, or can be changed globally in Windows.

2.2. *Linux*

For Linux, installation of GlobSol is not presently fully automatic, but should not be difficult for most systems, provided a standard-conforming Fortran 95 compiler is available. GlobSol is written totally in standard-conforming Fortran 95, except that the distribution that includes correctly rounded output employs Fortran 2003 constructs for standardized interface to `C`. A makefile using simple "lowest common denominator" constructs[1] with extensive in-line documentation is included, while specific versions of this makefile for Windows and Unix / Linux[2] can be downloaded.

The steps for obtaining and installing GlobSol version and makefile for Unix or Linux are as follows.

(1) Register to download GlobSol through the link `http://interval.louisiana.edu/GlobSol/download_GlobSol.html`.
(2) Click on the link sent in the confirmation email.
(3) On the web page that displays, click on the link "Download the zip file GlobSol.zip" to download the GlobSol source. Place this "zip" file into the directory into which you wish to install GlobSol.
(4) From the same page, click on the link "unpack_sun_sun" to obtain a shell script to unpack the zip file in the appropriate way[3], and place this script into the same directory as `GlobSol.zip`.
(5) Read the remainder of the "GlobSol Downloading Instructions" page carefully.
(6) Click on the link "install.html" from the "GlobSol Downloading Instructions" page.
(7) Read the "install.html" page carefully, then follow its instructions. You will probably need to alter the system-dependent routines, but you may be able to use `unpack_sun_sun` to ease the task.

Thus, persons with moderate knowledge of creating and using a makefile and knowledge of their Fortran 95 compiler options should be able to compile and install GlobSol within their own environment.

3. Initial Use of GlobSol

Solving problems with GlobSol involves

[1]So it will execute with many versions of "make."
[2]The main difference between these two versions is the use of "\" in path names instead of "/", and care must be taken in converting between the two.
[3]This script is meant for Sun Solaris with the Sun Fortran compiler. However, the script is largely generic.

(1) making sure a copy of the general configuration file `GlobSol.CFG` is present in the directory from which GlobSol is to be run,
(2) creating a Fortran 95 file that defines the problem and constraints,
(3) creating a "box data file" that defines the search region, overall tolerances, and, within limits, how the boundary of the search region is handled.
(4) running GlobSol with the "`globsol`" script[1].

We now give details and examples.

3.1. *The GlobSol Input Files*

Various sample input files are given in the directory `<GlobSol root dir>/integration_test_data`. In particular, a `GlobSol.CFG` file is there, and should be copied into the working directory where GlobSol is to be run. A simple sample Fortran 95 file defining the problem for GlobSol appears as follows:

```
!  This is a simple example of mixed equality and inequality
!  constraints, one of which is binding at the solution.

PROGRAM SIMPLE_MIXED_CONSTRAINTS

  USE CODELIST_CREATION

! The independent variables are in the CDLVAR array passed
! to INITIALIZE_CODELST.  The number of independent variables
! must be the exact dimension of this array.
   TYPE(CDLVAR), DIMENSION(2):: X
! The objective function must be declared this way.
   TYPE(CDLLHS):: PHI
! The dimension of this array should be the number of inequality
! constraints.
   TYPE(CDLINEQ), DIMENSION(2):: G
! The dimension of this array should be the number of equality
! constraints.
   TYPE(CDLEQ), DIMENSION(1) :: C

      CALL INITIALIZE_CODELIST(X)

! This statement defines the objective
  PHI = -2*X(1)**2 - X(2)**2

! These two statements define the inequality constraints
! G(1) <= 0 and G(2)<= 0.
  G(1) = X(1)**2 + X(2)**2 - 1
  G(2) = X(1)**2 - X(2)

! This statement defines the equality constraint C(1) == 0.
  C(1) = X(1)**2 - X(2)**2

      CALL FINISH_CODELIST

END PROGRAM SIMPLE_MIXED_CONSTRAINTS
```

This file, stored as `mixed.f90` in `<GlobSol root dir>/integration_test_data`, is largely self-explanatory. It defines the problem

$$\begin{aligned}
\text{minimize} \quad & \varphi(x) = -2x_1^2 - x_2^2 \\
\text{subject to equality constraint} \quad & c(x) = x_1^2 - x_2^2 \qquad = 0, \\
\text{and inequality constraints} \quad & g_1(x) = x_1^2 + x_2^2 - 1 \leq 0, \\
& g_2(x) = x_1^2 - x_2 \qquad \leq 0.
\end{aligned}$$

In such Fortran 95 problem definition files, most Fortran 95 syntax, including subroutine calls, can be used, with some exceptions. The exceptions are due to the

[1]A copy of this script, named "`globsol.bat`" on Windows systems and "`globsol`" on Unix or Linux systems, resides on <GlobSol root dir>.

fact that operator overloading is used to create the internal representation of the problem GlobSol employs in its algorithms. The following rules apply.

(1) The independent variables must be in a single array, declared to be of type `CDLVAR`, and whose dimension is the dimension of the domain space[1]. This array must be passed as an argument to the routine `INITIALIZE_CODELIST`, which should be called as the first executable statement.
(2) The objective must be declared to be of type[2] `CDLLHS`.
(3) The equality constraint functions must be declared to be of type `CDLEQ`.
(4) The inequality constraint functions must be declared to be of type `CDLINEQ`.
(5) Any intermediate quantities that are stored and that depend on the independent variables must be declared to be of type `CDLVAR`
(6) Binary operations may occur between variables of type `CDLVAR` and between variables of type `CDLVAR` and integers, double precision, or `TYPE(INTERVAL)` variables or constants. To prevent a common programming misconception that reduces the precision of results, operations between `CDLVAR` variables and single precision real constants and variables are not supported. For instance, if `T` is declared to be of type `CDLVAR` then the operation `0.1*T` will result in a compile-time error, whereas the operation `0.1D0*T` will not.
(7) Variables of type `CDLLHS`, `CDLEQ`, and `CDLINEQ` are "write-once" variables. That is, they can only appear on the left side of an assignment statement, and they can only be assigned once. If one of these quantities, say `PHI`, is the result of an accumulation, for example, it may be assigned by first computing its value in a variable of type `CDLVAR`, say `SUM`, then assigning `SUM` to `PHI`.
(8) Variables of type `CDLVAR` cannot appear within the decision part of a branch statement. For example, if `V` is of type `CDLVAR`, then "`IF (V < 0)`", "`WHILE (V > 1)`," etc., are invalid. This is because the actual value of `V` is not known until GlobSol itself runs, while compiling and running programs such as `mixed.f90` merely produces an internal symbolic representation of the function. To describe branching for GlobSol, the χ function, explained in [7, p. 211], can be used.

The following is the contents of a box data file. This box data file should be named `mixed.DT1` to correspond to `mixed.f90` and to be designated as instance[3] "1" of the data.

```
1D-5
 0   1
 0   1
F F
F F
```

The first line of this file gives an overall tolerance. The largest difference between a lower bound and an upper bound of a solution component is on the order of the square root of this tolerance. Because of the interplay between tolerances within GlobSol's algorithms, this tolerance[4] will be automatically reset to a larger value if it is set less than the square root of the double precision machine epsilon[5]. The next two[6] lines of this file represent the bounds on the components of the independent variables within which the global optimizers are sought, that is, they define the

[1]However, portions or elements of the array can be passed as arguments to a subroutine, in which the individual arguments are given different names.
[2]The name stands for "code list left hand side."
[3]The box data file may have a file type `DT?`, where "?" can be any digit between zero and 9. This digit is passed to the `globsol` script to control which of possibly several box data files is used.
[4]In the present version of GlobSol.
[5]About 1.49×10^{-8} in IEEE standard arithmetic.
[6]n lines for n independent variables.

search box[7].

The next two lines of this box data file are used to describe how the boundaries of the search box are to be treated. Each line consists of two characters, and each character can be either “F” or “T”. “T” signifies that the corresponding bound is to be treated as a bound constraint, but “F” signifies that the bound is not a bound constraint, but merely a bound on the search region. (This has consequences for interpretation of the results, as we explain below.) Caution should be exercised when setting these boundary indicators to be anything other than “F”. In particular, a preprocessing algorithm produces sub-problems the number of which increases exponentially with the number of boundary indicators set to “T”. For example, if there are n independent variables and each of the 2^n boundary indicators is set to “T,” then the preprocessing algorithm produces potentially 3^n sub-boxes. In such cases, boundary constraints should expressed in the `*.f90` file as general inequality constraints, and the boundary indicators in the box data file should be set mostly to “F.” Nonetheless, for small problems, expressing some or all bound constraints through boundary indicators may be the most efficient way[1] of using GlobSol. (For larger problems, with more bound constraints, at most one or two of the boundary indicators should be set to “T,” while the other bound constraints should be expressed simply as inequality constraints. Setting a boundary indicator to be “T” causes the “peeling” process explained in [7, §5.2.3] to be used.)

There are optional lines that may appear after the boundary indicators in the box data file. Immediately after the boundary indicators, an initial guess for a global optimizer may appear, one component per line. If such an initial guess appears in the box data file, then an additional line, giving “T” or “F,” may occur; a value of “T” on this line signifies that the initial guess is sufficiently good that an attempt to refine it is unnecessary.

3.2. *Interpretation of Results*

With the problem definition file `mixed.f90` and box data file `mixed.DT1` as above, we may run GlobSol with the command line[2]

```
globsol mixed 1
```

If everything is installed properly, this script produces the file `mixed.OT1`, portions of which are as follows[3].

```
 Output from FIND_GLOBAL_MIN on  12/22/2007  at  19:34:43.
 Version for the system is: February 2, 2007

 Codelist file name is: mixedG.CDL
 Box data file name is: mixed.DT1


 Initial box:
[ 0, 1 ], [ 0, 1 ]


 BOUND_CONSTRAINT:
   F F   F F
```

[7]If there are n independent variables, then there will be n such lines.
[1]Because this method may result in better upper bounds on the global optimum, due to less overestimation in the interval computations, because some of the variables in the evaluation are fixed at their point boundary values.
[2]Provided the <GlobSol root dir> directory is in the search path.
[3]We have abridged the output to save space and for clarity. Lines that we have omitted consist mainly of echoing of configuration variables and of performance statistics; we have replaced groups of such lines with a single line consisting of “.”.

```
 --------------------------------------
 CONFIGURATION VALUES:

 EPS_DOMAIN:   0.1000D-04   MAXITR:   20000
 MAX_CPU_SECONDS: 3600

.

 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ .7071, .7072 ], [ .7071, .7072 ]

PHI:
[ -1.501, -1.499 ]

 Box contains the following approximate root:
.7071, .7071

 OBJECTIVE ENCLOSURE AT APPROXIMATE ROOT:
[ -1.501, -1.5 ]

.

 ------------------------------------------------
 ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
       20000  OF BOXES.
 Number of bisections:  1

 .

 Total number of boxes processed in loop:  4
 BEST_ESTIMATE: -1.5
 Overall CPU time: .01001
 .
```

This output represents the best possible behavior one can expect from GlobSol. In particular, there is only one output box, and this output box has been verified to contain a feasible point. However, we must be cautious in interpreting these results. In particular, we cannot say without further analysis that there is only one global optimizer and that it has coordinates $x_1 \in [.7071, .7072]$ and $x_2 \in [.7071, .7072]$, with optimum value in $[-1.501, -1.499]$. The only thing that we can say for sure based only on this output is that any possible global optimizers within the initial box must lie within $x_1 \in [.7071, .7072]$ and $x_2 \in [.7071, .7072]$, and, if such an optimizer exists within those bounds, then its value must lie within the interval $[-1.501, -1.499]$. Here, `BEST_ESTIMATE` represents a rigorous upper bound for the global optimum, provided such an optimum exists.

Nonetheless, we *can* say more in many cases. If the problem *has* a global minimizer within the search box, then each global minimizer within the search box, must lie in one of the output boxes, and the bounds given for `PHI` must contain the global minimum. An analysis of the constraints in the simple problem represented by `mixed.f90` reveals that the feasible set is

$$\left\{(x_1, x_2) \mid x_2 = x_1,\ x_1 \le \sqrt{2}\right\}.$$

Since this set is unbounded, we still cannot conclude that a global optimizer exists within the search box. However, in this simple example, we can see that the global optimizer must indeed be as given in GlobSol's output. Furthermore, if we set all boundary indicators to “T,” then the set defined by the constraints and boundary indicators is compact, and since the objective is continuous, we know a global minimum must exist. The corresponding box data file is

```
1D-5
0   1
0   1
```

```
T T
T T
```

and the corresponding (abridged) GlobSol output file is

```
Output from FIND_GLOBAL_MIN on  12/22/2007  at  20:18:25.
Version for the system is: February 2, 2007

Codelist file name is: mixedG.CDL
Box data file name is: mixed.DT3


Initial box:
[ 0, 1 ], [ 0, 1 ]


BOUND_CONSTRAINT:
  T T   T T

---------------------------------------
CONFIGURATION VALUES:

EPS_DOMAIN:   0.1000D-04   MAXITR:   20000
MAX_CPU_SECONDS: 3600

.

THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

Box no.: 1
Box coordinates:
[ .7071, .7072 ], [ .7071, .7072 ]

PHI:
[ -1.501, -1.499 ]

.

-------------------------------------------------
ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
      20000  OF BOXES.
Number of bisections:  1

.

Total number of boxes processed in loop:  7
BEST_ESTIMATE: -1.5
Overall CPU time: .04006
CPU time in PEEL_BOUNDARY: .03004
CPU time in REDUCED_INTERVAL_NEWTON: 0
```

In summary, if we know that the constraints combined with the boundary indicators define a set that, when intersected with the search region, is compact, then the problem must contain a global optimizer within the search region, and any global optimizers within the search region must be within the output boxes. However, there may be extraneous boxes that do not correspond to global optimizers. This happens in ill-conditioned problems where GlobSol cannot resolve all boxes to within the tolerance.

As a first example, if we increase the search region for `mixed.f90` to $x_1 \in [-2, 2]$ and $x_2 \in [-2, 2]$, then the disk defined by the constraint $x_1^2 + x_2^2 \leq 1$ must lie completely within the search region, so the intersection of the feasible region with the search region must necessarily be compact, so GlobSol must find all global minimizers. In fact, running GlobSol with the box data file

```
1D-5
-2 2
-2 2
F F
F F
```

gives the result

```
.
THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:
```

```
Box no.: 1
Box coordinates:
[ -.7072, -.7071 ], [ .7071, .7072 ]

PHI:
[ -1.501, -1.499 ]
.
ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
      20000  OF BOXES.
Number of bisections:  5
.
Total number of boxes processed in loop:  11
BEST_ESTIMATE: -1.5
Overall CPU time: .02003
CPU time in PEEL_BOUNDARY: 0
.
```

For a second example, consider minimizing $x_1^2 - x_2^2$ subject to $x_1 - x_2 \le 0$, with search region $x_1 \in (-1, 1)$, $x_2 \in (-1, 1)$. (The open intervals correspond to not treating the bounds on the search region as bound constraints.) This problem has no minimizer within the search region, and, indeed, has no minimizer globally. However, the infimum of φ over the intersection of the feasible set with the search region is -1, and it occurs at $(x_1, x_2) = (0, 1)$. Corresponding `*.f90` and `*.DT?` files are

```
 PROGRAM EXAMPLE_2

  USE CODELIST_CREATION

      TYPE(CDLVAR), DIMENSION(2):: X
      TYPE(CDLLHS):: PHI
       TYPE(CDLINEQ) :: G

      CALL INITIALIZE_CODELIST(X)

  PHI =X(1)**2 - X(2)**2

  G = X(1) - X(2)

      CALL FINISH_CODELIST

END PROGRAM EXAMPLE_2
```

and

```
1D-5
-1 1
-1 1
F F
F F
```

The corresponding output contains 63 "boxes with verified feasible points," all lying on the boundary of the region defined by the inequality constraint, and it gives "`BEST_ESTIMATE: 2.5e-11`." If we change the box data file to include $(x_1, x_2) = (0, .999)$ as a good approximation[1] to a global minimizer, that is, if we use the following box data file

```
1D-5
-1 1
-1 1
F F
F F
0
.999D0
T
```

then we obtain only one box `[ -.03163, .03163 ], [ .9673, 1 ]`, with `BEST_ESTIMATE: -.998`. Furthermore, the intersection of each of the 63 boxes obtained in the previous run is empty. This is not a contradiction, since the problem itself has no global optimizer within the search region.

On the other hand, if we set a lower bound constraint on x_1 and an upper bound constraint on x_2, then the intersection of the feasible region with the search box is compact, and a global optimizer exists within the feasible region. The correspond-

[1]GlobSol does not allow this point to be too near to the boundary relative to the tolerance, if that boundary does not represent a bound constraint.

ing box data file is

```
1D-5
-1 1
-1 1
T F
F T
```

and the GlobSol output file contains only one box:

```
.
 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ -2.226e-308, 2.226e-308 ], [ 1, 1 ]

PHI:
[ -1.001, -0.999 ]
.
 ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
       20000  OF BOXES.
.
 Total number of boxes processed in loop:  7
 BEST_ESTIMATE: -1
 Overall CPU time: .01001
 CPU time in PEEL_BOUNDARY: .01001
 .
```

The undesirable result with the configuration file

```
1D-5
-1 1
-1 1
F F
F F
```

is not a contradiction, but it represents a performance issue with GlobSol that can be improved. In fact, with an experimental version[1] of GlobSol that uses linear relaxations as described in [10] and with the configuration file immediately above, the output consists of a single box, as follows:

```
 .
 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.:           1
 Box coordinates:
 [   -0.3162D-02,   0.3162D-02 ] [    0.9968D+00,   0.1003D+01 ]
PHI:
 [   -0.1006D+01,  -0.9937D+00 ]
.
 Total number of boxes processed in loop:           2
.
 BEST_ESTIMATE:   -0.1000D+01
 Overall CPU time:    0.2504D+00
 CPU time in PEEL_BOUNDARY:    0.0000D+00
.
```

3.3. *Standard Functions Supported*

GlobSol presently has partial or full support for:

> `ABS`, `ACOS`, `ASIN`, `ATAN`, `COS`, `COSH`, `EXP`, `LOG`, `MAX`, `MIN`,
> `SECH`, `SIN`, `SINH`, `SQRT`, `TAN`, and `TANH`.

For example, with

```
 PROGRAM EXAMPLE_SIN

  USE CODELIST_CREATION

      TYPE(CDLVAR), DIMENSION(1):: X
```

[1]We hope to publicly release this version within a year of this report.

```
      TYPE(CDLLHS):: PHI

      CALL INITIALIZE_CODELIST(X)

  PHI =( SIN(X(1)) - (X(1)/2) )**2

      CALL FINISH_CODELIST

END PROGRAM EXAMPLE_SIN
```

and box data file

```
1D-5
0 6.29d0
T T
```

GlobSol produces

```
.
 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ 1.895, 1.896 ]

PHI:
[ 0, 3.379e-24 ]

.
 -------------------------------------------------

 Box no.: 2
 Box coordinates:
[ -7.555e-12, 1.065e-11 ]

PHI:
[ 0, 6.976e-23 ]
.
 Total number of boxes processed in loop:  5
 BEST_ESTIMATE: 2.225e-308
 Overall CPU time: .04006
 CPU time in PEEL_BOUNDARY: 0
.
```

GlobSol can even handle some non-differentiable functions. For example, with

```
PROGRAM EXAMPLE_ABS

  USE CODELIST_CREATION

      TYPE(CDLVAR), DIMENSION(1):: X
      TYPE(CDLLHS):: PHI

      CALL INITIALIZE_CODELIST(X)

  PHI = ABS(X(1))

      CALL FINISH_CODELIST

END PROGRAM EXAMPLE_ABS
```

and box data file

```
1D-5
-2 1
T T
```

GlobSol gives:

```
.
 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ -7.348e-07, 7.476e-06 ]

PHI:
[ 0, 7.476e-06 ]

.
 Total number of boxes processed in loop:  1
 BEST_ESTIMATE: 3.91e-07
 Overall CPU time: 0
 CPU time in PEEL_BOUNDARY: 0
.
```

However, use of non-smooth functions may not always lead to desirable results in general. See [7, Chapter 6] for some theory and further discussion.

3.4. *Auxiliary Capabilities*

The experimental version of GlobSol and the version obtained from running `GlobSol-win-gnu.correct_IO.exe` have an easy facility for obtaining values and derivatives of the objective and constraints, through the `rg` script. For example, suppose we want to obtain an interval enclosure for the range of values for the problem `mixed.f90` (on page 4), over the range specified in `mixed.DT1`. The program that does this is `eval_if`. To run the program, we execute the command

```
rg eval_if mixed 1
```

This produces a file `mixed.IF1`, which contains the following.

```
 Version for the system is: February 2, 2007

 Codelist file name is: mixed.CDL
 Box data file name is: mixed.DT1


File: EVAL_IF.F90 -- F
Reference: Kearfott page 95, SUBROUTINE F(X,FVAL)

This subroutine returns a set of bounds on the ranges of the function
components over the box represented in the interval vector X(:) in the
interval vector FVAL.


Initial box:

[ 0, 1 ], [ 0, 1 ]


Interval value F(X) using the natural interval extension:

[ -3.001, 2.226e-308 ]
```

The list of available programs[1] appears in Table 1. The function `make_tex` only gives good results when the objective and constraints do not have too many operations, since it typesets the objective as a one-line formula[2].

4. Configuration of GlobSol

The file `GlobSol.CFG` contains definitions of global variables in GlobSol that specify

(1) limits on memory used in creation of the internal representation of the objective and constraints,
(2) amount of printing in various parts of the algorithm,
(3) number of digits to be printed in the output,
(4) tolerances used in various parts of the algorithm, and
(5) which algorithm paths are followed,

The syntax in the `GlobSol.CFG` file, developed by George Corliss et al., includes simple assignment statements and in-line documentation. The in-line documenta-

[1]Other programs are available, as seen in <`GlobSol root dir`>`/executables`, but are not normally of interest since they do not add significant functionality above those listed.
[2]Also, because of the `g95` compiler's handling of "\" in output character strings, `make_tex` as distributed may work for the `g95` versions, but not other versions.

Table 1. Auxiliary functions available in GlobSol

Name	Function	Output file extension
`eval_cd`	interval bounds on the ranges of the equality constraints	`CD?`
`eval_cgd`	interval bounds on the ranges of the gradients of the equality constraints	`CG?`
`eval_djm`	interval bounds on the ranges of the components of the gradient of the objective	`JM?`
`eval_dsm`	slope bounds on the ranges of the components of the gradient of the objective, using the center of the input intervals as base point	`DS?`
`eval_f2`	interval bounds on the range of the objective using a mean value extension	`F2?`
`eval_icd`	interval bounds on the ranges of the inequality constraints	`IC?`
`make_cosy`	creates a function to be used with the COSY [2] system to evaluate the objective	`fox`
`make_f77`	creates a `Fortran 77` function subroutine to evaluate the objective	`f`
`make_tex`	creates a LaTeX representation of the objective, equality, and inequality constraints	`tex`
`prntcodl`	creates a readable version of the code list (see §5)	`RCL`

tion describes each variable. Variables that do not occur in `GlobSol.CFG` are given default values after GlobSol parses `GlobSol.CFG`.

Although there are scores of control variables in the file, those of most interest include the following.

4.1. *Time Limits*

- `MAX_CPU_SECONDS` specifies the total number of seconds GlobSol is allowed to run, before the search for optimizers is aborted.
- `MAXITR` specifies the total number of nodes to be considered in the branch-and-bound search for optimizers, before the search is aborted.

4.2. *Printing*

- `PRINT_LENGTH` specifies the number of digits to be printed in each floating point number, in the "correct I/O" version of GlobSol. In the older version of GlobSol, `PRINT_LENGTH = 1` signifies that four digits are to be printed, and `PRINT_LENGTH = 2` signifies that 16 digits are to be printed.
- `PRINTING_IN_OVERALL_ALGORITHM` is a non-negative integer that controls the amount of printing done during algorithm execution. Setting it to successively higher values (0, 1, 2, . . . , 7) gives successively more information.
- `WANT_TABLE` is set to either "T" or "F". If `WANT_TABLE = T`, then a table of values, suitable for importation into a spreadsheet, is placed in the file `OPTTEST.TBL`. This table includes date and time stamps, some of the configuration values, and performance data. Successive runs of GlobSol append to this table, rather than overwriting it. This table printing is actually done in internal subroutine `PRINT_OPT_SPREADSHEET_LINE` in file `<GlobSol root dir>/f90intbi/finish_find_global_min.f90`.

In addition to the output <filename>.OT1, issuing "globsol <filename> 1" will produce a file <filename>_verified_list.txt if any boxes containing verified feasible points are produced, and a file <filename>_unverified_list.txt if any answer boxes not containing verified feasible points are produced. These files contain the interval coordinates of the boxes in list format, one output box per line; for n variables, there are $2n$ floating point values, representing the lower bounds and upper bounds for each coordinate. Such files can be imported directly into, e.g., MATLAB. Once these files are imported directly into MATLAB, the boxes can be plotted with MATLAB "m" files found in <GlobSol root dir>/IO, as follows:

(1) Run GlobSol_data_to_box.m to convert the array resulting from the default import of the GlobSol box list into MATLAB.
(2) Run plotboxes.m on the array output from GlobSol_data_to_box.m to plot the list of boxes.

4.3. *Nonlinear Systems*

There is a special algorithm path for solving square nonlinear systems of equations in GlobSol, that is usually more efficient than simply posing the system as a global optimization problem[1] with the equations as equality constraints. In particular, if NONLINEAR_SYSTEM = T, then GlobSol attempts to solve the nonlinear system represented as the equality constraints; the objective and inequality constraints can still appear, to help eliminate portions of the search space. For example, suppose we set NONLINEAR_SYSTEM = T in GlobSol.CFG, and we use the following NLS1.f90 and NLS1.DT1 files.

```
 PROGRAM NLS1

  USE CODELIST_CREATION

      TYPE(CDLVAR), DIMENSION(2):: X
      TYPE(CDLLHS):: PHI
       TYPE(CDLINEQ) :: G
      TYPE(CDLVAR), DIMENSION(2) :: E
      TYPE(CDLEQ), DIMENSION(2) :: EQUATION

      CALL INITIALIZE_CODELIST(X)

  E(1) = X(1)**2 - X(2)**2

  E(2) = X(1)**2 + X(2)**2 - 1

  EQUATION(1) = E(1)
  EQUATION(2) = E(2)

  PHI = E(1)**2 + E(2)**2

  G = -X(1)  -X(2)

      CALL FINISH_CODELIST

END PROGRAM NLS1
```

```
1D-10
-1 1
-1 1
F F
F F
```

Issuing "globsol NLS1 1" then gives

```
 Output from FIND_GLOBAL_MIN on  12/25/2007  at  01:44:00.
 Version for the system is: February 2, 2007

 Codelist file name is: NLS1G.CDL
 Box data file name is: NLS1.DT1


 Initial box:
```

[1]With either constant objective or with objective equal to the sum of squares of the residuals, etc.

```
[ -1, 1 ], [ -1, 1 ]

.
 ---------------------------------------
 CONFIGURATION VALUES:

 EPS_DOMAIN:   0.1000D-09   MAXITR:   20000
 MAX_CPU_SECONDS: 3600
.
 NONLINEAR_SYSTEM: T
.
 THERE WERE NO BOXES IN THE LIST OF SMALL BOXES.

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ .707, .7073 ], [ .707, .7073 ]

PHI:
[ 0, 1.601e-07 ]

.
 -------------------------------------------------

 Box no.: 2
 Box coordinates:
[ .7071, .7072 ], [ -.7072, -.7071 ]

PHI:
[ 0, 1.604e-23 ]

.
 -------------------------------------------------

 Box no.: 3
 Box coordinates:
[ -.7072, -.7071 ], [ .7071, .7072 ]

PHI:
[ 0, 1.604e-23 ]

.
 -------------------------------------------------
 ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
       20000  OF BOXES.
 Number of bisections:  4
.
 Total number of boxes processed in loop:  11
 BEST_ESTIMATE: 0
 Overall CPU time: 0
.
```

Note: Although the nonlinear systems path shares elements in common with the separate algorithm explained in [7, Chapter 4], the overall algorithm in the present version of GlobSol is the same algorithm as for general global optimization. The main differences are as follows.

- Where an interval Newton method is used, the system is formed from the equations only, rather than as a Kuhn–Tucker or Fritz–John system.
- The sum of squares of the residual is used as objective, and the best upper bound for the optimum is initialized to 0.

4.4. *Problems with Uncertain (Fuzzy) Coefficients*

It sometimes occurs that certain coefficients in a problem are not known precisely, and cannot be computed more precisely from available data. In that case, it makes sense to treat those coefficients as imprecisely known constants, rather than as variables, and to find the range of all possible solutions to the problem as the coefficients range over all possible values. For a further discussion of the distinction between unknown variables and imprecisely known coefficients, and consequences thereof, see [11].

To express an imprecisely known constant in GlobSol, one simply declares it to be an interval value. GlobSol then adjusts the stopping tolerances[1] so that the output bounds fit the actual bounds on the range of possible solutions. For example, suppose

```
 PROGRAM UNKNOWN_CONSTANT
    USE CODELIST_CREATION
    IMPLICIT NONE

    INTEGER, PARAMETER:: NN=1
    TYPE(CDLVAR), DIMENSION(NN):: X
    TYPE(CDLLHS) :: PHI

    TYPE(INTERVAL) :: A

    CALL INITIALIZE_CODELIST(X)

    A = INTERVAL(-1,1)

    PHI = (X(1) - A)**2 + 10*A**2

    CALL FINISH_CODELIST
END PROGRAM UNKNOWN_CONSTANT
```

is stored in `unknown_constant.f90` and

```
 1d-9 -10 10 F F
```

is stored in `unknown_constant.DT1`. Then, issuing `globsol unknown_constant 1` results in the following in file `unknown_constant.OT1`.

```
 Codelist file name is: unknown_constantG.CDL
 Box data file name is: unknown_constant.DT1


 Initial box:
[ -10, 10 ]


 BOUND_CONSTRAINT:
   F F

.
 LIST OF SMALL BOXES:

 Box no.: 1
 Box coordinates:
[ -1.001, -.0003162 ]

PHI:
[ 0, 14.01 ]

.
 ------------------------------------------------

 LIST OF BOXES CONTAINING VERIFIED FEASIBLE POINTS:

 Box no.: 1
 Box coordinates:
[ -.0003163, 1.001 ]

PHI:
[ 0, 14.01 ]

.
 ------------------------------------------------
 ALGORITHM COMPLETED WITH LESS THAN THE MAXIMUM NUMBER,
       20000  OF BOXES.
.
 Total number of boxes processed in loop:  2
 BEST_ESTIMATE: 11
 Overall CPU time: .01001
.
```

GlobSol also has a facility for storing up to 10 constants that can be reset during execution of GlobSol. For example, to set constant 1 to $[2,3]$, one would place `CALL SET_CONSTANT(1,IVL(2,3))` in the `*.f90` program defining the problem, but would refer to that constant with the global variable `SETTABLE_CONSTANT(1)` (of type

[1]Finer control over the stopping tolerances can be obtained by adjusting particular variables in the `GlobSol.CFG` file. These tolerances are described in [11], and are documented in-line in `GlobSol.CFG`.

INTERVAL) in routines that executed during the optimization. In the expressions or computations defining the objective and constraints in the *.f90 program defining the problem, that constant would be referenced as C_(1), without declaration. This facility is envisioned mainly for people modifying GlobSol's algorithms and experimenting with them.

5. GlobSol's "CODELIST" Files

The operations performed during execution of the *.f90 problem definition program produce an "unrolled" sequence of operations that we call a "code list." This code list is written to a file, which is then differentiated, optimized[1], and used within GlobSol to evaluate the objective, constraints, and derivatives. The code list is very similar to the DAG produced in the "COCONUT" environment [19]. The basics of construction of the code list appear in [7, p. 83 ff.], with operation codes listed in [7, Table 2.6, p. 84]. The code list in the present GlobSol distribution has the same basic structure, with some additional operations added, etc. For example, running "`globsol mixed 1`" creates the files `mixed.CDL`, `mixedG.CDL`, and `mixedGO.CDL`, representing the code list, derivative code list, and optimized derivative code list corresponding to `mixed.f90`. In the present distribution of GlobSol, only a binary representation of code lists is available. However, once the present experimental version is released, there will be an option to store small code lists in ASCII format.

6. The GlobSol Script

The "`globsol`" command used in earlier examples in this document is an operating system command line script that successively calls components of the GlobSol package. In particular, the command "`globsol <filename> ?`" does the following, in sequence.

(1) compiles the file `<filename>.f90` to produce an executable;
(2) executes `<filename>` to produce the code list `<filename>.CDL`;
(3) executes the GlobSol program "`makegrad`" to produce a code list "`<filename>G.CDL`" that contains instructions for computing partial derivatives of the objective and constraints.
(4) executes the GlobSol program "`optimize_codelist`" to remove redundant operations from `<filename>G.CDL` and place the resulting code list in `<filename>GO.CDL`;
(5) renames `<filename>GO.CDL` to `<filename>G.CDL`;
(6) runs the main GlobSol optimization program "`find_global_min`, which uses the files `GlobSol.CFG`, `<filename>G.CDL`, and `<filename>.DT?` to produce the output files `<filename>.OT?`, `<filename>_verified_list.TXT`, `<filename>_unverified_list.TXT`, and, optionally, `OPTTEST.TBL`.

Here, the `optimize_codelist` process in step (4) is an $\mathcal{O}\left(\nu^2\right)$ process, where ν is the number of operations in a single evaluation of the objective, constraints, and gradients thereof. Thus, for complicated functions, such as functions whose values depend on accumulations within a long-running loop, the `optimize_codelist` step may be a limiting factor. In such cases, it may be useful to remove this step.

[1]This means that redundant operations are removed.

Highlights of the GlobSol directory tree are as follows.

(1) The root directory <`GlobSol root dir`> contains
a) the `makefile`,
b) license information files[1],
c) Fortran module information files[2],
d) the documentation files `install.html` and `install-win-gnu.html`,
e) the Gnu make program `mingw32-make.exe`, and
f) the scripts `globsol`, `rg`, etc.

(2) The directory <`GlobSol root dir`>`/integration_test_data` contains a copy of `GlobSol.CFG` with default settings, and also contains a number of sample problem files, such as `mixed.f90` and `mixed.DT1`.

(3) <`GlobSol root dir`>`/executables` contains the executable codes, as well as the library[3] for linking with GlobSol components.

(4) <`GlobSol root dir`>`/f90intbi` contains the source code for the overall branch and bound algorithm in file <`GlobSol root dir`>`/f90intbi/rigorous_global_search.f90`, as well as initialization and completion routines.

(5) <`GlobSol root dir`>`/intlib.alt` contains a slightly modified version of the `INTLIB` library [12] for interval arithmetic.

(6) <`GlobSol root dir`>`/linpack.alt` contains the LINPACK library [4] for solving dense linear systems of equations on serial machines.

(7) <`GlobSol root dir`>`/minpack.alt` contains the MINPACK-1 [14] routines for local unconstrained approximate optimization and local approximate root-finding.

(8) <`GlobSol root dir`>`/overload` contains the modules used throughout GlobSol. Some of these are as follows.
a) `intarith.f90` contains a modified version of the `INTERVAL_ARITHMETIC` module described in [6].
b) `overload.f90` contains the source for module `CODELIST_CREATION`, used in creating code lists from `*.f90` problem definition files.
c) `optimize_codelist.f90` contains the main program for removing redundant operations from a code list.
d) `makegrad.f90` contains the driver program for symbolically differentiating a code list, while `gradvars.f90` contains the source for module `GRADIENT_VARIABLES`, which defines how the differentiation is done for each operation.
e) `prints.f90` contains the source for module `PRINT_ROUTINES`, where the generic `PRINT_VECTOR` for general printing in GlobSol is defined.
f) Other modules in <`GlobSol root dir`> define data structures and global variables used throughout GlobSol routines.

(9) <`GlobSol root dir`>`/function` contains subdirectories with the source code for functions that evaluate the objective, constraints, derivatives, second-order derivatives, and Fritz John matrices by interpreting the code list.

(10) In the version of GlobSol distributed in executable form with the `g95` compiler, <`GlobSol root dir`>`/IO` contains the "C" source for David Gay's

[1]Such as `BOOST_license.txt` and `gnu_Fortran_95_and_gnu_make_license.txt`
[2]for most builds
[3]`GlobSol.a` on Unix builds and on the build distributed for the `g95` compiler, and `GlobSol.lib` on other MS-Windows builds

gdtoa routines, as well as Fortran source for interfaces to gdtoa for use with GlobSol's output of intervals.

(11) <GlobSol root dir>/symbolic contains a module in source file cdllatex.f90 and source code, such as that in make_tex.f90 for producing a LaTeX representation from a code list.

8. GlobSol as a Callable Subroutine Library

As part of the installation process, a subroutine library for GlobSol is constructed and placed in the directory <GlobSol root dir>/executables. (In the default installation, the file in which this library resides is called "GlobSol.a.") All of the components of GlobSol corresponding to source files reside in this directory. These routines can be called by any program compiled with a compatible compiler, and may be linked by including "GlobSol.a" in the list of libraries to be linked[1]. Although not all of the individual components of GlobSol are presently thoroughly documented, examples of use are available by examining the source code for the auxiliary functions listed in Table 1. These source routines are found in the directories "<GlobSol root dir>/evaluate" and "<GlobSol root dir>/symbolic."

9. Towards Interfacing with Other Systems

During December, 2006, Hermann Schichl at the University of Vienna and the author of this guide developed and tested a converter to convert the DAG [19] format[2] from the COCONUT system [15] to GlobSol's code list format. This gives the experimental version of GlobSol access to the large libraries of test problems available in DAG format [20]. Furthermore, the COCONUT system has facilities for conversion of AMPL representations and GAMS representations into DAG format, thus making problems expressed in both the AMPL and GAMS languages available to GlobSol[3].

The author of this guide also sees significant benefit in a tighter interface of GlobSol with MATLAB, and will do so, time permitting. Advantages would be

- direct input and output of initial bounds, tolerances, and lists of bounds on answers;
- defining the problem through MATLAB "m" files with standard syntax (except for branching);
- no need for a Fortran compiler[4] to run GlobSol.

A straightforward interface would include

- a MATLAB user-defined data type for generation of GlobSol code lists[5];
- MATLAB interfaces to the routines
  <GlobSol root dir>/overload/makegrad.f90,
  <GlobSol root dir>/overload/optimize_codelist.f90, and

[1]The Fortran 2003 standardized interface to the "C" language may be useful in this context.
[2]DAG stands for "Directed Acyclic Graph," and is the internal representation of the objective and constraints within the COCONUT system.
[3]provided the COCONUT environment is installed and provided AMPL and GAMS are installed with appropriate licenses.
[4]but a need for MATLAB
[5]Defining this data type can be done by translating the GlobSol module CODELIST_CREATION in file <GlobSol root dir>/overload/overload.f90 into MATLAB syntax.

`<GlobSol root dir>/f90intbi/find_global_min.f90` (with possible modification of the argument lists to these.)

- a MATLAB function[1] to invoke the functions defining the objective and constraints and to invoke the interfaces to GlobSol.

A more ambitious interface would involve use of MATLAB's linear algebra within the operations of GlobSol, and use of INTLAB [17] for GlobSol's interval arithmetic.

10. Other Future Work

Among other improvements, the present experimental version of GlobSol includes use of linear relaxations. This provides extremely significant performance improvement on many problems; see [1, 10]. That version also provides access to Taylor arithmetic through the COSY system; see [9], and uses the IPOPT solver [21] for approximate local optimization. Putting this version into releasable form is the next step.

Acknowledgements

The author of this guide wishes to thank others who have worked on GlobSol and have made it possible. In particular, he wishes to thank his former students, the authors of [12], who also worked on early versions of GlobSol, he wishes to thank Bill Walster and SunSoft for support during development, and he wishes to thank George Corliss for testing, use of early versions, contributions of some of the modules and scripts, and much advice on software development. The author also wishes to thank Arnold Neumaier and the Erwin Schrödinger Institute for support while in Vienna in December, 2006.

References

[1] R. Baker Kearfott, *Discussion and empirical comparisons of linear relaxations and alternate techniques in validated deterministic global optimization*, Optimization Methods and Software 21 (2006), pp. 715–731.

[2] M. Berz et al., *COSY INFINITY and its applications to nonlinear dynamics*, in *Computational Differentiation: Techniques, Applications, and Tools*, M. Berz et al., eds., SIAM, Philadelphia, PA (1996), pp. 363–365.

[3] B. Dawes, D. Frey, and D. Abrahams, *Information about the boost software*, World-Wide Web document (2004), URL `\url{http://www.boost.org/more/license_info.html}`.

[4] J.J. Dongarra et al., *LINPACK Users' Guide*, SIAM, Philadelphia (1979).

[5] R. Fourer, D. Gay, and B. Kernighan, *AMPL A Modeling Language for Mathematical Programming*, Boyd and Frazer, Danvers, Massachusetts (1993).

[6] R.B. Kearfott, *Algorithm 763: INTERVAL_ARITHMETIC: A Fortran 90 module for an interval data type*, ACM Transactions on Mathematical Software 22 (1996), pp. 385–392.

[7] ———, *Rigorous Global Search: Continuous Problems*, no. 13 in Nonconvex optimization and its applications, Kluwer Academic Publishers, Norwell, MA, USA, and Dordrecht, The Netherlands (1996).

[8] ———, *Verified branch and bound for singular linear and nonlinear programs* (2007), submitted.

[9] R.B. Kearfott and A. Arazyan, *Taylor series models in deterministic global optimization*, in *Automatic Differentiation: From Simulation to Optimization*, G. Corliss et al., eds., chap. 44, Computer and Information Science, Springer-Verlag, New York, NY (2001), pp. 365–372.

[10] R.B. Kearfott and S. Hongthong, *Validated linear relaxations and preprocessing: Some experiments*, SIAM J. on Optimization 16 (2005), pp. 418–433.

[11] R.B. Kearfott and G.W. Walster, *On stopping criteria in verified nonlinear systems or optimization algorithms*, ACM Transactions on Mathematical Software 26 (2000), pp. 373–389.

[12] R.B. Kearfott et al., *Algorithm 737: INTLIB: A portable Fortran-77 elementary function library*, ACM Transactions on Mathematical Software 20 (1994), pp. 447–459.

[1]in an "`m`" file

[13] ———, *Libraries, tools, and interactive systems for verified computations: Four case studies*, in *Numerical Software with Result Verification, Lecture Notes in Computer Science no. 2991*, R. Alt et al., eds., Springer-Verlag, New York (2004), pp. 36–63.

[14] J.J. Moré, B.S. Garbow, and K.E. Hillstrom, *User guide for MINPACK-1*, Tech. Rep. ANL-80-74, Argonne National Laboratories (1980).

[15] A. Neumaier, *Complete search in continuous global optimization and constraint satisfaction*, in *Acta Numerica 2004*, A. Iserles, ed., Cambridge University Press (2004), pp. 271–369.

[16] R.E. Rosenthal, *GAMS — A User's Guide* (2008), `http://www.mat.univie.ac.at/~neum/glopt/coconut`.

[17] S.M. Rump, *INTLAB – INTerval LABoratory* (1999–2008), `http://www.ti3.tu-harburg.de/rump/intlab/`.

[18] N.V. Sahinidis, *BARON: A general purpose global optimization software package*, J. Global Optim. 8 (1996), pp. 201–205.

[19] H. Schichl and A. Neumaier, *Interval analysis on directed acyclic graphs for global optimization*, J. of Global Optimization 33 (2005), pp. 541–562.

[20] O. Shcherbina et al., *Benchmarking global optimization and constraint satisfaction codes*, in *COCOS*, C. Bliek, C. Jermann, and A. Neumaier, eds., *Lecture Notes in Computer Science*, vol. 2861, Springer-Verlag (2003), pp. 211–222.

[21] A. Wächter, *Homepage of IPOPT* (2002), `https://projects.coin-or.org/Ipopt`.