MISCLIB: A Library of Support Routines

User's Guide

Barry W. Brown

This work supported by grant CA11672 from the National Cancer Institute.

Contact: Barry W. Brown, (bwbrown@mdanderson.org)

1 Introduction

Misclib is a library of Fortran routines written (or in some cases adapted) by members of the Section of Computer Science of the Department of Biomathematics in the course of creating statistical software. The work was supported in part by grant CA11672 from the National Cancer Institute. This grant is the core grant to M.D. Anderson Cancer Center.

Briefly, the capabilities of this library include the following:

- Naming constants such as sixth = 1/6 and stdin=5.
- Formatting single numbers with a bit more flexibility than is provided by the Fortran edits.
- Printing according to a template.
- Prompting and obtaining user answers – numbers or values such as yes or no.
- Calculating complete beta and gamma and erf functions.
- Prompt for a file name and open the file.
- Permute indices for an array so that the permutation yields a sorted array.
- Lexically analyse a string of characters.
- Sort a one- or two-dimension array.
- Find the zero of a monotone function of one variable.
- Find a maximum of a function of one variable.

NOTE: For simplicity in documentation, we use the old Fortran term DOUBLE PRECISION whereas the code uses

dpkind = KIND(0.0d0)

We recommend the latter practice.

2 Legalities Regarding Use of this Code

We place our efforts in writing this package in the public domain. However, code from ACM publications is subject to the ACM policy (below). evan though we may have modified the packaging of these routines.

The routines in **mathlib_mod** are from publications of the ACM, sometimes with the interface slightly changed. The complete beta and gamma functions are from the two articles by DiDinato and Morris; the cumulative error function (erf) is from Cody. The core of the zero finder routine is from the article by Alefeld et al. The core of the zero finder routine is from the article by Alefeld et al. The algorithm for the maximum of a function of a single variable is from Brent (and was not published in an ACM journal).

References

Incomplete Beta

DiDinato, A. R. and Morris, A. H. (1993) "Algorithm 708: Significant Digit Computation of the Incomplete Beta Function Ratios." ACM Trans. Math. Softw. 18, 360-373.

Incomplete Gamma

DiDinato, A. R. and Morris, A. H. (1986) "Computation of the incomplete gamma function ratios and their inverse." ACM Trans. Math. Softw. 12, 377-393.

Erf

Cody, W.D. (1993). "ALGORITHM 715: SPECFUN - A Portable FORTRAN Package of Special Function Routines and Test Drivers" ACM Trans. Math. Softw. 19, 22-32.

Finding a Zero of a Monotone Function

Alefeld, G. E., Potra, F. A., Shi, Y. (1995) "Algorithm 748: Enclosing Zeros of Continuous Functions.", by G. E. Alefeld, F. A. Potra, YiXun Shi, ACM Trans.

Math. Softw., 21, No. 3, 327-344

Maximum of a Function

Forsythe, George E., Malcolm, Michael A., and Moler, Cleve B. ”Computer Methods for Mathematical Computations”, Prentice Hall, Englewood Cliffs, NJ page 185. they in turn credit the algorithm to Richard Brent, ”Algorithms for Mminimization without Derivatives”, Prentice Hall (1973).

ACM Policy on Use of Code

> Submittal of an algorithm for publication in one of the ACM Transactions implies that unrestricted use of the algorithm within a computer is permissible. General permission to copy and distribute the algorithm without fee is granted provided that the copies are not made or distributed for direct commercial advantage. The ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.
>
> Krogh, F. (1997) “Algorithms Policy.” ACM Tran. Math. Softw. 13, 183-186.

We do not know the policy of the Royal Statistical Society; they have discontinued publishing algorithms. However, they made a number of these programs available on Statlib on condition that there be no charge for their distribution.

NO WARRANTY

WE PROVIDE ABSOLUTELY NO WARRANTY OF ANY KIND EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE

3 How to Install this Library

These instructions pertain to installing as per the distributor of the code's model in which all routines are included in a single Fortran library called **misclib.a**.
The file **COMPILE.IT** should be modified (if necessary) and executed. The name of the Fortran compiler in this file is **g95**; if you use a different compiler this string in the file should be changed (everywhere) to the name of your compiler. The **ar** (archive) command creates the library from the .o (object) files in g95 (and in Unix). The files whose names contain the characters **_mod** are Fortran MODULE programs that must be USEd to be seen by the compiler. Fortran 95 compilers generally create a file separate from the .o file that is use by the compiler in a subsequent USE. In many compilers. this file has the suffix **.mod** and there is usually a compiler option for specifying the path for the .mod files (the I option in g95). Thus, it might be convenient to place all of the .mod files in a single directory.

4 biomath_constants_mod

4.1 Function.

Defines a number of named constants. All are dpkind (double precision) except **dpkind, spkind, stdin,** and **stdout**.

4.2 Details

- **INTEGER, PARAMETER :: dpkind = KIND(0.0D0)**
- **INTEGER, PARAMETER :: spkind = KIND(0.0)**
- **INTEGER, PARAMETER :: stdin = 5, stdout = 6**
- **REAL (dpkind), PARAMETER :: eight = 8.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: five = 5.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: four = 4.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: hundred = 100.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: nine = 9.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: one = 1.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: seven = 7.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: six = 6.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: sixth = one/six**
- **REAL (dpkind), PARAMETER :: ten = 10.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: tenth = one/ten**
- **REAL (dpkind), PARAMETER :: thousand = 1000.0E0_dpkind**

- **REAL (dpkind), PARAMETER :: thousandth = one/thousand**
- **REAL (dpkind), PARAMETER :: three = 3.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: twelve = 12.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: two = 2.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: zero = 0.0E0_dpkind**
- **REAL (dpkind), PARAMETER :: eighth = one/eight**
- **REAL (dpkind), PARAMETER :: fifth = one/five**
- **REAL (dpkind), PARAMETER :: fourth = one/four**
- **REAL (dpkind), PARAMETER :: half = one/two**
- **REAL (dpkind), PARAMETER :: hundredth = one/hundred**
- **REAL (dpkind), PARAMETER :: third = one/three**

5 format_number_mod

Contains one generic routine that converts a number (of type real, double precision, or integer) to a character string. The number can be left, right, or center justified. The number can be written either in F format (e.g., 99.99) or E format (e.g., 9.999E1) depending on the size of the number.

5.1 Calling Sequence

For a real or double precision number, X, the calling sequence is:

```
CALL format_number(x,justi,width,maxf,minf,ndecf,npe,ndece, &
                   & qpad,charx,qfit)
```

For an integer number, X, many of these arguments make no sense, so the simplified calling sequence is:

CALL format_number(x,justi,width,charx,qfit)

5.2 Arguments

- **<type>, INTENT(IN):: X** The numeric value to be converted to a character string. <**type**> must be one of integer, real, or double precision (or equivalent via a KIND expression).
- **INTEGER, INTENT(IN):: justi** Justification of the character string representing the number.
  - -1 – Left justified
  - 0 – Centered
  - 1 – Right justified
- **INTEGER, INTENT(IN):: width** The width of the string representing the value of **x**.
- **<type>, INTENT(IN):: maxf**. Numbers with absolute values greater than **maxf** will be converted in E format. <type> must be the same as for **x**. This argument is not present for integer **x**.
- **<type>, INTENT(IN):: minf**. Numbers with absolute values less than **minf** will be converted in E format. <type> must be the same as for **x**. This argument is not present for integer **x**.

  Note: Values of X between **minf** and **maxf** are converted by an F format.
- **INTEGER, INTENT(IN):: ndecf** The number of digits after the decimal point in the F format. This argument is not present for integer **x**.
- **INTEGER, INTENT(IN):: npe** Scale factor for exponential display. This argument is not present for integer **x**.

- **INTEGER, INTENT(IN):: ndece** The number of digits after the decimal point in the E format. This argument is not present for integer **x**.
- **LOGICAL, INTENT(IN):: qpad** IF TRUE, there will be exatly NDECF or NDECE digits after the decimal point. If FALSE, all right-most zeros will be deleted. This argument is not present for integer **x**.
- **CHARACTER(LEN=*), INTENT(OUT):: charx**. The character string to receive the value of **x**. Should be at least **width** long.
- **LOGICAL, INTENT(OUT):: qfit** TRUE if the character value of **x** fits in a string of length **width** else FALSE.

6 format_specs – Change a template file into one or more Fortran FORMAT statements

This functionality is provided for Fortran programmers who would rather lay out a complex page format using a text editor rather than to manually figure out a Fortran format.

6.1 Calling Sequence

This is a Perl script called with the statement

format_specs <file list>

Each file, infile, in <file list> is processed and the resulting Fortran code is written to a file, 'infile.fmt'.

6.2 Description

All lines in infile are ignored until a line beginning with the characters '>>BEGIN' is seen. Processing starts on the next line and continues until either a line beginning with the characters '>>END' or the characters '>>CONTINUE' is seen. A processed continguous block of lines produces a Fortran format and some auxillary code. The format goes into the character string called **message_format**. A '>>CONTINUE' line causes the previous block to terminate and a new one to be begun. This function is for pages too long or complex to produce with a single 'WRITE' statement.

Processing of a line consists of adding the contents of the line to a Format statement verbatim except of strings of the '%' character. These strings are to be replaced by an appropriate character string at run time.

6.3 Example

Here are the contents of the input file, infile.

```
Below is a ruler to show columns.  It (and everything not between >>BEGIN and
>>END is ignored by format_specs.

00000000011111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

>>BEGIN
Column1                    Column2                       Column3
Header                     Header                        Header
>>END

>>BEGIN
%%%%%%%%                   %%%%%%%                       %%%%%%%
>>END
```

The format_specs program converts this into the following fragment of Fortran code in file, **infile.txt.**

```
USE print_it
always_print = .TRUE..
num_subs = 0
message_format = "(&
&5X,'Column1                  Column2                      Column3'/&
&5X,'Header                   Header                       Header')"
CALL print_message_format()
always_print = .TRUE.
num_subs = 3
sub_pos(1:6) = (/ 6, 13, 31, 37, 61, 67 /)
! Assign sub_strings(1:3)
! sub_string_len = 8
sub_strings(1)
! sub_string_len = 7
sub_strings(2)
! sub_string_len = 7
sub_strings(3)
message_format = "(&
&5X,'%%%%%%%%                 %%%%%%%                      %%%%%%%')"
CALL print_message_format()
```

The module **print_it** is used with this output; it contains the routine **print_message_format** and definitions for **sub_strings** among other things. The first call to **print_message_format** writes the header lines moved right by five characters. The programmer should change the code to place character values in **sub_strings(1), sub_strings(2),** and **sub_strings(3)** before invoking the second call to **print_message_format**. This change must modify the lines involving **sub_strings** into legal Fortran statments. Presumably, the second call prints values of a table and so will be invoked within a **do** loop. An example of a Fortran statement that sets the value of the **sub_strings** is

WRITE (substrings,'(F8.2/I7/I7)') val, intval1, intval2

7 Obtain Numbers from User

7.1 Call

This routine is generic for type and scalar versus singly dimensioned array. The calling sequence for obtaining a scalar is:
SUBROUTINE get_numbers(format, lo, lo_eq_ok, hi, hi_eq_ok, x, &
failed)
The calling sequence for obtaining the contents of a singly dimensioned array is
SUBROUTINE get_numbers(format, lo, lo_eq_ok, hi, hi_eq_ok, x, &
number_wanted,failed)
The array version has an argument, **number_wanted**, that the scalar version lacks.

7.2 Function

The routine prints a prompting message to standard output and accepts one or more numbers as an answer. These numbers are optionally range checked. Errors cause a repeat of the entire process. After three unsuccessfum tries, the routine exits with an error.
There are versions of get_numbers for integer, real, and double precision values. We use the symbol ¡TYPE¿ to indicate one of the words, 'INTEGER', 'REAL', or 'DOUBLE PRECISION'. The same word must be used for all arguments in the call to the routine.

7.3 Arguments

- CHARACTER (LEN=*), INTENT(IN):: **format**. A Fortran format that is written to standard output to prompt the user to enter the numbers.

- ¡TYPE¿. INTENT(IN), OPTIONAL:: **LO**. Low values for range checking user supplied numbers. This version is for obtaining a scalar.

  ¡TYPE¿. INTENT(IN), DIMENSION(:), OPTIONAL:: **LO**. This version is used when obtaining a singly dimensioned array.

  The routine checks that the i'th number input is greater than or equal to lo(i) if lo has dimension greater than one. If array lo has dimension 1, then all input numbers are checked against this single value.

- LOGICAL, OPTIONAL, INTENT(IN):: **lo_eq_ok**. If TRUE then it is okay if the answer equals the specified low bound. IF FALSE then this is an error. Default if not presentTRUE.

- ¡TYPE¿. INTENT(IN), OPTIONAL:: **hi**. High values for range checking user supplied numbers. This version is for obtaining a scalar.

  ¡TYPE¿. INTENT(IN), DIMENSION(:), OPTIONAL:: **hi**. This version is used when obtaining a singly dimensioned array.

  The routine checks that the i'th number input is less than or equal to hi(i) if lo has dimension greater than one. If array hi has dimension 1, then all input numbers are checked against this single value.

- LOGICAL, OPTIONAL, INTENT(IN):: **hi_eq_ok**. If TRUE then it is okay if the answer equals the specified upper bound. IF FALSE then this is an error. Default if not present TRUE.

- ¡TYPE¿, INTENT(OUT):: **x**. Variable in which to store the number. This is the scalar case; the singly dimensioned array case is ¡TYPE¿, INTENT(OUT), DIMENSION(:):: **x**

- INTEGER, INTENT(IN), OPTIONAL:: **number_wanted**. The number of values to be obtained from the user. If absent from the calling sequence, SIZE(x) is used.

- LOGICAL, INTENT(OUT), OPTIONAL:: **failed**. If TRUE on return, the routine failed to get the numbers. If FALSE, the routine succeeded.

  If the argument is missing, the routine will STOP with an error message if it does not obtain the desired numbers.

8 Choose a Character from a List

8.1 Call

SUBROUTINE get_character(mssg,chars,num_chars,which,qerr)

8.2 Function

The routine allows the user to be prompted to choose one of a list of characters. Prints a prompting message specified by **mssg** and reads a character string which is converted to lower case. The first non-blank character of the string is matched against each character in **chars**. Returns the ordinal number of the character matched in **which**.

8.3 Arguments

- **CHARACTER(LEN=*), INTENT(IN):: mssg** A Fortran format that will be printed to the user prior to asking for the choice.

- **CHARACTER(LEN=*), INTENT(IN):: chars** A character string containing one of each character that can be chosen. **Note: chars** should all be lower case.

- **INTEGER, INTEENT(IN);; num_chars**. The number of initial characters in **char** which should be examined for equality with the input character.

- **INTEGER, INTENT(OUT):: which** The index of the first character in **chars** equal to the input character. Undefined if **qerr** is TRUE.
- **LOGICAL, INTENT(OUT):: qerr**. TRUE if a match is not achieved in 3 tries, else FALSE

.

9 Obtain Numbers from User

9.1 Call

This routine is generic for type and scalar versus singly dimensioned array. The calling sequence for obtaining a scalar is:
SUBROUTINE get_numbers(format, lo, lo_eq_ok, hi, hi_eq_ok, x, &
failed)
The calling sequence for obtaining the contents of a singly dimensioned array is
SUBROUTINE get_numbers(format, lo, lo_eq_ok, hi, hi_eq_ok, x, &
number_wanted,failed)
The array version has an argument, **number_wanted**, that the scalar version lacks.

9.2 Function

The routine prints a prompting message to standard output and accepts one or more numbers as an answer. These numbers are optionally range checked. Errors cause a repeat of the entire process. After three unsuccessfum tries, the routine exits with an error.
There are versions of get_numbers for integer, real, and double precision values. We use the symbol ¡TYPE¿ to indicate one of the words, 'INTEGER', 'REAL', or 'DOUBLE PRECISION'. The same word must be used for all arguments in the call to the routine.

9.3 Arguments

- CHARACTER (LEN=*), INTENT(IN):: **format**. A Fortran format that is written to standard output to prompt the user to enter the numbers.
- ¡TYPE¿. INTENT(IN), OPTIONAL:: **LO**. Low values for range checking user supplied numbers. This version is for obtaining a scalar.

  ¡TYPE¿. INTENT(IN), DIMENSION(:), OPTIONAL:: **LO**. This version is used when obtaining a singly dimensioned array.

  The routine checks that the i'th number input is greater than or equal to lo(i) if lo has dimension greater than one. If array lo has dimension 1, then all input numbers are checked against this single value.
- LOGICAL, OPTIONAL, INTENT(IN):: **lo_eq_ok**. If TRUE then it is okay if the answer equals the specified low bound. IF FALSE then this is an error. Default if not presentTRUE.
- ¡TYPE¿. INTENT(IN), OPTIONAL:: **hi**. High values for range checking user supplied numbers. This version is for obtaining a scalar.

  ¡TYPE¿. INTENT(IN), DIMENSION(:), OPTIONAL:: **hi**. This version is used when obtaining a singly dimensioned array.

  The routine checks that the i'th number input is less than or equal to hi(i) if lo has dimension greater than one. If array hi has dimension 1, then all input numbers are checked against this single value.
- LOGICAL, OPTIONAL, INTENT(IN):: **hi_eq_ok**. If TRUE then it is okay if the answer equals the specified upper bound. IF FALSE then this is an error. Default if not present TRUE.
- ¡TYPE¿, INTENT(OUT):: **x**. Variable in which to store the number. This is the scalar case; the singly dimensioned array case is ¡TYPE¿, INTENT(OUT), DIMENSION(:):: **x**

- INTEGER, INTENT(IN), OPTIONAL:: **number_wanted**. The number of values to be obtained from the user. If absent from the calling sequence, SIZE(x) is used.

- LOGICAL, INTENT(OUT), OPTIONAL:: **failed**. If TRUE on return, the routine failed to get the numbers. If FALSE, the routine succeeded.

  If the argument is missing, the routine will STOP with an error message if it does not obtain the desired numbers.

10 Obtain a Character String from the User

10.1 Call

CALL get_string(fmt,nline,string,qnulok,qfail)

10.2 Function

Prompts the user to enter a character string then returns that string. Lines starting with '#' are ignored as are any characters following a '#' in the middle of a line.

10.3 Arguments

- **CHARACTER(LEN=*), INTENT(IN):: fmt**. A Fortran format in an array of character strings. The user prompt will be provided by WRITE (*,fmt(i)) for each line i in fmt.

- **INTEGER, INTENT(IN):: nline**. The number of elements of **fmt** to be processed.

- **CHARACTER(LEN=*), INTENT(OUT):: string**. The string entered by the user.

- **LOGICAL, INTENT(IN):: qnulok**. If TRUE then a null line (all blanks or comments) is acceptable. A null line causes **string** to be set to blanks. If FALSE, an error message is emmitted and the process repeated.
- **LOGICAL, INTENT(OUT):: qfail**. TRUE if three attempts at prompting and reading failed. Otherwise FALSE.

11 Get a Yes or No Answer

11.1 Call

CALL get_yn(mssg,qyes,qerr)

11.2 Arguments

- **CHARACTER(LEN=*), INTENT(IN):: mssg**. A Fortran format that is printed to the user as a prompt for the yes or no answer.
- **LOGICAL, INTENT(OUT):: qyes**. TRUE if the answer yes (or y) was obtained, FALSE if the answer no (or n) was obtained.
- **LOGICAL, INTENT(OUT):: qfail**. TRUE if 3 attempts did not yield a yes or no answer. In that case, **qyes** is undefined. Otherwise FALSE.

12 Miscellaneous Mathematical Functions

12.1 Real Error Function

12.1.1 Definition

FUNCTION erf(x)

12.1.2 Argument

DOUBLE PRECISION, INTENT(IN):: x

x is the upper limit of integration of the error function defined by

$$\frac{2}{\sqrt{2\pi}} \int_0^x e^{-t^2} dt$$

12.2 Complement of the Error Function

12.2.1 Definition

FUNCTION erfc1(ind,x)

12.2.2 Arguments

INTEGER, INTENT(IN):: ind

DOUBLE PRECISION, INTENT(IN):: x

If ind==0 then returns the complement of the error function

$$erfc(x) = \frac{2}{\sqrt{2\pi}} \int_x^\infty e^{-t^2} dt$$

If ind is not 0, the function returns

$$e^{x^2} erfc(x)$$

12.3 Logarithm of the Complete Beta Function

12.3.1 Definition

FUNCTION log_beta(a0,b0)

12.4 Arguments

DOUBLE PRECISION, INTENT(IN):: a0, b0

$a0$ and $b0$ are the parameters of the complete beta function which is defined as:

$$\int_0^1 t^{a0-1}(1-t)^{b0-1}dt$$

12.5 Logarithm of the Complete Gamma Function

12.6 Definition

FUNCTION log_gamma(a)

12.7 Argument

DOUBLE PRECISION, INTENT(IN):: a

a is the parameter of the complete gamma function defined by

$$\int_0^\infty t^{a-1}e^{-t}dt$$

12.8 EXP(X)-1

12.8.1 Definition

FUNCTION rexp(x)

12.8.2 Argument

Computes $\exp(x)-1$ accurately. Doing the obvious leads to catastrophic cancellation near 0.

13 open_file – Get file name from user and open it

13.1 Calling Sequence

FUNCTION open_file(mssg,read_only,appendable,delimiter,ierr)

13.2 Function

Queries the user for a file name and opens it.

13.3 Arguments

- INTEGER:: open_file. The function's return value. If the function succeeded in eliciting an appropriate file name from the user and opening it, the funtion returns the unit number associated with the file. Otherwise, the return value is -1.

- CHARACTER(LEN=*), OPTIONAL, INTENT(IN):: mssg. A Fortran format yielding the message to be displayed to the user before he or she is prompted to enter a file name. If missing or blank, no message is displayed.

- LOGICAL, OPTIONAL, INTENT(IN):: read_only. A logical flag indicating if the file should be opened for reading only. The default if missing is .FALSE., indicating that the file should be opened for both reading and writing.

- LOGICAL, OPTIONAL, INTENT (IN) :: appendable A logical flag indicating if the user is allowed to specify that the file should be opened for appending. The default if missing is .TRUE., indicating that the user may specify that the file be opened for appending. This argument is only meaningful if READ_ONLY is .FALSE. or missing.

- CHARACTER (LEN=*), OPTIONAL, INTENT (IN) :: delimiter.. A character string indicating the delimiter character which should be used for list directed or NAMELIST character I/O. The default value if missing is 'none'. Note that this argument is exactly the same as the I/O specifier DELIM. Allowable values are 'apostrophe', 'quote', or 'none'. DELIMITER is not case-sensitive.

- INTEGER, OPTIONAL, INTENT (OUT) :: ierr Integer error code. On failure, if IERR is present, IERR is set to a non-zero value from the list below and the funtion returns -1; if IERR is absent, an error message is issued to STDOUT and the code STOPs. On success, IERR is set to zero if it is present and the function returns the unit associated with the file.

  Values of ierr and their meaning are:

  - -2 Illegal delimiter specification.
  - -1 No units available.
  - 0 No error; the file is open and associated with the unit returned as open_file.
  - 1 There were MAX_BAD failures getting a file name from the user.
  - 2 The user chose to abort, or entered "quit" when prompted for a file name.
  - 3 The user entered "back" when prompted for a file name.

14 permutation_sort_mod

There are two generic routines in this module. The first is **permutation_sort_matrix** that changes an identity index to a permutation corresponding to sorting the columns of a matrix in ascending order. The matrix may be or type character, integer, real, or double precision.

To illustrate this, suppose that the columns of the matrixhave only one row and the four columns have real values

$$(3.0, 2.5, 1.6, 4.9).$$

Before **permutation_sort_matrix** is called, a singly dimensioned array, **index**, is the identity

$$(1, 2, 3, 4).$$

After the call, the matrix is unchanged, but **index** is $(3, 2, 1, 4)$. This indicates that the smallest value of the array is in column 3, the next smallest in column 2, and so on; the largest value is in column 4. **Note:** The routines provided here perform a permutation sort only on two dimension arrays, not on singly dimensioned arrays. Singly dimensioned arrays can be faked by giving a row dimension of 1.
There are generally two reasons for using a permutation sort rather than the usual sort. One is efficiency. The usual sort interchanges rows of the matrix; the permutation sort only interchanges the single elements of the index vector. The second reason is that it is fairly common to want to sort something by something else. This is easily done given a permutation sort on the values of something else.

14.1 Calling Sequence

CALL permutation_sort_matrix(a,ncol,a_gt_b,nrowdm,irow,index)

14.2 Arguments

- **<type>:: a(ncol)** where **a** is the array whose permutation for sorting is to be determined and <type> is one of real (single precision), double precision, or **character(len=*)**.

- **ncol**. The dimensioned number of columns of the array to be sorted.

- **logical, optional:: a_gt_b.** If present, a logical function

  logical function a_gt_b

  that returns .TRUE. if column **a** is greater than column **b** in the desired sort order. The interface is:

  FUNCTION a_gt_b(a,b,irow)

  The argument irow is passed from the call to **sort_matrix** and may be used as the user wishes. If there is no user supplied routine, the columns are sorted in ascending order of the values in row **irow.**

- **integer, intent(in);;nrowdm.** The row dimension (first dimension) of **a**.
- **integer, intent(in)::irow** An auxillary integer argument passed to the function **a_gt_b**.
- **integer, intent(inout);;index** The singly dimensioned array to be permuted as per the values of the rows of **a**. Almost always contains the integers $1 \ldots nrowdm$.

15 qlex: Lexical Analyser

15.1 Calling Sequence

FUNCTION qlex(string,qstart,curtyp,cval,lcval,ival,dval,indov)

15.2 Function

Each call returns one lexical element of the character string **string**. Tokens may be separated by blanks – the blanks are not returned as separate tokens. Successive calls to qlex yield successive tokens. The types of the elements returned are as follows:

- **IN** Integer.
- **RL** Real number.

  Both integers and reals can begin with a sign or a digit; a real can also begin with a decimal point. The sign, if any, can be separated from the following characters by blanks. The fol- lowing characters can be only digits for an integer. A real can have with a decimal point, '.', followed by a fractional part and an optional exponential part. The exponential part has the form E (or D) followed by a number. Both types of tokens are terminated by being followed by a blank character, an operator character, a delimiter, or a character not defined to QLEX (the latter is usually an error). NOTE: Spaces between the sign and the first character of the remainder of the number are removed in CVAL.

- **ID** Alphanumeric name or identifier.

  The token must begin with a letter. Succeeding characters can be letters, digits, or one of the characters '$', '.', or '_'. The token is terminated by being followed by a blank character, an operator character, a delimiter, or a character not defined to QLEX (the latter is usually an error).

- **QS** Quoted character string.

  A quoted string is surrounded by the same quote delimiters, which can be either a single or double quote (' or "). The appearance of the quote delimiter within a string is indicated by its doubling. Thus, the string 'don"t' is recognized as the word "don't". Any characters whatsoever may be in a quoted string, even characters not recognized by QLEX. The quoted string returned by QLEX in CVAL does not include the surrounding single or double quotes. Doubled occurrences of the quoting character within the string are replaced by a single occurrence within CVAL.

- **DL** Delimiter.

  One of the characters ',', '/', '(', ')', '', '', '/[', '/]', ':', ';'.

- **OP** String of operator symbols.

  An operator symbol is one of '+', '-', '*', '/'.

- **UC** Unrecognized Characters.

  An unrecognized character is one not appearing in some definition above.

15.3 Arguments

- LOGICAL:: **qlex**. Is returned TRUE if another token is found; FALSE means that all tokens have been obtained from the string.
- CHARACTER(LEN=*), INTENT(IN):: **string**. The string to be analysed.
- LOGICAL, INTENT(INOUT):: **qstart**. If TRUE, the lexical analysis begins at the first character of string and qlex changes the value to FALSE. If FALSE on input, the analysis begins one character following the last token returned.
- CHARACTER(LEN=2), INTENT(OUT):: **curtyp**. The two character abrieviation for the returned token type as shown in the table above. E.g., IN for integer, RL for a real number, etc. Defined only if qlex is returned TRUE.
- CHARACTER(LEN=*), INTENT(OUT):: **cval**. The character string composing the currently found token. Defined only if qlex returns TRUE.
- INTEGER, INTENT(OUT):: **lcval**. The (logical) length of cval. Defined only if qlex returns TRUE.
- INTEGER, INTENT(OUT):: **ival**. The integer value of the current token. Defined only if qlex is TRUE and curtyp is 'IN' or "RL'. If curtyp is 'RL' and indov==0 then contains the integer part of the value of the token..
- DOUBLE PRECISION, INTENT(OUT):: **dval**. The double precision value of the current token. Defined only if qlex is TRUE and curtyp is 'RL'.
- INTEGER, INTENT(OUT):: **indov**. Indicator of an overflow condition..
  - 0 No overflow – ival and dval defined.
  - 1 The integer value of the token exceeds the range of the default machine integer but the double precision value is okay. ival undefined; dval defined.

  – 2 Value outside the range of a double precision number. Caused by the exponent part being too large.

16 sort_mod

There are two generic routines in this modyle. The first is **sortlist** and sorts singly dimensioned arrays of type character, integer, real, and double precision.
The second generic routine is called sort_matrix and sorts columns (i.e., rearranges first dimension indices) of two dimension arrays. The types sorted can again be character, integer, real, and double precision.

16.1 sort_list

16.1.1 Calling Sequence

CALL sort_list(a,ncol,a_gt_b)

16.2 Function

A generic function that sorts a singly dimensioned array in ascending order according to an optional user supplied routine defining the order. If the optional routine is not supplied, the sort order is the natural one (alphabetic or numeric).

16.3 Arguments

- **¡type¿:: a(ncol)** where **a** is the list to be sorted.
- **INTEGER, INTENT(IN):: ncol** The dimension (size) of **a**.
- **logical, optional:: a_gt_b.** If present, a logical function

  logical function a_gt_b

  that returns .TRUE. if column **a** is greater than column **b** in the desired sort order. The interface is:

FUNCTION a_gt_b(a,b)

where **a** and **b** are elements of **a**.

16.4 sort_matrix

16.5 Calling Sequence

CALL sort_matrix(a,ncol,a_gt_b,nrowus,nrowdim,irow)

16.6 Arguments

- **¡type¿:: a(ncol,nrowdim)** where **a** is the list to be sorted.
- **INTEGER, INTENT(IN):: ncol** The column dimension of **a**.
- **logical, optional:: a_gt_b.** If present, a logical function

  logical function a_gt_b

  that returns .TRUE. if column **a** is greater than column **b** in the desired sort order. The interface is:

  FUNCTION a_gt_b(a,b)

  where **a** and **b** are elements of **a**.

  The argument irow is passed from the call to **sort_matrix** and may be used as the user wishes. If there is no user supplied routine, the columns are sorted in ascending order of the values in row **irow.**
- **¡type¿:: a(ncol,nrowdim)** shows the dimension of matrix **a** where **a** is the array to be sorted and ¡type¿ is one of real (single precision), double precision, or **character(len=*)**.

- **logical, optional:: a_gt_b.** If present, a logical function

  logical function a_gt_b(a,b)

  that returns .TRUE. if column **a** is greater than column **b** in the desired sort order. The interface is: FUNCTION a_gt_b(a,b,irow)

  The argument irow is passed from the call to **sort_matrix** and may be used as the user wishes. If there is no user supplied routine, the columns are sorted in ascending order of the values in row **irow.**

- **INTEGER, OPTIONAL, INTENT(IN):: nrowus** The number of rows of **a** that contain useful data. When the positions of two columns are swapped, only rows (elements) 1:nrowus are actually swapped. If ommitted, the value is set to **nrowdm**.

- **INTEGER, INTENT(IN)::nrowdim** The row dimension of **a**.

- **INTEGER, INTENT(IN)::irow**. An integer passed to **a_gt_b**.

17 Finding the Zero of a Monotone Function

There are four routines for finding a zero (or other specified value) of a monotone function of one variable. Two routines are provided an initial guess as to the answer and step up and down until the answer is bounded then tighten the interval. These two routines have the word 'step' as part of their name. The other two routines are provided with a lower and upper bound on the answer and refine the interval. These routines have the word 'interval' in their name.

One of the two step or interval routines uses reverse communication and the routine name starts with 'rc'. In these routines, there is a status variable one of whose values indicates that a function evaluation is to be performed by the calling routine at a value x supplied by the 'rc' routine. An argument, 'fx', should have its value set to the value of the function and the 'rc' routine called again. Reverse communication is useful in cases in which there is program data and structures needed to evaluate the function and the code is cleaner if these items are not made visible at a level within the zero-finder.

The routines whose name does not begin with 'rc' take the usual direct communication form. The name of the function to be evaluated is passed as an argument and the routine takes care of calling it.

17.1 Reference

The algorithm is described in the article: Algorithm 748: Enclosing Zeros of Continuous Functions, by G. E. Alefeld, F. A. Potra, YiXun Shi, *ACM Transactions on Mathematical Software, Vol. 21, No. 3*, Sep. 1995 pages 327-344 and code from this article is used here. The packaging has been changed from that in the article.

17.1.1 Setting Values for the Zero-Finder Routines.

A call to set_zero_finder is *required* before invoking any of the specific routines. This routine sets convergence and step-size criteria for the zero-finding routines. Note that all arguments are optional and take default values if not set by the calling routine. Thus the call to set_zero_finder without any arguments is legal although rarely done. Usually, at least the bounds of the search are particular to the problem at hand.

```
CALL set\_zero\_finder(low\_limit,hi\_limit,abs\_tol,rel\_tol, &
    abs_step,rel_step,step_multiplier,local)
```

17.2 Arguments

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: low_limit, hi_limit**
  The least and greatest value of x to be searched for a zero of f(x). DEFAULT VALUEX: -1.0E35 and 1.0E35.

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: abs_tol, rel_tol**
  DEFAULT VALUES for both is 1.0E-6.

The search for a value, x, producing a 0 of f(x) is terminated when x is bounded in a region of length less than or equal to $tol = MAX(abs_tol, rel_tol * x)$.

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: abs_step, rel_step**
  The largest value of x to be searched for a zero of f(x). DEFAULT VALUES: 1.0E-4 and 1 respectively.

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: step_multiplier**
  DEFAULT VALUE: 2/

In the 'step' routines, x is set to the initial guess and the initial step size is set to

$$MAX(rel\_step * x, abs\_step)$$

.

If the previous step fails to bound a zero, then the step size is multiplied by **step_multiplier** and another step is taken.

- TYPE zf_locals:: local. A structure used to hold all local variables of the zero-finding routines – thus the routines are re-entrant – can be called from different places in the code without one call interferring with another. The type, **zf_locals** is defined in zero_finder_mod. A calling program should USE this module and declare some variable to be of this type. That variable can then be used as a value for **local**. DEFAULT: there is a default structure that is used if the caller does not specify his own. In this case, the code is not reentrant.

17.2.1 The Step Routines

CALL step_zf(f,y,answer,status,local)

CALL rc_step_zf(status,x,fx,local)

17.2.2 Arguments

- **f** The function of which the zero is sought is $f(x)-y=0$. Here is the definition of the argument f.

  INTERFACE DOUBLE PRECISION FUNCTION f(x) DOUBLE PRECISION, INTENT(IN):: x END INTERFACE

- **DOUBLE PRECISION, INTENT(IN):: y** The function of which the zero is sought is $f(x)-y=0$.

- **DOUBLE PRECISION, INTENT(INOUT):: answer**. On input. the initial guess as to x such that $f(x)-y=0$. On output if status==0, the answer obtained by the routine.

- **INTEGER, INTENT(INOUT):: status**. On input, status should be set to 0 to indicate the beginning of a new problem. On output, status==0 if an answer has been successfully found.

  For rc_step_zf, status can be returned with a value of 1. That indicates that the calling program whould evaluate f at x and place the result in the argument **fx** then call rc_step_zf again (without changing the value of status).

  A return value that is not 0 or 1 indicates that an error occurred in the zero-finding process.

- **TYPE zf_locals:: local**. This should be the same variable passed to set_zero_finder.

18 Finding a Maximum of a Function of a Single Variable

Two different packaging of one method are provided. The routine **fun_max** uses direct communication with the function to be maximized passed to the routine. The other routine **rc_fun_max** uses reverse communication with the calling program. When the value of the argument **status** is returned as one, the routine wants the function evaluated at the value of the argument **x** and the result returned in the argument **fx** on a subsequent call to **rc_fun_max.**

18.0.3 Setting Values for the Function Maximization Routines.

A call to set_fun_max is *required* before invoking any of the specific routines. This routine sets convergence criteria and optionally local variables for the function maximization routines. Note that all arguments are optional and take default values if not set by the calling routine. Thus the call to set_fun_max without any arguments is legal although rarely done. Usually, at least the bounds of the search are particular to the problem at hand.

```
    CALL set\_fun\_max(low\_limit,hi\_limit,abs\_tol,rel\_tol,local)
```

18.1 Arguments

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: low_limit, hi_limit**
  The least and greatest value of x to be searched for a maximum of f(x). DEFAULT VALUEX: -1.0E35 and 1.0E35.

- **DOUBLE PRECISION, INTENT(IN), OPTIONAL:: abs_tol, rel_tol**
  DEFAULT VALUES for both is 1.0E-6.

The search for a value, x, producing a 0 of f(x) is terminated when x is bounded in a region of length less than or equal to $tol = MAX(abs_tol, rel\_tol * x)$.

- TYPE fm_locals:: local. A structure used to hold all local variables of the function maximization routines – thus the routines are re-entrant – can be called from different places in the code without one call interferring with another (a property called re-entrant code). The type, **fm_locals** is defined in max_fun_mod. A calling program should USE this module and declare some variable to be of this type. That variable can then be used as a value for **local**. DEFAULT: there is a default structure that is used if the caller does not specify his own. In this case, the code is not reentrant.

18.1.1 The maximization routines

CALL fun_max(f,answer,local)

CALL rc_fun_max(status,x,fx,local)

18.1.2 Arguments

- **f**. The function of which the maximumizer is sought. Here is the definition of the argument **f**

```
INTERFACE
     DOUBLE PRECISION FUNCTION f(x)
     DOUBLE PRECISION, INTENT(IN):: x
END INTERFACE
```

- **DOUBLE PRECISION, INTENT(OUT):: answer**. The value of x that yields approximately a local maximum of $f(x)$ in the interval $[low\_limit, high\_limit]$.
- **TYPE (fm_locals), OPTIONAL :: local**. The same **local** argument as in the corresponding call to **set_fun_max**.
- **DOUBLE PRECISION, INTENT(OUT):: x**. If **status** is 1, the calling program should evaluate $f(x)$ and return this value in argument **fx** upon a subsequent call to rc_fun_max.
- **DOUBLE PRECISION, INTENT(IN):: fx**. The value of $f(x)$ returned by the calling program.

- **INTEGER, INTENT(INOUT):: status**.On input, status should be set to 0 to indicate the beginning of a new problem. On output, status is set to zero if an answer has been successfully found.

  For rc_fun_max, status can be returned with a value of 1. That indicates that the calling program whould evaluate f at x and place the result in the argument **fx** then call rc_fun_max again (without changing the value of status).

  A return value that is not 0 or 1 indicates that an error occurred in the zero-finding process.