```
*********************************************************
*                                                       *
*    Guide to the SLATEC Common Mathematical Library    *
*                                                       *
*********************************************************
```

Kirby W. Fong
National Magnetic Fusion Energy Computer Center
Lawrence Livermore National Laboratory

Thomas H. Jefferson
Operating Systems Division
Sandia National Laboratories Livermore

Tokihiko Suyehiro
Computing and Mathematics Research Division
Lawrence Livermore National Laboratory

Lee Walton
Network Analysis Division
Sandia National Laboratories Albuquerque

July 1993

****

Table of Contents

****

SECTION 1. ABSTRACT

This document is a guide to the SLATEC Common Mathematical Library (CML) [1]. The SLATEC CML is written in FORTRAN 77 (ANSI standard FORTRAN as defined by ANSI X3.9-1978, reference [6]) and contains general purpose mathematical and statistical routines. Included in this document are a Library description, code submission procedures, and a detailed description of the source file format. This report serves as a guide for programmers who are preparing codes for inclusion in the library. It also provides the information needed to process the source file automatically for purposes such as extracting documentation or inserting usage monitoring calls. This guide will be updated periodically, so be sure to contact a SLATEC CML subcommittee member to ensure you have the latest version.

****

SECTION 2. BACKGROUND

SLATEC is the acronym for the Sandia, Los Alamos, Air Force Weapons Laboratory Technical Exchange Committee. This organization was formed in 1974 by the computer centers of Sandia National Laboratories Albuquerque, Los Alamos National Laboratory, and Air Force Weapons Laboratory to foster the exchange of technical information. The parent committee established several subcommittees to deal with various computing specialties. The SLATEC Common Mathematical Library (CML) Subcommittee decided in 1977 to construct a mathematical FORTRAN subprogram library that could be used on a variety of computers at the three sites. A primary impetus for the library development was to provide portable, non-proprietary, mathematical software for member sites' supercomputers.

In l980 the computer centers of Sandia National Laboratories Livermore and the Lawrence Livermore National Laboratory were admitted as members of the parent committee and subcommittees. Lawrence Livermore National Laboratory, unlike the others, has two separate computer centers: the National Magnetic Fusion Energy Computer Center (NMFECC) and the Livermore Computer Center (LCC). In 1981 the National Bureau of Standards (now the National Institute of Standards and Technology) and the Oak Ridge National Laboratory were invited to participate in the math library subcommittee because of their great interest in the project.

Version 1.0 of the CML was released in April 1982 with 114,328 records and 491 user-callable routines. In May 1984 Version 2.0, with 151,864 records and 646 user-callable routines was released. This was followed in April 1986 by Version 3.0 with 196,013 records and 704 user-callable routines. Version 3.1 followed in August 1987 with 197,931 records and 707 user-callable routines and Version 3.2 in August 1989 with 203,587 records and 709 user-callable routines. The committee released Version 4.0 in December 1992 with 298,954 records and 901 user-callable routines. Finally, on July 1, 1993, Version 4.1 was released with 290,907 records and 902 user-callable routines.

The sole documentation provided by SLATEC for the routines of the SLATEC

Library is via comment lines in the source code. Although the library comes with portable documentation programs to help users access the documentation in the source code, various installations may wish to use their own documentation programs. To facilitate automatic extraction of documentation or further processing by other computer programs, the source file for each routine must be arranged in a precise format. This document describes that format for the benefit of potential library contributors and for those interested in extracting library documentation from the source code.

****

SECTION 3. MEMBERS OF THE SLATEC COMMON MATHEMATICAL LIBRARY SUBCOMMITTEE

Current member sites and voting members of the subcommittee are the following.

Air Force Phillips Laboratory, Kirtland (PLK)	Reginald Clemens
Lawrence Livermore National Laboratory (LCC)	Fred N. Fritsch
Lawrence Livermore National Laboratory (NERSC)	Steve Buonincontri
Los Alamos National Laboratory (LANL)	W. Robert Boland (Chairman)
National Institute of Standards and Technology (NIST)	Daniel W. Lozier
Oak Ridge National Laboratory (ORNL)	Thomas H. Rowan
Sandia National Laboratories/California (SNL/CA)	Thomas H. Jefferson
Sandia National Laboratories/New Mexico (SNL/NM)	Sue Goudy

****

SECTION 4. OBTAINING THE LIBRARY

The Library is in the public domain and distributed by the Energy Science and Technology Software Center.

Energy Science and Technology Software Center
P.O. Box 1020
Oak Ridge, TN 37831

Telephone 615-576-2606
E-mail estsc%a1.adonis.mrouter@zeus.osti.gov

****

SECTION 5. CODE SUBMISSION PROCEDURES

The SLATEC Library is continuously searching for portable high-quality routines written in FORTRAN 77 that would be of interest to the member sites. The subcommittee meets several times annually with the member sites rotating as meeting hosts. At these meetings new routines are introduced, discussed, and eventually voted on for inclusion in the library. Some of the factors that are considered in deciding whether to accept a routine into the Library are the following:

1. Usefulness. Does the routine fill a void in the Library? Will the routine have widespread appeal? Will it add a new capability?

2. Robustness. Does the routine give accurate results over a wide range of problems? Does it diagnose errors? Is the routine well tested?

3. Maintainability. Is the author willing to respond to bugs in the routine? Does the source code follow good programming practices?

4. Adherence to SLATEC standards and coding guidelines. These standards are described further in this guide and include such things as the order of subprogram arguments, the presence of a correctly formatted prologue at the start of each routine, and the naming of routines.

5. Good documentation. Is clear, concise computer readable documentation built into the source code?

6. Freely distributable. Is the program in the public domain?

A typical submission procedure begins with contact between an author and a Library committee member. Preliminary discussions with the member are encouraged for initial screening of any code and to gain insight into the workings of SLATEC. This member champions the routine to be considered. The code is introduced at a meeting where the author or committee member describes the code and explains why it would be suitable for SLATEC. Copies of the code are distributed to all committee members. Hopefully, the code already adheres to SLATEC standards. However, most codes do not. At this first formal discussion, the committee members are able to provide some useful suggestions for improving the code and revising it for SLATEC.

Between meetings, changes are made to the code and the modified code is distributed in machine readable format for testing. The code is then considered at a subsequent meeting, to be voted on and accepted. However, because committee members and authors do not always see eye to eye, and because time constraints affect all, the code is usually discussed at several meetings.

If codes adhered to the programming practices and formatting described in this guide, the time for acceptance could be greatly reduced.

****

SECTION 6. CODING GUIDELINES--GENERAL REQUIREMENTS FOR SLATEC

A software collection of the size of the SLATEC Library that is designed to run on a variety of computers demands uniformity in handling machine dependencies, in handling error conditions, and in installation procedures. Thus, while the decision to add a new subroutine to the library depends mostly on its quality and whether it fills a gap in the library, these are not the only considerations. Programming style must also be considered, so that the library as a whole behaves in a consistent manner. We now list the stylistic and documentational recommendations and requirements for routines to be incorporated into the library.

1. The SLATEC Library is intended to have no restriction on its distribution; therefore, new routines must be in the public domain. This is generally not a problem since most authors are proud of their work and would like their routines to be used widely.

2. Routines must be written in FORTRAN 77 (ANSI standard FORTRAN as defined by ANSI X3.9-1978, reference [6]). Care must be taken so that machine dependent features are not used.

3. To enhance maintainability codes are to be modular in structure. Codes must be composed of reasonably small subprograms which in turn are made up of easily understandable blocks.

4. Equivalent routines of different precision are to look the same where possible. That is, the logical structure, statement numbers, variable names, etc. are to be as close to identical as possible. This implies that generic intrinsics must be used instead of specific intrinsics. Extraneous use of INT, REAL and DBLE are strongly discouraged; use mixed-mode expressions in accordance with the Fortran 77 standard.

5. New routines must build on existing routines in the Library, unless there are compelling reasons to do otherwise. For example, the SLATEC Library contains the LINPACK and EISPACK routines, so new routines should use the existing linear system and eigensystem routines rather than introduce new ones.

6. System or machine dependent values must be obtained by calling routines D1MACH, I1MACH, and R1MACH. The SLATEC Library has adopted these routines from the Bell Laboratories' PORT Library [2] [3]. See Appendix B for a description of these machine dependent routines.

7. The SLATEC Library has a set of routines for handling error messages. Each user-callable routine, if it can detect errors, must have as one of its arguments an error flag, whose value upon exiting the routine indicates the success or failure of the routine. It is acceptable for a routine to set the error flag and RETURN; however, if the routine wishes to write an error message, it must call XERMSG (see Appendix C) rather than use WRITE or PRINT statements. In general, all errors (even serious ones) should be designated as "recoverable" rather than "fatal," and the routine should RETURN to the user. This permits the user to try an alternate strategy if a routine decides a particular calculation is inappropriate. A description of the entire original error handling package appears in reference [4].

8. Each user-callable routine (and subsidiary routine if appropriate) must have a small demonstration routine that can be used as a quick check. This demonstration routine can be more exhaustive, but in general, it should be structured to provide a "pass" or "fail" answer on whether the library routine appears to be functioning properly. A more detailed description of the required format of the quick checks appears later in this document.

9. Common blocks and SAVEd variables must be avoided. Use subprogram arguments for interprogram communication. The use of these constructs often obstructs multiprocessing.

   Variables that are statically allocated in memory and are used as working storage cannot be used simultaneously by several processors. SAVEd variables and common block variables are most likely to fall into this category. Such variables are acceptable if they are DATA loaded or set at run time to values that are to be read (but not written) since it does not matter in what order multiple processors read the values. However, such variables should not be used as working storage since no processor can use the work space while some other processor is using it. Library routines should ask the user to provide any needed work space by passing it in as an argument. The user is then responsible for giving each processor a different work space even though each processor may be executing the same library routine.

10. Complete self-contained documentation must be supplied as comments in user-callable routines. This documentation must be self-contained because SLATEC provides no other documentation for using the routines. This documentation is called the "prologue" for the routine. The rigid prologue format for user-callable routines is described below. The prologue must tell the user how to call the routine but need not go into algorithmic details since such explanations often require diagrams or non-ASCII symbols. Subsidiary routines are those called by other library routines but which are not intended to be called directly by the user. Subsidiary routines also have prologues, but these prologues are considerably less elaborate than those of user-callable routines.

11. No output should be printed. Instead, information should be returned to the user via the subprogram arguments or function values. If there is some overriding reason that printed output is necessary, the user must be able to suppress all output by means of a subprogram input variable.

****

SECTION 7. SOURCE CODE FORMAT

In this section and the two sections on prologues, we use the caret (^) character to indicate a position in which a single blank character must appear. Upper case letters are used for information that appears literally. Lower case is used for material specific to the routine.

1. The first line of a subprogram must start with one of:

```
SUBROUTINE^name^(arg1,^arg2,^...argn)
FUNCTION^name^(arg1,^arg2,^...argn)
COMPLEX^FUNCTION^name^(arg1,^arg2,^...argn)
DOUBLE^PRECISION^FUNCTION^name^(arg1,^arg2,^...argn)
INTEGER^FUNCTION^name^(arg1,^arg2,^...argn)
REAL^FUNCTION^name^(arg1,^arg2,^...argn)
LOGICAL^FUNCTION^name^(arg1,^arg2,^...argn)
CHARACTER[*len]^FUNCTION^name^(arg1,^arg2,^...argn)
```

Each of the above lines starts in column 7. If there is an argument list, then there is exactly one blank after the subprogram name and after each comma (except if the comma appears in column 72). There is no embedded blank in any formal parameter, after the leading left parenthesis, before the trailing right parenthesis, or before any comma. Formal parameters are never split across lines. Any line to be continued must end with a comma.

For continuation lines, any legal continuation character may be used in column 6, columns 7-9 must be blank and arguments or formal parameters start in column 10 of a continuation line and continue up to the right parenthesis (or comma if another continuation line is needed). The brackets in the CHARACTER declaration do not appear literally but indicate the optional length specification described in the FORTRAN 77 standard.

2. The author must supply a prologue for each subprogram. The prologue must be in the format that will subsequently be described. The prologue begins with the first line after the subprogram declaration (including continuation lines for long argument lists).

3. Except for the "C***" lines (to be described) in the prologue and the "C***" line marking the first executable statement, no other line may begin with "C***".

4. The first line of the prologue is the comment line

   C***BEGIN^PROLOGUE^^name

   where "name", starting in column 21, is the name of the subprogram.

5. The last line of a subprogram is the word "END" starting in column 7.

6. All alphabetic characters, except for those on comment lines or in character constants, must be upper case, as specified by the FORTRAN 77 standard (see [6]).

7. In the prologue, the comment character in column 1 must be the upper case "C".

8. All subprogram, common block, and any formal parameter names mentioned in the prologue must be in upper case.

9. Neither FORTRAN statements nor comment lines can extend beyond column 72. Columns 73 through 80 are reserved for identification or sequence numbers.

10. Before the first executable statement of every subprogram, user-callable or not, is the line

    C***FIRST^EXECUTABLE^STATEMENT^^name

    where "name" (starting in column 33) is the name of the subprogram. Only comment lines may appear between the C***FIRST EXECUTABLE STATEMENT line and the first executable statement.

11. The subprogram name consists of a maximum of six characters. Authors should choose unusual and distinctive subprogram names to minimize possible name conflicts. Double precision routines should begin with "D". Subprograms of type complex should begin with "C". The letter "Z" is reserved for future use by possible double precision complex subprograms. No other subprograms should begin with either "D", "C", or "Z".

12. The recommended order for the formal parameters is:

    1. Names of external subprograms.

    2. Input variables.

    3. Variables that are both input and output (except error flags).

4. Output variables.

5. Work arrays.

6. Error flags.

However, array dimensioning parameters should immediately follow the associated array name.

****

SECTION 8. PROLOGUE FORMAT FOR SUBPROGRAMS

Each subprogram has a section called a prologue that gives standardized information about the routine. The prologue consists of comment lines only. A subsidiary subprogram is one that is usually called by another SLATEC Library subprogram only and is not meant to be called by a user's routine. The prologue for a user-callable subprogram is more extensive than the prologue for a subsidiary subprogram. The prologue for a user-callable subprogram has up to 14 sections, of which 12 are required and one is required if and only if a common block is present. Several of these sections are optional in subsidiary programs and in the quick check routines. The sections are always in the order described in the table below.

	Section	User-callable	Subsidiary	Quick Checks
1.	BEGIN PROLOGUE	Required	Required	Required
2.	SUBSIDIARY	Not present	Required	Optional
3.	PURPOSE	Required	Required	Required
4.	LIBRARY SLATEC	Required	Required	Required
5.	CATEGORY	Required	Optional	Optional
6.	TYPE	Required	Required	Required
7.	KEYWORDS	Required	Optional	Optional
8.	AUTHOR	Required	Required	Required
9.	DESCRIPTION	Required	Optional	Optional
10.	SEE ALSO	Optional	Optional	Optional
11.	REFERENCES	Required	Optional	Optional
12.	ROUTINES CALLED	Required	Required	Required
13.	COMMON BLOCKS	Required***	Required***	Required***
14.	REVISION HISTORY	Required	Required	Required
15.	END PROLOGUE	Required	Required	Required

***Note: The COMMON BLOCKS section appears in a subprogram prologue if and only if the subprogram contains a common block.

In the prologue section descriptions that follow, the caret (^) character is used for emphasis to indicate a required blank character.

1. BEGIN PROLOGUE
This section is a single line that immediately follows the subprogram declaration and its continuation lines. It is

```
C***BEGIN^PROLOGUE^^name
```

where "name" (beginning in column 21) is the name of the subprogram.

2. SUBSIDIARY
   This section is the single line

```
C***SUBSIDIARY
```

   and indicates the routine in which this appears is not intended to be user-callable.

3. PURPOSE
   This section gives one to six lines of information on the purpose of the subprogram. The letters may be in upper or lower case. There are no blank lines in the purpose section; i.e., there are no lines consisting solely of a "C" in column 1. The format for the first line and any continuation lines is

```
C***PURPOSE^^information
C^^^^^^^^^^^^^more information
```

   Information begins in column 14 of the first line and no earlier than column 14 of continuation lines.

4. LIBRARY   SLATEC
   The section is a single line used to show that the routine is a part of the SLATEC library and, optionally, to indicate other libraries, collections, or packages (sublibraries) of which the routine is a part or from which the routine has been derived.   The format is

```
C***LIBRARY^^^SLATEC
        or
C***LIBRARY^^^SLATEC^(sublib1,^sublib2,^...sublibn)
```

   The leading left parenthesis is immediately followed by the first member of the list. Each member, except for the last, is immediately followed by a comma and a single blank. The last member is immediately followed by the trailing right parenthesis.

5. CATEGORY
   This section is a list of classification system categories to which this subprogram might reasonably be assigned. There must be at least one list item. The first category listed is termed the primary category, and others, if given, should be listed in monotonically decreasing order of importance. Categories must be chosen from the classification scheme listed in Appendix A. The required format for the initial line and any continuation lines is

```
C***CATEGORY^^cat1,^cat2,^cat3,^...catn,
C^^^^^^^^^^^^^^continued list
```

   All alphabetic characters are in upper case.

   Items in the list are separated by the two characters, comma and space. If the list will not fit on one line, the line may be ended at a comma (with zero or more trailing spaces), and be continued on the next line. The list and any continuations of the list begin with a nonblank character in column 15.

6. TYPE
   This section gives the datatype of the routine and indicates which routines, including itself, are equivalent (except possibly for type) to the routine. The format for this section is

```
C***TYPE^^^^^^routine_type^(equivalence list
C^^^^^^^^^^^^^continued equivalence list
C^^^^^^^^^^^^^continued equivalence list)
```

Routine_type, starting in column 15, is the data type of the routine, and is either SINGLE PRECISION, DOUBLE PRECISION, COMPLEX, INTEGER, CHARACTER, LOGICAL, or ALL. ALL is a pseudo-type given to routines that could not reasonably be converted to some other type. Their purpose is typeless. An example would be the SLATEC routine that prints error messages.

Equivalence list is a list of the routines (including this one) that are equivalent to this one, but perhaps of a different type. Each item in the list consists of a routine name followed by the "-" character and then followed by the first letter of the type (except use "H" for type CHARACTER) of the equivalent routine. The order of the items is S, D, C, I, H, L and A.

The initial item in the list is immediately preceded by a blank and a left parenthesis and the final item is immediately followed by a right parenthesis. Items in the list are separated by the two characters, comma and space. If the list will not fit on one line, the line may be ended at a comma (with zero or more trailing spaces), and be continued on the next line. The list and any continuations of the list begin with a nonblank character in column 15.

All alphabetic characters in this section are in upper case.

Example

```
C***TYPE      SINGLE PRECISION (ACOSH-S, DACOSH-D, CACOSH-C)
```

7. KEYWORDS
   This section gives keywords or keyphrases that can be used by information retrieval systems to identify subprograms that pertain to the topic suggested by the keywords. There must be at least one keyword. Keywords can have embedded blanks but may not have leading or trailing blanks. A keyword cannot be continued on the next line; it must be short enough to fit on one line. No keyword can have an embedded comma. Characters are limited to the FORTRAN 77 character set (in particular, no lower case letters). There is no comma after the last keyword in the list. It is suggested that keywords be in either alphabetical order or decreasing order of importance. The format for the initial line and any continuation lines is

```
C***KEYWORDS^^list
C^^^^^^^^^^^^^continued list
```

   Items in the list are separated by the two characters, comma and space. If the list will not fit on one line, the line may be ended at a comma (with zero or more trailing spaces), and be continued on the next line. The list and any continuations of the list begin with a nonblank character in column 15.

8. AUTHOR
   This required section gives the author’s name. There must be at least one author, and there may be coauthors. At least the last name of the author must be given. The first name (or initials) is optional. The company, organization, or affiliation of the author is also optional. The brackets below indicate optional information. Note that if an organization is to be

listed, the remainder of the author's name must also be given. If the remainder of the author's name is given, the last name is immediately followed by a comma. If the organization is given, the first name (or initials) is immediately followed by a comma. The remainder of the name and the organization name may have embedded blanks. The remainder of the name may not have embedded commas. This makes it possible for an information retrieval system to count commas to identify the remainder of the name and the name of an organization. Additional information about the author (e.g., address or telephone number) may be given on subsequent lines. The templates used are

```
C***AUTHOR^^last-name[,^first-name[,^(org)]]
C^^^^^^^^^^^^^^more information
C^^^^^^^^^^^^^^more information
    .
    .
    .
C^^^^^^^^^^^^last-name[,^first-name[,^(org)]]
C^^^^^^^^^^^^^^more information
    .
    .
    .
```

Each author's name starts in column 13. Continued information starts in column 15.

9. DESCRIPTION
This section is a description giving the program abstract, method used, argument descriptions, dimension information, consultants, etc. The description of the arguments is in exactly the same order in which the arguments appear in the calling sequence. The description section may use standard, 7-bit ASCII graphic characters, i.e., the 94 printing characters plus the blank. Names of subprograms, common blocks, externals, and formal parameters are all in upper case. Names of variables are also in upper case. The first line of this section is "C***DESCRIPTION" starting in column 1. All subsequent lines in this section start with a "C" in column 1 and no character other than a blank in column 2. Lines with only a "C" in column 1 may be used to improve the appearance of the description.

A suggested format for the DESCRIPTION section is given in Appendix E.

10. SEE ALSO
This section is used for listing other SLATEC routines whose prologues contain documentation on the routine in which this section appears.
The form is

```
C***SEE ALSO^^name,^name,^name
```

where each "name" is the name of a user-callable SLATEC CML subprogram whose prologue provides a description of this routine. The names are given as a list (starting in column 15), with successive names separated by a comma and a single blank.

11. REFERENCES
This section is for references. Any of the 94 ASCII printing characters plus the blank may be used. There may be more than one reference. If there are no references, the section will consist of the single line

```
C***REFERENCES^^(NONE)
```

If there are references, they will be in the following format:

```
C***REFERENCES^^reference 1
C^^^^^^^^^^^^^^^^^continuation of reference 1
    .
    .
    .
C^^^^^^^^^^^^^^^reference 2
C^^^^^^^^^^^^^^^^^continuation of reference 2
    .
    .
    .
```

Information starts in column 17 of the first line of a reference and no earlier than column 19 of continuation lines.

References should be listed in either alphabetical order by last name or order of citation. They should be in upper and lower case, have initials or first names ahead of last names, and (for multiple authors) have "and" ahead of the last author's name instead of just a comma. The first word of the title of journal articles should be capitalized as should all important words in titles of books, pamphlets, research reports, and proceedings. Titles should be given without quotation marks. The names of journals should be spelled out completely, or nearly so, because software users may not be familiar with them.

A complete example of a journal reference is:

```
C               F. N. Fritsch and R. E. Carlson, Monotone piecewise
C                 cubic interpolation, SIAM Journal on Numerical Ana-
C                 lysis, 17 (1980), pp. 238-246.
```

A complete example of a book reference is:

```
C               Carl de Boor, A Practical Guide to Splines, Applied
C                 Mathematics Series 27, Springer-Verlag, New York,
C                 1978.
```

12. ROUTINES CALLED
This section gives the names of routines in the SLATEC Common Mathematical Library that are either directly referenced or declared in an EXTERNAL statement and passed as an argument to a subprogram. Note that the FORTRAN intrinsics and other formal parameters that represent externals are not listed. A list is always given for routines called; however, if no routine is called, the list will be the single item "(NONE)" where the parentheses are included. If there are genuine items in the list, the items are in alphabetical order. The collating sequence has "0" through "9" first, then "A" through "Z". The format is

```
C***ROUTINES^CALLED^^name,^name,^name,^name,
C^^^^^^^^^^^^^^^^^^^^name,^name,^name
```

Items in the list are separated by the two characters, comma and space. If the list will not fit on one line, the line may be ended at a comma (with zero or more trailing spaces), and be continued on the next line. The list and any continuations of the list begin with a nonblank character in column 22.

13. COMMON BLOCKS
This section, that may or may not be required, tells what common blocks are used by this subprogram. If this subprogram uses no common blocks, this section does not appear. If this subprogram does use common blocks, this

section must appear. The list of common blocks is in exactly the same format as the list of routines called and uses the same collating sequence. In addition, the name of blank common is "(BLANK)" where the parentheses are included. Blank common should be last in the list if it appears. The format for this section is

```
C***COMMON^BLOCKS^^^^name,^name,^name,^name,
C^^^^^^^^^^^^^^^^^^^^^name,^name,^name^
```

The list starts in column 22.

14. REVISION HISTORY
    This section provides a summary of the revisions made to this code. Revision dates and brief reasons for revisions are given. The format is

```
C***REVISION^HISTORY^^(YYMMDD)
C^^^yymmdd^^DATE^WRITTEN
C^^^yymmdd^^revision description
C^^^^^^^^^^^^more revision description
C^^^^^^^^^^^^...
C^^^yymmdd^^revision description
C^^^^^^^^^^^^more revision description
C^^^^^^^^^^^^...
C^^^^^^^^^^^^...
```

    where, for each revision, "yy" (starting in column 5) is the last two digits of the year, "mm" is the month (01, 02, ..., 12), and "dd" is the day of the month (01, 02, ..., 31). Because this ANSI standard form for the date may not be familiar to some people, the character string "(YYMMDD)" (starting in column 23) is included in the first line of the section to assist in interpreting the sequence of digits. Each line of the revision descriptions starts in column 13. The second line of this section contains the date the routine was written, with the characters "DATE WRITTEN" beginning in column 13. These items must be in chronological order.

15. END PROLOGUE
    The last section is the single line

```
C***END^PROLOGUE^^name
```

    where "name" is the name of the subprogram.

****

SECTION 9. EXAMPLES OF PROLOGUES

This section contains examples of prologues for both user-callable and subsidiary routines. The routines are not from the SLATEC CML and should be used only as guidelines for preparing routines for SLATEC. Note that the C***DESCRIPTION sections follow the suggested LDOC format that is described in Appendix E. Following the suggested LDOC format with its "C *"subsections helps to ensure that all necessary descriptive information is provided.

```
      SUBROUTINE ADDXY (X, Y, Z, IERR)
C***BEGIN PROLOGUE  ADDXY
C***PURPOSE  This routine adds two single precision numbers together
```

```
C            after forcing both operands to be stored in memory.
C***LIBRARY   SLATEC
C***CATEGORY  A3A
C***TYPE      SINGLE PRECISION (ADDXY-S, DADDXY-D)
C***KEYWORDS  ADD, ADDITION, ARITHMETIC, REAL, SUM,
C             SUMMATION
C***AUTHOR  Fong, K. W., (NMFECC)
C             Mail Code L-560
C             Lawrence Livermore National Laboratory
C             Post Office Box 5509
C             Livermore, CA  94550
C           Jefferson, T. H., (SNLL)
C             Org. 8235
C             Sandia National Laboratories Livermore
C             Livermore, CA  94550
C           Suyehiro, T., (LLNL)
C             Mail Code L-316
C             Lawrence Livermore National Laboratory
C             Post Office Box 808
C             Livermore, CA  94550
C***DESCRIPTION
C
C *Usage:
C
C     INTEGER IERR
C     REAL X, Y, Z
C
C     CALL ADDXY (X, Y, Z, IERR)
C
C *Arguments:
C
C  X :IN   This is one of the operands to be added.  It will not
C          be modified by ADDXY.
C
C  Y :IN   This is the other operand to be added.  It will not be
C          modified by ADDXY.
C
C  Z :OUT  This is the sum of X and Y.  In case of an error,
C          this argument will not be modified.
C
C  IERR:OUT  This argument will be set to 0 if ADDXY added the two
C          operands.  It will be set to 1 if it appears the addition
C          would generate a result that might overflow.
C
C *Description:
C
C  ADDXY first divides X and Y by the largest single precision number
C  and then adds the quotients.  If the absolute value of the sum is
C  greater than 1.0, ADDXY returns with IERR set to 1.  Otherwise
C  ADDXY stores X and Y into an internal array and calls ADDZZ to add
C  them.  This increases the probability (but does not guarantee) that
C  operands and result are stored into memory to avoid retention of
C  extra bits in overlength registers or cache.
C
C***REFERENCES  W. M. Gentleman and S. B. Marovich, More on algorithms
C                 that reveal properties of floating point arithmetic
C                 units, Communications of the ACM, 17 (1974), pp.
C                 276-277.
C***ROUTINES CALLED  ADDZZ, R1MACH, XERMSG
C***REVISION HISTORY  (YYMMDD)
C   831109  DATE WRITTEN
```

```
C   880325  Modified to meet new SLATEC prologue standards.  Only
C           comment lines were modified.
C   881103  Brought DESCRIPTION section up to Appendix E standards.
C   921215  REFERENCE section modified to reflect recommended style.
C***END PROLOGUE  ADDXY
      DIMENSION R(3)
C***FIRST EXECUTABLE STATEMENT  ADDXY
      BIG = R1MACH( 2 )
C
C  This is an example program, not meant to be taken seriously.  The
C  following illustrates the use of XERMSG to send an error message.
C
      IF ( (ABS((X/BIG)+(Y/BIG))-1.0) .GT. 0.0 ) THEN
         IERR = 1
         CALL XERMSG ( 'SLATEC', 'ADDXY', 'Addition of the operands '//
     *      'is likely to cause overflow', IERR, 1 )
      ELSE
         IERR = 0
         R(1) = X
         R(2) = Y
         CALL ADDZZ( R )
         Z    = R(3)
      ENDIF
      RETURN
      END
      SUBROUTINE ADDZZ (R)
C***BEGIN PROLOGUE  ADDZZ
C***SUBSIDIARY
C***PURPOSE  This routine adds two single precision numbers.
C***LIBRARY   SLATEC
C***AUTHOR  Fong, K. W., (NMFECC)
C             Mail Code L-560
C             Lawrence Livermore National Laboratory
C             Post Office Box 5509
C             Livermore, CA  94550
C           Jefferson, T. H., (SNLL)
C             Org. 8235
C             Sandia National Laboratories Livermore
C             Livermore, CA  94550
C           Suyehiro, T., (LLNL)
C             Mail Code L-316
C             Lawrence Livermore National Laboratory
C             Post Office Box 808
C             Livermore, CA  94550
C***SEE ALSO  ADDXY
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   831109  DATE WRITTEN
C   880325  Modified to meet new SLATEC prologue standards.  Only
C           comment lines were modified.
C***END PROLOGUE  ADDZZ
      DIMENSION R(3)
C***FIRST EXECUTABLE STATEMENT  ADDZZ
      R(3) = R(1) + R(2)
      RETURN
      END
```

****

SECTION 10. SLATEC QUICK CHECK PHILOSOPHY

The SLATEC Library is distributed with a set of test programs that may be used as an aid to insure that the Library is installed correctly. This set of test programs is known as the SLATEC quick checks. The quick checks are not meant to provide an exhaustive test of the Library. Instead they are designed to protect against gross errors, such as an unsatisfied external. Because the SLATEC Library runs on a great variety of computers, the quick checks often detect arithmetic difficulties with either particular Library routines or with a particular computational environment.

A list of the quick check guidelines follows.

1. A quick check should test a few problems successfully solved by a particular library subprogram. It is not intended to be an extensive test of a subprogram.

2. A quick check should provide consistent and minimal output in most cases, including a "PASS" or "FAIL" indicator. However, more detailed output should be available on request to help track down problems in the case of failures.

3. Some reasonable error conditions should be tested by the quick check by purposefully referencing the routine incorrectly.

4. A quick check subprogram is expected to execute correctly on any machine with an ANSI Fortran 77 compiler and library. No test should have to be skipped to avoid an abort on a particular machine.

5. As distributed on the SLATEC tape, the quick check package consists of a number of quick check main programs and a moderate number of subprograms. Each quick check main program, more frequently called a quick check driver, calls one or more quick check subprograms. Usually, a given driver initiates the tests for a broadly related set of subprograms, e.g. for the single precision Basic Linear Algebra Subprograms (BLAS). Each quick check subprogram will test one or more closely related library routines of the same precision. For example, single precision routines and their double precision equivalents are not to be tested in the same quick check subprogram.

6. The format of the quick check package does not rigidly dictate how it must be executed on a particular machine. For example, memory size of the machine might preclude loading all quick check modules at once.

****

SECTION 11. SPECIFIC PROGRAMMING STANDARDS FOR SLATEC QUICK CHECKS

Just as the routines in the SLATEC Common Mathematical Library must meet certain standards, so must the quick checks. These standards are meant to ensure that the quick checks adhere to the SLATEC quick check philosophy and to enhance maintainability. The list of these quick check standards follow.

1. Each module must test only a few related library subprograms.

2. Each module must be in the form of a subroutine with three arguments. For example:

```
SUBROUTINE ADTST (LUN, KPRINT, IPASS)
```

The first is an input argument giving the unit number to which any output should be written. The second is an input argument specifying the amount of printing to be done by the quick check subroutine. The third is an output flag indicating passage or failure of the subroutine.

LUN Unit number to which any output should be written.

KPRINT = 0 No printing is done (pass/fail is presumably monitored at a higher level, i.e. in the driver). Error messages will not be printed since the quick check driver sets the error handling control flag to 0, using CALL XSETF(0) when KPRINT = 0 or 1.

= 1 No printing is done for tests which pass; a short message (e.g., one line) is printed for tests which fail. Error messages will not be printed since the quick check driver sets the error handling control flag to 0, using CALL XSETF(0) when KPRINT = 0 or 1.

= 2 A short message is printed for tests which pass; more detailed information is printed for tests which fail. Error messages describing the reason for failure should be printed.

= 3 (Possibly) quite detailed information is printed for all tests. Error messages describing the reason for failure should be printed.

IPASS = 0 Indicates failure of the quick check subroutine (i.e., at least one test failed).

= 1 Indicates that all tests passed in the quick check subroutine.

In the case of a subroutine whose purpose is to produce output (e.g., a printer-plotter), output of a more detailed nature might be produced for KPRINT >= 1.

The quick check must execute correctly and completely using each value of KPRINT. KPRINT is used only to control the printing and does not affect the tests made of the SLATEC routine.

3. The quick check subprograms must be written in ANSI Fortran 77 and must make use of I1MACH, R1MACH, and D1MACH for pass/fail tolerances.

4. Where possible, compute constants in a machine independent fashion. For example, PI = 4. * ATAN(1.0)

5. Using one library routine to test another is permitted, though this should be done with care.

6. Known solutions can be stored using DATA or PARAMETER statements. Some subprograms return a "solution" which is more than one number - for example, the eigenvalues of a matrix. In these cases, take special care that the quick check test passes for ALL orderings of the output which are mathematically correct.

7. Where subprograms are required by a routine being tested, they should accompany the quick check. However, care should be taken so that

no two such subprograms have the same name. Choosing esoteric or odd names is a good idea. It is extremely desirable that each such subprogram contain comments indicating which quick check needed it (a C***SEE ALSO line should be used).

8. Detailed output should be self-contained yet concise. No external reference material or additional computations should be required to determine what, for example, the correct solution to the problem really is.

9. For purposes of tracking down the cause of a failure, external reference material or the name of a (willing) qualified expert should be listed in the comment section of the quick check.

10. Quick checks must have SLATEC prologues and be adequately commented and cleanly written so that the average software librarian has some hope of tracking down problems. For example, if a test problem is known to be tricky or if difficulties are expected for short word length machines, an appropriate comment would be helpful.

11. After deliberately calling a library routine with incorrect arguments, invoke the function IERR=NUMXER(NERR) to verify that the correct error number was set. (NUMXER is a function in the SLATEC error handling package that returns the number of the most recent error via both the function value and the argument.) Then CALL XERCLR to clear it before this (or the next) quick check makes another error.

12. A quick check should be written in such a way that it will execute identically if called several times in the same program. In particular, there should be no modification of DATA loaded variables which cause the quick check to start with the wrong values on subsequent calls.

****

SECTION 12. QUICK CHECK DRIVERS (MAIN PROGRAMS)

Many people writing quick checks are not aware of the environment in which the individual quick check is called. The following aspects of the quick check drivers are illustrated by the example driver in Section 14.

1. Each quick check driver will call one or more quick check subprograms.

2. The input and output units for the tests are set in the driver.

```
LIN = I1MACH(1)        the input unit
LUN = I1MACH(2)        the output unit
```

The output unit is communicated to the quick check subprograms through the argument list. All output should be directed to the unit LUN that is in the argument list.

3. Each quick check has three arguments LUN, KPRINT, and IPASS. The meaning of these arguments within the quick checks is detailed thoroughly in the previous section.

   a. The quick check driver reads in KPRINT without a prompt, and passes KPRINT as an argument to each quick check it calls. KPRINT must not be changed by any driver or quick check. The driver uses KPRINT to help determine what output to write.

b. The variable IPASS must be set to 0 (for fail) or to 1 (for pass) by each quick check before returning to the driver. Within the driver, the variable NFAIL is set to 0. If IPASS = 0 upon return to the driver, then NFAIL is incremented. After calling all the quick checks, NFAIL will then have the number of quick checks which failed.

c. Quick check driver output should follow this chart:

```
NFAIL        OUTPUT
-----        ------

not 0        driver writes fail message
  0          driver writes pass message
```

4. There are calls to three SLATEC error handler routines in each quick check driver:

```
CALL XSETUN(LUN)       Selects unit LUN as the unit to which
                          error messages will be sent.
CALL XSETF(1)          Only fatal (not recoverable) error messages
  or XSETF(0)             will cause an abort.  XSETF sets the
                          KONTROL variable for the error handler
                          routines to the value of the XSETF
                          argument.  A value of either 0 or 1 will
                          make only fatal errors cause a program
                          abort.  A value of 1 will allow printing
                          of error messages, while a value of zero
                          will print only fatal error messages.
CALL XERMAX(1000)      Increase the number of times any
                          single message may be printed.
```

****

SECTION 13. QUICK CHECK SUBROUTINE EXAMPLE

The following program provides a very minimal check of the sample routine from Section 9.

```
      SUBROUTINE ADTST (LUN, KPRINT, IPASS)
C***BEGIN PROLOGUE  ADTST
C***SUBSIDIARY
C***PURPOSE  Quick check for SLATEC routine ADDXY
C***LIBRARY   SLATEC
C***CATEGORY  A3A
C***TYPE      SINGLE PRECISION (ADTST-S, DADTST-D)
C***KEYWORDS  QUICK CHECK, ADDXY,
C***AUTHOR  Suyehiro, Tok, (LLNL)
C           Walton, Lee, (SNL)
C***ROUTINES CALLED  ADDXY, R1MACH
C***REVISION HISTORY  (YYMMDD)
C   880511  DATE WRITTEN
C   880608  Revised to meet new prologue standards.
C***END PROLOGUE  ADTST
C
C***FIRST EXECUTABLE STATEMENT  ADTST
```

```
      IF ( KPRINT .GE. 2 ) WRITE (LUN,99999)
99999 FORMAT ('OUTPUT FROM ADTST')
      IPASS = 1
C
C EXAMPLE PROBLEM
      X = 1.
      Y = 2.
      CALL ADDXY(X, Y, Z, IERR)
      EPS = R1MACH(4)
      IF( (ABS(Z-3.) .GT. EPS)  .OR.  (IERR .EQ. 1) ) IPASS = 0
      IF ( KPRINT .GE. 2 ) THEN
      WRITE (LUN,99995)X, Y, Z
99995 FORMAT (/' EXAMPLE PROBLEM ',/' X = ',E20.13,' Y = ',E20.13,' Z = ',
     *   E20.13)
      ENDIF
      IF ( (IPASS .EQ. 1 ) .AND. (KPRINT .GT. 1) ) WRITE (LUN,99994)
      IF ( (IPASS .EQ. 0 ) .AND. (KPRINT .NE. 0) ) WRITE (LUN,99993)
99994 FORMAT(/' **************ADDXY  PASSED ALL TESTS**************')
99993 FORMAT(/' **************ADDXY  FAILED SOME TESTS**************')
      RETURN
      END
```

****

SECTION 14. QUICK CHECK MAIN PROGRAM EXAMPLE

The following is an example main program which should be used to drive a quick check. The names of the quick check subroutines it calls, ADTST and DADTST, should be replaced with the name or names of real quick checks. The dummy names of the SLATEC routines being tested, ADDXY and DADDXY, should be replaced with the names of the routines which are actually being tested.

```
      PROGRAM TEST00
C***BEGIN PROLOGUE  TEST00
C***SUBSIDIARY
C***PURPOSE  Driver for testing SLATEC subprograms
C            ADDXY    DADDXY
C***LIBRARY   SLATEC
C***CATEGORY  A3
C***TYPE      ALL (TEST00-A)
C***KEYWORDS  QUICK CHECK DRIVER, ADDXY, DADDXY
C***AUTHOR  Suyehiro, Tok, (LLNL)
C           Walton, Lee, (SNL)
C***DESCRIPTION
C
C *Usage:
C     One input data record is required
C         READ (LIN,990) KPRINT
C     990 FORMAT (I1)
C
C *Arguments:
C     KPRINT = 0  Quick checks - No printing.
C                 Driver       - Short pass or fail message printed.
C              1  Quick checks - No message printed for passed tests,
C                                short message printed for failed tests.
C                 Driver       - Short pass or fail message printed.
C              2  Quick checks - Print short message for passed tests,
```

```
C                                 fuller information for failed tests.
C                  Driver       - Pass or fail message printed.
C               3  Quick checks - Print complete quick check results.
C                  Driver       - Pass or fail message printed.
C
C *Description:
C     Driver for testing SLATEC subprograms
C        ADDXY    DADDXY
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  ADTST, DADTST, I1MACH, XERMAX, XSETF, XSETUN
C***REVISION HISTORY  (YYMMDD)
C   880511  DATE WRITTEN
C   880608  Revised to meet the new SLATEC prologue standards.
C   881103  Brought DESCRIPTION section up to Appendix E standards.
C***END PROLOGUE  TEST00
C
C***FIRST EXECUTABLE STATEMENT  TEST00
      LUN   = I1MACH(2)
      LIN   = I1MACH(1)
      NFAIL = 0
C
C   Read KPRINT parameter
C
      READ (LIN,990) KPRINT
  990 FORMAT (I1)
      CALL XSETUN(LUN)
      IF ( KPRINT .LE. 1 ) THEN
         CALL XSETF(0)
      ELSE
         CALL XSETF(1)
      ENDIF
      CALL XERMAX(1000)
C
C   Test ADDXY
C
      CALL ADTST(LUN, KPRINT, IPASS)
      IF ( IPASS .EQ. 0 ) NFAIL = NFAIL + 1
C
C   Test DADDXY
C
      CALL DADTST(LUN, KPRINT, IPASS)
      IF ( IPASS .EQ. 0 ) NFAIL = NFAIL + 1
C
      IF ( NFAIL .GT. 0 ) WRITE (LUN,980) NFAIL
  980 FORMAT (/' ************* WARNING -- ', I5,
     * ' TEST(S) FAILED IN PROGRAM TEST00 *************' )
      IF ( NFAIL .EQ. 0 ) WRITE (LUN,970)
  970 FORMAT
     * (/' --------------TEST00  PASSED ALL TESTS----------------')
      END
```

****

APPENDIX A. GAMS (AND SLATEC) CLASSIFICATION SCHEME

SLATEC has adopted the GAMS (Guide to Available Mathematical Software)
Classification Scheme for Mathematical and Statistical Software,

reference [5].

GAMS (and SLATEC) Classification Scheme
for
Mathematical and Statistical Software

Version 1.2 October 1983

A. Arithmetic, error analysis
A1. Integer
A2. Rational
A3. Real
A3A. Single precision
A3B. Double precision
A3C. Extended precision
A3D. Extended range
A4. Complex
A4A. Single precision
A4B. Double precision
A4C. Extended precision
A4D. Extended range
A5. Interval
A5A. Real
A5B. Complex
A6. Change of representation
A6A. Type conversion
A6B. Base conversion
A6C. Decomposition, construction
A7. Sequences (e.g., convergence acceleration)
B. Number theory
C. Elementary and special functions (search also class L5)
C1. Integer-valued functions (e.g., floor, ceiling, factorial, binomial coefficient)
C2. Powers, roots, reciprocals
C3. Polynomials
C3A. Orthogonal
C3A1. Trigonometric
C3A2. Chebyshev, Legendre
C3A3. Laguerre
C3A4. Hermite
C3B. Non-orthogonal
C4. Elementary transcendental functions
C4A. Trigonometric, inverse trigonometric
C4B. Exponential, logarithmic
C4C. Hyperbolic, inverse hyperbolic
C4D. Integrals of elementary transcendental functions
C5. Exponential and logarithmic integrals
C6. Cosine and sine integrals
C7. Gamma
C7A. Gamma, log gamma, reciprocal gamma
C7B. Beta, log beta
C7C. Psi function
C7D. Polygamma function
C7E. Incomplete gamma
C7F. Incomplete beta
C7G. Riemann zeta

C8. Error functions
C8A. Error functions, their inverses, integrals, including the normal
distribution function
C8B. Fresnel integrals
C8C. Dawson’s integral
C9. Legendre functions
C10. Bessel functions
C10A. J, Y, H-(1), H-(2)
C10A1. Real argument, integer order
C10A2. Complex argument, integer order
C10A3. Real argument, real order
C10A4. Complex argument, real order
C10A5. Complex argument, complex order
C10B. I, K
C10B1. Real argument, integer order
C10B2. Complex argument, integer order
C10B3. Real argument, real order
C10B4. Complex argument, real order
C10B5. Complex argument, complex order
C10C. Kelvin functions
C10D. Airy and Scorer functions
C10E. Struve, Anger, and Weber functions
C10F. Integrals of Bessel functions
C11. Confluent hypergeometric functions
C12. Coulomb wave functions
C13. Jacobian elliptic functions, theta functions
C14. Elliptic integrals
C15. Weierstrass elliptic functions
C16. Parabolic cylinder functions
C17. Mathieu functions
C18. Spheroidal wave functions
C19. Other special functions
D. Linear Algebra
D1. Elementary vector and matrix operations
D1A. Elementary vector operations
D1A1. Set to constant
D1A2. Minimum and maximum components
D1A3. Norm
D1A3A. L-1 (sum of magnitudes)
D1A3B. L-2 (Euclidean norm)
D1A3C. L-infinity (maximum magnitude)
D1A4. Dot product (inner product)
D1A5. Copy or exchange (swap)
D1A6. Multiplication by scalar
D1A7. Triad (a*x+y for vectors x,y and scalar a)
D1A8. Elementary rotation (Givens transformation)
D1A9. Elementary reflection (Householder transformation)
D1A10. Convolutions
D1B. Elementary matrix operations
D1B1. Set to zero, to identity
D1B2. Norm
D1B3. Transpose
D1B4. Multiplication by vector
D1B5. Addition, subtraction
D1B6. Multiplication
D1B7. Matrix polynomial
D1B8. Copy
D1B9. Storage mode conversion
D1B10. Elementary rotation (Givens transformation)
D1B11. Elementary reflection (Householder transformation)
D2. Solution of systems of linear equations (including inversion, LU and

related decompositions)
D2A. Real nonsymmetric matrices
D2A1. General
D2A2. Banded
D2A2A. Tridiagonal
D2A3. Triangular
D2A4. Sparse
D2B. Real symmetric matrices
D2B1. General
D2B1A. Indefinite
D2B1B. Positive definite
D2B2. Positive definite banded
D2B2A. Tridiagonal
D2B4. Sparse
D2C. Complex non-Hermitian matrices
D2C1. General
D2C2. Banded
D2C2A. Tridiagonal
D2C3. Triangular
D2C4. Sparse
D2D. Complex Hermitian matrices
D2D1. General
D2D1A. Indefinite
D2D1B. Positive definite
D2D2. Positive definite banded
D2D2A. Tridiagonal
D2D4. Sparse
D2E. Associated operations (e.g., matrix reorderings)
D3. Determinants
D3A. Real nonsymmetric matrices
D3A1. General
D3A2. Banded
D3A2A. Tridiagonal
D3A3. Triangular
D3A4. Sparse
D3B. Real symmetric matrices
D3B1. General
D3B1A. Indefinite
D3B1B. Positive definite
D3B2. Positive definite banded
D3B2A. Tridiagonal
D3B4. Sparse
D3C. Complex non-Hermitian matrices
D3C1. General
D3C2. Banded
D3C2A. Tridiagonal
D3C3. Triangular
D3C4. Sparse
D3D. Complex Hermitian matrices
D3D1. General
D3D1A. Indefinite
D3D1B. Positive definite
D3D2. Positive definite banded
D3D2A. Tridiagonal
D3D4. Sparse
D4. Eigenvalues, eigenvectors
D4A. Ordinary eigenvalue problems (Ax = (lambda) * x)
D4A1. Real symmetric
D4A2. Real nonsymmetric
D4A3. Complex Hermitian
D4A4. Complex non-Hermitian

D4A5. Tridiagonal
D4A6. Banded
D4A7. Sparse
D4B. Generalized eigenvalue problems (e.g., Ax = (lambda)*Bx)
D4B1. Real symmetric
D4B2. Real general
D4B3. Complex Hermitian
D4B4. Complex general
D4B5. Banded
D4C. Associated operations
D4C1. Transform problem
D4C1A. Balance matrix
D4C1B. Reduce to compact form
D4C1B1. Tridiagonal
D4C1B2. Hessenberg
D4C1B3. Other
D4C1C. Standardize problem
D4C2. Compute eigenvalues of matrix in compact form
D4C2A. Tridiagonal
D4C2B. Hessenberg
D4C2C. Other
D4C3. Form eigenvectors from eigenvalues
D4C4. Back transform eigenvectors
D4C5. Determine Jordan normal form
D5. QR decomposition, Gram-Schmidt orthogonalization
D6. Singular value decomposition
D7. Update matrix decompositions
D7A. LU
D7B. Cholesky
D7C. QR
D7D. Singular value
D8. Other matrix equations (e.g., AX+XB=C)
D9. Overdetermined or underdetermined systems of equations, singular systems,
pseudo-inverses (search also classes D5, D6, K1a, L8a)
E. Interpolation
E1. Univariate data (curve fitting)
E1A. Polynomial splines (piecewise polynomials)
E1B. Polynomials
E1C. Other functions (e.g., rational, trigonometric)
E2. Multivariate data (surface fitting)
E2A. Gridded
E2B. Scattered
E3. Service routines (e.g., grid generation, evaluation of fitted functions)
(search also class N5)
F. Solution of nonlinear equations
F1. Single equation
F1A. Smooth
F1A1. Polynomial
F1A1A. Real coefficients
F1A1B. Complex coefficients
F1A2. Nonpolynomial
F1B. General (no smoothness assumed)
F2. System of equations
F2A. Smooth
F2B. General (no smoothness assumed)
F3. Service routines (e.g., check user-supplied derivatives)
G. Optimization (search also classes K, L8)
G1. Unconstrained
G1A. Univariate
G1A1. Smooth function
G1A1A. User provides no derivatives

G1A1B. User provides first derivatives
G1A1C. User provides first and second derivatives
G1A2. General function (no smoothness assumed)
G1B. Multivariate
G1B1. Smooth function
G1B1A. User provides no derivatives
G1B1B. User provides first derivatives
G1B1C. User provides first and second derivatives
G1B2. General function (no smoothness assumed)
G2. Constrained
G2A. Linear programming
G2A1. Dense matrix of constraints
G2A2. Sparse matrix of constraints
G2B. Transportation and assignments problem
G2C. Integer programming
G2C1. Zero/one
G2C2. Covering and packing problems
G2C3. Knapsack problems
G2C4. Matching problems
G2C5. Routing, scheduling, location problems
G2C6. Pure integer programming
G2C7. Mixed integer programming
G2D. Network (for network reliability search class M)
G2D1. Shortest path
G2D2. Minimum spanning tree
G2D3. Maximum flow
G2D3A. Generalized networks
G2D3B. Networks with side constraints
G2D4. Test problem generation
G2E. Quadratic programming
G2E1. Positive definite Hessian (i.e. convex problem)
G2E2. Indefinite Hessian
G2F. Geometric programming
G2G. Dynamic programming
G2H. General nonlinear programming
G2H1. Simple bounds
G2H1A. Smooth function
G2H1A1. User provides no derivatives
G2H1A2. User provides first derivatives
G2H1A3. User provides first and second derivatives
G2H1B. General function (no smoothness assumed)
G2H2. Linear equality or inequality constraints
G2H2A. Smooth function
G2H2A1. User provides no derivatives
G2H2A2. User provides first derivatives
G2H2A3. User provides first and second derivatives
G2H2B. General function (no smoothness assumed)
G2H3. Nonlinear constraints
G2H3A. Equality constraints only
G2H3A1. Smooth function and constraints
G2H3A1A. User provides no derivatives
G2H3A1B. User provides first derivatives of function and constraints
G2H3A1C. User provides first and second derivatives of function and constraints
G2H3A2. General function and constraints (no smoothness assumed)
G2H3B. Equality and inequality constraints
G2H3B1. Smooth function and constraints
G2H3B1A. User provides no derivatives
G2H3B1B. User provides first derivatives of function and constraints
G2H3B1C. User provides first and second derivatives of function and constraints

G2H3B2. General function and constraints (no smoothness assumed)
G2I. Global solution to nonconvex problems
G3. Optimal control
G4. Service routines
G4A. Problem input (e.g., matrix generation)
G4B. Problem scaling
G4C. Check user-supplied derivatives
G4D. Find feasible point
G4E. Check for redundancy
G4F. Other
H. Differentiation, integration
H1. Numerical differentiation
H2. Quadrature (numerical evaluation of definite integrals)
H2A. One-dimensional integrals
H2A1. Finite interval (general integrand)
H2A1A. Integrand available via user-defined procedure
H2A1A1. Automatic (user need only specify required accuracy)
H2A1A2. Nonautomatic
H2A1B. Integrand available only on grid
H2A1B1. Automatic (user need only specify required accuracy)
H2A1B2. Nonautomatic
H2A2. Finite interval (specific or special type integrand including weight functions, oscillating and singular integrands, principal value integrals, splines, etc.)
H2A2A. Integrand available via user-defined procedure
H2A2A1. Automatic (user need only specify required accuracy)
H2A2A2. Nonautomatic
H2A2B. Integrand available only on grid
H2A2B1. Automatic (user need only specify required accuracy)
H2A2B2. Nonautomatic
H2A3. Semi-infinite interval (including e**(-x) weight function)
H2A3A. Integrand available via user-defined procedure
H2A3A1. Automatic (user need only specify required accuracy)
H2A3A2. Nonautomatic
H2A4. Infinite interval (including e**(-x**2)) weight function)
H2A4A. Integrand available via user-defined procedure
H2A4A1. Automatic (user need only specify required accuracy)
H2A4A2. Nonautomatic
H2B. Multidimensional integrals
H2B1. One or more hyper-rectangular regions
H2B1A. Integrand available via user-defined procedure
H2B1A1. Automatic (user need only specify required accuracy)
H2B1A2. Nonautomatic
H2B1B. Integrand available only on grid
H2B1B1. Automatic (user need only specify required accuracy)
H2B1B2. Nonautomatic
H2B2. Nonrectangular region, general region
H2B2A. Integrand available via user-defined procedure
H2B2A1. Automatic (user need only specify required accuracy)
H2B2A2. Nonautomatic
H2B2B. Integrand available only on grid
H2B2B1. Automatic (user need only specify required accuracy)
H2B2B2. Nonautomatic
H2C. Service routines (compute weight and nodes for quadrature formulas)
I. Differential and integral equations
I1. Ordinary differential equations
I1A. Initial value problems
I1A1. General, nonstiff or mildly stiff
I1A1A. One-step methods (e.g., Runge-Kutta)
I1A1B. Multistep methods (e.g., Adams’ predictor-corrector)
I1A1C. Extrapolation methods (e.g., Bulirsch-Stoer)

I1A2. Stiff and mixed algebraic-differential equations
I1B. Multipoint boundary value problems
I1B1. Linear
I1B2. Nonlinear
I1B3. Eigenvalue (e.g., Sturm-Liouville)
I1C. Service routines (e.g., interpolation of solutions, error handling)
I2. Partial differential equations
I2A. Initial boundary value problems
I2A1. Parabolic
I2A1A. One spatial dimension
I2A1B. Two or more spatial dimensions
I2A2. Hyperbolic
I2B. Elliptic boundary value problems
I2B1. Linear
I2B1A. Second order
I2B1A1. Poisson (Laplace) or Helmholz equation
I2B1A1A. Rectangular domain (or topologically rectangular in the coordinate system)
I2B1A1B. Nonrectangular domain
I2B1A2. Other separable problems
I2B1A3. Nonseparable problems
I2B1C. Higher order equations (e.g., biharmonic)
I2B2. Nonlinear
I2B3. Eigenvalue
I2B4. Service routines
I2B4A. Domain triangulation (search also class P2a2c1)
I2B4B. Solution of discretized elliptic equations
I3. Integral equations
J. Integral transforms
J1. Fast Fourier transforms (search class L10 for time series analysis)
J1A. One-dimensional
J1A1. Real
J1A2. Complex
J1A3. Trigonometric (sine, cosine)
J1B. Multidimensional
J2. Convolutions
J3. Laplace transforms
J4. Hilbert transforms
K. Approximation (search also class L8)
K1. Least squares (L-2) approximation
K1A. Linear least squares (search also classes D5, D6, D9)
K1A1. Unconstrained
K1A1A. Univariate data (curve fitting)
K1A1A1. Polynomial splines (piecewise polynomials)
K1A1A2. Polynomials
K1A1A3. Other functions (e.g., rational, trigonometric, user-specified)
K1A1B. Multivariate data (surface fitting)
K1A2. Constrained
K1A2A. Linear constraints
K1A2B. Nonlinear constraints
K1B. Nonlinear least squares
K1B1. Unconstrained
K1B1A. Smooth functions
K1B1A1. User provides no derivatives
K1B1A2. User provides first derivatives
K1B1A3. User provides first and second derivatives
K1B1B. General functions
K1B2. Constrained
K1B2A. Linear constraints
K1B2B. Nonlinear constraints
K2. Minimax (L-infinity) approximation

K3. Least absolute value (L-1) approximation
K4. Other analytic approximations (e.g., Taylor polynomial, Pade)
K5. Smoothing
K6. Service routines (e.g., mesh generation, evaluation of fitted functions)
(search also class N5)
L. Statistics, probability
L1. Data summarization
L1A. One univariate quantitative sample
L1A1. Ungrouped data
L1A1A. Location
L1A1B. Dispersion
L1A1C. Shape
L1A1D. Distribution, density
L1A2. Ungrouped data with missing values
L1A3. Grouped data
L1A3A. Location
L1A3B. Dispersion
L1A3C. Shape
L1C. One univariate qualitative (proportional) sample
L1E. Two or more univariate samples or one multivariate sample
L1E1. Ungrouped data
L1E1A. Location
L1E1B. Correlation
L1E2. Ungrouped data with missing values
L1E3. Grouped data
L1F. Two or more multivariate samples
L2. Data manipulation (search also class N)
L2A. Transform (search also class N6 for sorting, ranking)
L2B. Group
L2C. Sample
L2D. Subset
L3. Graphics (search also class Q)
L3A. Histograms
L3B. Distribution functions
L3C. Scatter diagrams
L3C1. Y vs. X
L3C2. Symbol plots
L3C3. Multiple plots
L3C4. Probability plots
L3C4B. Beta, binomial
L3C4C. Cauchy, chi-squared
L3C4D. Double exponential
L3C4E. Exponential, extreme value
L3C4F. F distribution
L3C4G. Gamma, geometric
L3C4H. Halfnormal
L3C4L. Lambda, logistic, lognormal
L3C4N. Negative binomial, normal
L3C4P. Pareto, Poisson
L3C4T. t distribution
L3C4U. Uniform
L3C4W. Weibull
L3C5. Time series plots (X(i) vs. i, vertical, lag)
L3D. EDA graphics
L4. Elementary statistical inference, hypothesis testing
L4A. One univariate quantitative sample
L4A1. Ungrouped data
L4A1A. Parameter estimation
L4A1A2. Binomial
L4A1A5. Extreme value
L4A1A14. Normal

L4A1A16. Poisson
L4A1A21. Uniform
L4A1A23. Weibull
L4A1B. Distribution-free (nonparametric) analysis
L4A1C. Goodness-of-fit tests
L4A1D. Tests on sequences of numbers
L4A1E. Density and distribution function estimation
L4A1F. Tolerance limits
L4A2. Ungrouped data with missing values
L4A3. Grouped data
L4A3A. Parameter estimation
L4A3A14. Normal
L4B. Two or more univariate quantitative samples
L4B1. Ungrouped data
L4B1A. Parameter estimation
L4B1A14. Normal
L4B1B. Distribution-free (nonparametric) analysis
L4B2. Ungrouped data with missing values
L4B3. Grouped data
L4C. One univariate qualitative (proportional) sample
L4D. Two or more univariate samples
L4E. One multivariate sample
L4E1. Ungrouped data
L4E1A. Parameter estimation
L4E1A14. Normal
L4E1B. Distribution-free (nonparametric) analysis
L4E2. Ungrouped data with missing values
L4E2A. Parameter estimation
L4E2B. Distribution-free (nonparametric) analysis
L4E3. Grouped data
L4E3A. Parameter estimation
L4E3A14. Normal
L4E3B. Distribution-free (nonparametric) analysis
L4E4. Two or more multivariate samples
L4E4A. Parameter estimation
L4E4A14. Normal
L5. Function evaluation (search also class C)
L5A. Univariate
L5A1. Cumulative distribution functions, probability density functions
L5A1B. Beta, binomial
L5A1C. Cauchy, chi-squared
L5A1D. Double exponential
L5A1E. Error function, exponential, extreme value
L5A1F. F distribution
L5A1G. Gamma, general, geometric
L5A1H. Halfnormal, hypergeometric
L5A1K. Kolmogorov-Smirnov
L5A1L. Lambda, logistic, lognormal
L5A1N. Negative binomial, normal
L5A1P. Pareto, Poisson
L5A1T. t distribution
L5A1U. Uniform
L5A1W. Weibull
L5A2. Inverse cumulative distribution functions, sparsity functions
L5A2B. Beta, binomial
L5A2C. Cauchy, chi-squared
L5A2D. Double exponential
L5A2E. Exponential, extreme value
L5A2F. F distribution
L5A2G. Gamma, general, geometric
L5A2H. Halfnormal

L5A2L. Lambda, logistic, lognormal
L5A2N. Negative binomial, normal, normal scores
L5A2P. Pareto, Poisson
L5A2T. t distribution
L5A2U. Uniform
L5A2W. Weibull
L5B. Multivariate
L5B1. Cumulative distribution functions, probability density functions
L5B1N. Normal
L6. Pseudo-random number generation
L6A. Univariate
L6A2. Beta, binomial, Boolean
L6A3. Cauchy, chi-squared
L6A4. Double exponential
L6A5. Exponential, extreme value
L6A6. F distribution
L6A7. Gamma, general (continuous, discrete) distributions, geometric
L6A8. Halfnormal, hypergeometric
L6A9. Integers
L6A12. Lambda, logical, logistic, lognormal
L6A14. Negative binomial, normal
L6A15. Order statistics
L6A16. Pareto, permutations, Poisson
L6A19. Samples, stable distribution
L6A20. t distribution, time series, triangular
L6A21. Uniform
L6A22. Von Mises
L6A23. Weibull
L6B. Multivariate
L6B3. Contingency table, correlation matrix
L6B13. Multinomial
L6B14. Normal
L6B15. Orthogonal matrix
L6B21. Uniform
L6C. Service routines (e.g., seed)
L7. Experimental design, including analysis of variance
L7A. Univariate
L7A1. One-way analysis of variance
L7A1A. Parametric analysis
L7A1A1. Contrasts, multiple comparisons
L7A1A2. Analysis of variance components
L7A1B. Distribution-free (nonparametric) analysis
L7A2. Balanced multiway design
L7A2A. Complete
L7A2A1. Parametric analysis
L7A2A1A. Two-way
L7A2A1B. Factorial
L7A2A1C. Nested
L7A2A2. Distribution-free (nonparametric) analysis
L7A2B. Incomplete
L7A2B1. Parametric analysis
L7A2B1A. Latin square
L7A2B1B. Lattice designs
L7A2B2. Distribution-free (nonparametric) analysis
L7A3. Analysis of covariance
L7A4. General linear model (unbalanced design)
L7A4A. Parametric analysis
L7A4B. Distribution-free (nonparametric) analysis
L7B. Multivariate
L8. Regression (search also classes G, K)
L8A. Linear least squares (L-2) (search also classes D5, D6, D9)

L8A1. Simple
L8A1A. Ordinary
L8A1A1. Unweighted
L8A1A1A. No missing values
L8A1A1B. Missing values
L8A1A2. Weighted
L8A1B. Through the origin
L8A1C. Errors in variables
L8A1D. Calibration (inverse regression)
L8A2. Polynomial
L8A2A. Not using orthogonal polynomials
L8A2A1. Unweighted
L8A2A2. Weighted
L8A2B. Using orthogonal polynomials
L8A2B1. Unweighted
L8A2B2. Weighted
L8A3. Piecewise polynomial (i.e. multiphase or spline)
L8A4. Multiple
L8A4A. Ordinary
L8A4A1. Unweighted
L8A4A1A. No missing values
L8A4A1B. Missing values
L8A4A1C. From correlation data
L8A4A1D. Using principal components
L8A4A1E. Using preference pairs
L8A4A2. Weighted
L8A4B. Errors in variables
L8A4D. Logistic
L8A5. Variable selection
L8A6. Regression design
L8A7. Several multiple regressions
L8A8. Multivariate
L8A9. Diagnostics
L8A10. Hypothesis testing, inference
L8A10A. Lack-of-fit tests
L8A10B. Analysis of residuals
L8A10C. Inference
L8B. Biased (ridge)
L8C. Linear least absolute value (L-1)
L8D. Linear minimax (L-infinity)
L8E. Robust
L8F. EDA
L8G. Nonlinear
L8G1. Unweighted
L8G1A. Derivatives not supplied
L8G1B. Derivatives supplied
L8G2. Weighted
L8G2A. Derivatives not supplied
L8G2B. Derivatives supplied
L8H. Service routines
L9. Categorical data analysis
L9A. 2-by-2 tables
L9B. Two-way tables
L9C. Log-linear model
L9D. EDA (e.g., median polish)
L10. Time series analysis (search also class L3c5 for time series graphics)
L10A. Transformations, transforms (search also class J1)
L10B. Smoothing, filtering
L10C. Autocorrelation analysis
L10D. Complex demodulation
L10E. ARMA and ARIMA modeling and forecasting

L10E1. Model and parameter estimation
L10E2. Forecasting
L10F. Spectral analysis
L10G. Cross-correlation analysis
L10G1. Parameter estimation
L10G2. Forecasting
L11. Correlation analysis
L12. Discriminant analysis
L13. Factor analysis
L13A. Principal components analysis
L14. Cluster analysis
L14A. Unconstrained
L14A1. Nested
L14A1A. Joining (e.g., single link)
L14A1B. Divisive
L14A2. Non-nested
L14B. Constrained
L14B1. One-dimensional
L14B2. Two-dimensional
L14C. Display
L15. Life testing, survival analysis
M. Simulation, stochastic modeling (search also classes L6, L10)
M1. Simulation
M1A. Discrete
M1B. Continuous (Markov models)
M2. Queueing
M3. Reliability
M3A. Quality control
M3B. Electrical network
M4. Project optimization (e.g., PERT)
N. Data handling (search also class L2)
N1. Input, output
N2. Bit manipulation
N3. Character manipulation
N4. Storage management (e.g., stacks, heaps, trees)
N5. Searching
N5A. Extreme value
N5B. Insertion position
N5C. On a key
N6. Sorting
N6A. Internal
N6A1. Passive (i.e. construct pointer array, rank)
N6A1A. Integer
N6A1B. Real
N6A1B1. Single precision
N6A1B2. Double precision
N6A1C. Character
N6A2. Active
N6A2A. Integer
N6A2B. Real
N6A2B1. Single precision
N6A2B2. Double precision
N6A2C. Character
N6B. External
N7. Merging
N8. Permuting
O. Symbolic computation
P. Computational geometry (search also classes G, Q)
P1. One dimension
P2. Two dimensions
P2A. Points, lines

P2A1. Relationships
P2A1A. Closest and farthest points
P2A1B. Intersection
P2A2. Graph construction
P2A2A. Convex hull
P2A2B. Minimum spanning tree
P2A2C. Region partitioning
P2A2C1. Triangulation
P2A2C2. Voronoi diagram
P2B. Polygons (e.g., intersection, hidden line problems)
P2C. Circles
P3. Three dimensions
P3A. Points, lines, planes
P3B. Polytopes
P3C. Spheres
P4. More than three dimensions
Q. Graphics (search also classes L3, P)
Q1. Line printer plotting
R. Service routines
R1. Machine-dependent constants
R2. Error checking (e.g., check monotonicity)
R3. Error handling
R3A. Set criteria for fatal errors
R3B. Set unit number for error messages
R3C. Other utility programs
R4. Documentation retrieval
S. Software development tools
S1. Program transformation
S2. Static analysis
S3. Dynamic analysis
Z. Other

****

APPENDIX B. MACHINE CONSTANTS

The SLATEC Common Math Library uses three functions for keeping machine constants. In order to keep the source code for the Library as portable as possible, no other Library routines should attempt to DATA load machine dependent constants. Due to the subtlety of trying to calculate machine constants at run time in a manner that yields correct constants for all possible computers, no Library routines should attempt to calculate them. Routines I1MACH, R1MACH, and D1MACH in the SLATEC Common Math Library are derived from the routines of these names in the Bell Laboratories' PORT Library and should be called whenever machines constants are needed. These functions are DATA loaded with carefully determined constants of type integer, single precision, and double precision, respectively, for a wide range of computers. Each is called with one input argument to indicate which constant is desired. The appropriate Fortran statements are:

For integer constants:

```
      INTEGER I1MACH, I
      I = I1MACH(N)                 where 1 .LE. N .LE. 16
```

For single precision constants:

```
      REAL R1MACH, R
```

```
      R = R1MACH(N)                 where 1 .LE. N .LE. 5
```

For double precision constants:

```
      DOUBLE PRECISION D1MACH, D
      D = D1MACH(N)                 where 1 .LE. N .LE. 5
```

The different constants that can be retrieved will be explained below after we give a summary of the floating point arithmetic model which they characterize.

The PORT and SLATEC machine constant routines acknowledge that a computer can have some minor flaws in how it performs arithmetic and that the purpose of machine constant routines is to keep other library routines out of trouble. For example, a computer may have a 48-bit coefficient, but due to round-off or other deficiencies may be able to perform only 47-bit (or even 46-bit) arithmetic reliably. A machine can also misbehave at the extreme ends of its exponent range. The machine constants are chosen to describe a subset of the floating point numbers of a computer on which operations such as addition, subtraction, multiplication, reciprocation, and comparison work as your intuition would expect. If the actual performance of the machine is such that results fall into the "expected" intervals of the subset floating point system, then the usual forms of error analysis will apply. For details, see [7].

The machine constants normally cannot be determined by reading a computer's hardware reference manual. Such manuals tell the range and representation of floating point numbers but usually do not describe the errors in the floating point addition, subtraction, multiplication, reciprocation, or division units. The constants for I1MACH, R1MACH, and D1MACH are found by doing extensive testing using operands on which the hardware is most likely to fail. Failure is most likely to occur at the extreme ends of the exponent range and near powers of the number base. If such failures are relatively minor, we can choose machine constants for I1MACH, R1MACH, and D1MACH to restrict the domain of floating point numbers to a subset on which arithmetic operations work.

The subset model of floating point arithmetic is characterized by four parameters:

- B   the number base or radix. This is usually 2 or 16.
- T   the number of digits in base B of the coefficient of the floating point number.
- EMIN the smallest (most negative) exponent (power of B)
- EMAX the largest exponent (power of B)

A floating point number is modeled as FRACTION*(B**EXP) where EXP falls between EMIN and EMAX and the FRACTION is of the form

```
     + or - ( f(1)*B**(-1) + ... + f(T)*B**(-T) )

     with f(1) in the range 1 to B-1 inclusive and
          f(2) through f(T) in the range 0 to B-1 inclusive.
```

In this model the fraction has the radix point at the left end. Some computers have their radix point at the right end so that when their representation is mapped onto this model, they appear to have an unbalanced exponent range (i.e., EMIN is not close to the negative of EMAX). If the computer cannot correctly calculate results near underflow, EMIN is increased to a more conservative value. Likewise, if the computer cannot correctly calculate results near overflow, EMAX is decreased. If a base 2 machine with a 48-bit fraction is unable to calculate 48-bit results due to hardware round-off, T may be set to 47 (or even 46) to account for the loss of accuracy.

The complete set of machine constants (including those not related to floating point arithmetic) are:

I/O Unit Numbers

I1MACH(1) = the FORTRAN unit number for the standard input device.

I1MACH(2) = the FORTRAN unit number for the standard output device.

I1MACH(3) = the FORTRAN unit number for the standard punch device.

I1MACH(4) = the FORTRAN unit number for the standard error message device.

Word Properties

I1MACH(5) = the number of bits per integer storage unit.

I1MACH(6) = the number of characters per integer storage unit.

Integer Arithmetic

I1MACH(7) = the base or radix for integer arithmetic.

I1MACH(8) = the number of digits in radix I1MACH(7) used in integer arithmetic.

I1MACH(9) = the largest magnitude integer for which the machine and compiler perform the complete set of arithmetic operations.

Floating Point Arithmetic

I1MACH(10) = the base or radix for floating point arithmetic. This is the B of the floating point model.

Single Precision Arithmetic

I1MACH(11) = the number of digits in radix I1MACH(10) used in single precision arithmetic. This is the T in the floating point model.

I1MACH(12) = the most negative usable exponent short of underflow of radix I1MACH(10) for a single precision number. This is the EMIN in the floating point model.

I1MACH(13) = the largest usable exponent short of overflow of radix I1MACH(10) for a single precision number. This is the EMAX in the floating point model.

Double Precision Arithmetic

I1MACH(14) = the number of digits in radix I1MACH(10) used in double precision arithmetic. This is the T of the floating point model.

I1MACH(15) = the most negative usable exponent short of underflow of radix I1MACH(10) for a double precision number. This is the EMIN of the floating point model.

I1MACH(16) = the largest usable exponent short of overflow of radix I1MACH(10) for a double precision number. This is the EMAX of the floating point model.

Special Single Precision Values

R1MACH(1) = B**(EMIN-1). This is the smallest, positive, single precision number in the range for safe, accurate arithmetic.

R1MACH(2) = B**EMAX*(1-B**(-T)). This is the largest, positive, single precision number in the range for safe, accurate arithmetic.

R1MACH(3) = B**(-T). This is the smallest relative spacing between two adjacent single precision numbers in the floating point model. This constant is not machine epsilon; see below for machine epsilon.

R1MACH(4) = B**(1-T). This is the largest relative spacing between two adjacent single precision numbers in the floating point model. Any two single precision numbers that have a greater relative spacing than R1MACH(4) can be compared correctly (with operators like .EQ. or .LT.). This constant is an upper bound on theoretical machine epsilon.

R1MACH(5) = logarithm to base ten of the machine’s floating point number base.

Special Double Precision Values

D1MACH(1) = B**(EMIN-1). This is the smallest, positive, double precision numbers in the range for safe, accurate arithmetic.

D1MACH(2) = B**EMAX*(1-B**(-T)). This is the largest, positive, double precision number in the range for safe, accurate arithmetic.

D1MACH(3) = B**(-T). This is the smallest relative spacing between two adjacent double precision numbers in the floating point model. This constant is not machine epsilon; see below for machine epsilon.

D1MACH(4) = B**(1-T). This is the largest relative spacing between two adjacent double precision numbers in the floating point model. Any two double precision numbers that have a greater relative spacing than D1MACH(4) can be compared correctly (with operators like .EQ. or .LT.). This constant is an upper bound on theoretical machine epsilon.

D1MACH(5) = logarithm to base ten of the machine’s floating point number base.

In theory, all of the R1MACH and D1MACH values can be calculated from I1MACH values; however, they are provided (1) to save having to calculate them and (2) to avoid rousing any bugs in the exponentiation (** operator) or logarithm routines.

Machine epsilon (the smallest number that can be added to 1.0 or 1.0D0 that yields a result different from 1.0 or 1.0D0) is not one of the special values that comes from this model. If the purpose of machine epsilon is to decide when iterations have converged, the proper constants to use are R1MACH(4) or D1MACH(4). These may be slightly larger than machine epsilon; however, trying to iterate to smaller relative differences may not be possible due to hardware round-off error.

The Fortran standard requires that the amount of storage assigned to an INTEGER and a REAL be the same. Thus, the number of bits that can be used to represent an INTEGER will almost always be larger than the number of bits in the mantissa of a REAL. In converting from an INTEGER to a REAL, some machines will correctly round or truncate, but some will not. Authors are therefore advised to check the magnitude of INTEGERs and not attempt to convert INTEGERs to REALs that can not be represented exactly as REALs. Similar problems can occur when converting INTEGERs to DOUBLEs.

****

APPENDIX C. ERROR HANDLING

Authors of Library routines must use at least the first and preferably both of the following techniques to handle errors that their routines detect.

1. One argument, preferably the last, in the calling sequence must be an error flag if the routine can detect errors. This is an integer variable to which a value is assigned before returning to the caller. A value of zero means the routine completed successfully. A positive value (preferably in the range 1 to 999) should be used to indicate potential, partial, or total failure. Separate values should be used for distinct conditions so that the caller can determine the nature of the failure. Of course, the possible values of this error flag and their meanings must be documented in the description section of the prologue of the routine.

2. In addition to returning an error flag, the routine can supply more information by writing an error message via a call to XERMSG. XERMSG has an error number as one of its arguments, and the same value that will be returned in the error flag argument must be used in calling XERMSG.

XERMSG is part of the SLATEC Common Math Library error handling package which consists of a number of routines. It is not necessary for authors to learn about the entire package. Instead we summarize here a few aspects of the package that an author must know in order to use XERMSG correctly.

1. Although XERMSG supports three levels of severity (warning, recoverable error, and fatal error), be sparing in the use of fatal errors. XERMSG will terminate the program for fatal errors but may return for recoverable errors, and will definitely return after warning messages. An error should be designated fatal only if returning to the caller is likely to be disastrous (e.g. result in an infinite loop).

2. The error handling package remembers the value of the error number and has an entry point whereby the user can retrieve the most recent error number. Successive calls to XERMSG replace this retained value. In the case of warning messages, it is permissible to issue multiple warnings. In the case of a recoverable error, no additional calls to XERMSG must be made by the Library routine before returning to the caller since the caller must be given a chance to retrieve and clear the error number (and error condition) from the error handling package. In particular, if the user calls Library routine X and X calls a lower level Library Y, it is permissible for Y

to call XERMSG, but after it returns to X, X must be careful to note any recoverable errors detected in Y and not make any additional calls to XERMSG in that case. In practice, it would be simpler if subsidiary routines did not call XERMSG but only returned error flags indicating a serious problem. Then the highest level Library routine could call XERMSG just before returning to its caller. This also allows the highest level routine the most flexibility in assigning error numbers and assures that all possible error conditions are documented in one prologue rather than being distributed through prologues of subsidiary routines.

Below we describe only subroutine XERMSG. Other routines in the error handling package are described in their prologues and in Reference [4]. The call to XERMSG looks like

Template: CALL XERMSG (library, routine, message, errornumber, level)

```
Example:   CALL XERMSG ('SLATEC', 'MMPY',
         1   'The order of the matrix exceeds the row dimension', 3, 1)
```

where the meaning of the arguments is

library — A character constant (or character variable) with the name of the library. This will be 'SLATEC' for the SLATEC Common Math Library. The error handling package is general enough to be used by many libraries simultaneously, so it is desirable for the routine that detects and reports an error to identify the library name as well as the routine name.

routine — A character constant (or character variable) with the name of the routine that detected the error. Usually it is the name of the routine that is calling XERMSG. There are some instances where a user callable library routine calls lower level subsidiary routines where the error is detected. In such cases it may be more informative to supply the name of the routine the user called rather than the name of the subsidiary routine that detected the error.

message — A character constant (or character variable) with the text of the error or warning message. In the example below, the message is a character constant that contains a generic message.

```
      CALL XERMSG ('SLATEC', 'MMPY',
     *   'The order of the matrix exceeds the row dimension',
     *   3, 1)
```

It is possible (and is sometimes desirable) to generate a specific message--e.g., one that contains actual numeric values. Specific numeric values can be converted into character strings using formatted WRITE statements into character variables. This is called standard Fortran internal file I/O and is exemplified in the first three lines of the following example. You can also catenate substrings of characters to construct the error message. Here is an example showing the use of both writing to an internal file and catenating character strings.

```
      CHARACTER*5 CHARN, CHARL
      WRITE (CHARN,10) N
      WRITE (CHARL,10) LDA
   10 FORMAT(I5)
      CALL XERMSG ('SLATEC', 'MMPY', 'The order'//CHARN//
     *   ' of the matrix exceeds its row dimension of'//
```

```
     *   CHARL, 3, 1)
```

There are two subtleties worth mentioning. One is that the // for character catenation is used to construct the error message so that no single character constant is continued to the next line. This avoids confusion as to whether there are trailing blanks at the end of the line. The second is that by catenating the parts of the message as an actual argument rather than encoding the entire message into one large character variable, we avoid having to know how long the message will be in order to declare an adequate length for that large character variable. XERMSG calls XERPRN to print the message using multiple lines if necessary. If the message is very long, XERPRN will break it into pieces of 72 characters (as requested by XERMSG) for printing on multiple lines. Also, XERMSG asks XERPRN to prefix each line with ' * ' so that the total line length could be 76 characters. Note also that XERPRN scans the error message backwards to ignore trailing blanks. Another feature is that the substring '$$' is treated as a new line sentinel by XERPRN. If you want to construct a multiline message without having to count out multiples of 72 characters, just use '$$' as a separator. '$$' obviously must occur within 72 characters of the start of each line to have its intended effect since XERPRN is asked to wrap around at 72 characters in addition to looking for '$$'.

errornumber — An integer value that is chosen by the library routine's author. It must be in the range 1 to 999. Each distinct error should have its own error number. These error numbers should be described in the machine readable documentation for the routine. The error numbers need be unique only within each routine, so it is reasonable for each routine to start enumerating errors from 1 and proceeding to the next integer.

level — An integer value in the range 0 to 2 that indicates the level (severity) of the error. Their meanings are

0 A warning message. This is used if it is not clear that there really is an error, but the user's attention may be needed.

1 A recoverable error. This is used even if the error is so serious that the routine cannot return any useful answer. If the user has told the error package to return after recoverable errors, then XERMSG will return to the Library routine which can then return to the user's routine. The user may also permit the error package to terminate the program upon encountering a recoverable error.

2 A fatal error. XERMSG will not return to its caller after it receives a fatal error. This level should hardly ever be used; it is much better to allow the user a chance to recover. An example of one of the few cases in which it is permissible to declare a level 2 error is a reverse communication Library routine that is likely to be called repeatedly until it integrates across some interval. If there is a serious error in the input such that another step cannot be taken and the Library routine is called again without the input error having been corrected by the caller, the Library routine will probably be called forever with improper input. In this case, it is reasonable to declare the error to be fatal.

Each of the arguments to XERMSG is input; none will be modified by XERMSG. A routine may make multiple calls to XERMSG with warning level messages; however, after a call to XERMSG with a recoverable error, the routine should return to the user. Do not try to call XERMSG with a second recoverable error after the first recoverable error because the error package saves the error number. The user can retrieve this error number by calling another entry point in the error handling package and then clear the error number when recovering from the error. Calling XERMSG in succession causes the old error number to be overwritten by the latest error number. This is considered harmless for error numbers associated with warning messages but must not be done for error numbers of serious errors. After a call to XERMSG with a recoverable error, the user must be given a chance to call NUMXER or XERCLR to retrieve or clear the error number.

****

APPENDIX D. DISTRIBUTION FILE STRUCTURE

The source files of the SLATEC library distribution tape are ASCII text files. Each line image consists of exactly 80 characters. The first file of the tape is text file describing the contents of the tape.

The SLATEC source code file has the following characteristics.

1. All subprograms in the file are in alphabetic order. The collating sequence is 0 through 9 and then A through Z.

2. Before each subprogram, of name for example XYZ, there is a line starting in column 1 with

   *DECK XYZ

   This allows the source file to be used as input for a source code maintenance program.

3. No comments other than the *DECK lines appear between subprograms.

****

APPENDIX E. SUGGESTED FORMAT FOR A SLATEC SUBPROGRAM

A template embodying the suggested format for a SLATEC subprogram is given below. As elsewhere in this Guide, the caret (^) denotes a required blank character. These should be replaced with blanks AFTER filling out the template. The template itself begins with the *DECK line, below. All occurrences of "NAME" are to be replaced with the actual name of the subprogram, of course. Items in brackets [] are either explanations or optional information. Lines that do not have C or * in column 1 are explanatory remarks that are intended to be deleted by the programmer. In all cases where "or" is used, exactly one of the indicated forms must occur.

Lines that begin with C*** are standard SLATEC lines. These must be in the indicated order. See Section 8 of this Guide for information on required vs optional lines. In all but the C***DESCRIPTION section, the exact spacing and punctuation are as mandated by this Guide. Spacing within this section is only suggestive, except as noted below. The SLATEC standard mandates that no other comments may begin "C***". All other lines between the C***BEGIN^PROLOGUE and the C***END^PROLOGUE must be comment lines with "C^" in columns 1-2.

Within the C***DESCRIPTION section, lines that begin with "C^*" are for the LLNL LDOC standard [9]. If present, these lines must be exactly as given here. They should be in the indicated order. All other lines in this section must have "C^^" in columns 1-3.

In the Arguments subsection, each argument must be followed by exactly one argument qualifier. The qualifier must be preceded by a colon and followed by at least one blank. The allowable qualifiers and their meanings follow.

Qualifier	Meaning
:IN	input variable. Must be set by the user prior to the call (unless otherwise indicated). Must NOT be changed by the routine under any circumstances.
:OUT	output variable. Values will be set by the routine. Must be initialized before first usage in the routine.
:INOUT	input/output variable. Must be set by the user prior to the call (as indicated in argument description); value(s) may be set or changed by the routine.
:WORK	workspace. Simply working storage required by the routine. Need not be set prior to the call and will not contain information meaningful to the user on return. (Some routines require the contents of a work array to remain unchanged between successive calls. Such usage should be carefully explained in the argument description.)
:EXT	external procedure. The actual argument must be the name of a SUBROUTINE, FUNCTION, or BLOCK DATA subprogram. It must appear in an EXTERNAL statement in the calling program. The argument description following should precisely specify the expected calling sequence.
:DUMMY	dummy argument. Need not be set by user; will not be referenced by the routine. [Use discouraged!]

To avoid potential problems with automatic formatting of argument descriptions, none of these key words should appear anywhere else in the text immediately preceded by a colon.

NOTES:

1. Make a template by copying the following "*DECK^NAME" through "^^^^^^END" lines, inclusive, from this Guide.
2. You will probably want to customize this template by filling in the C***AUTHOR section and adding other things you customarily include in your prologues. If all of your routines are in the same category(ies), you may wish to fill in the C***CATEGORY and C***KEYWORDS sections, too. Be sure to eliminate the brackets [].
3. Be sure to delete the "C***SUBSIDIARY" line if this is a user-callable routine.

```
*DECK^NAME
^^^^^^SUBROUTINE^NAME[^(ARG1[,^ARG2[,^...]])]                or
^^^^^^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])                    or
^^^^^^COMPLEX^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])            or
^^^^^^DOUBLE^PRECISION^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])  or
^^^^^^INTEGER^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])            or
^^^^^^REAL^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])               or
^^^^^^LOGICAL^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])            or
^^^^^^CHARACTER[*len]^FUNCTION^NAME^(ARG1[,^ARG2[,^...]])
```

```
C***BEGIN^PROLOGUE^^NAME
C***SUBSIDIARY
C***PURPOSE^^Brief (1-6 lines) summary of the purpose of this routine.
C^^^^^^^^^^^^^(To best fit LDOC standards, first line should be suitable
C^^^^^^^^^^^^^for a table of contents entry for this routine.)
C***LIBRARY^^^SLATEC[^(Package)]
C***CATEGORY^^CAT1[,^CAT2]
C***TYPE^^^^^^SINGLE PRECISION^(NAME-S,^DNAME-D)
C***KEYWORDS^^KEY1[,^KEY2[,
C^^^^^^^^^^^^^^MORE]]
C***AUTHOR^^Last-name[,^First-name[,^(Organization)]][
C^^^^^^^^^^^^^^More information][
C^^^^^^^^^^^^Second-last-name[,^First-name[,^(Organization)]][
C^^^^^^^^^^^^^^More information]]
C***DESCRIPTION
C^^
C^*Usage:
C^^ This subsection should have declarations for all arguments to the
C^^   routine and a model call of the routine.  Use the actual names of
C^^   the arguments here. Ideally, arguments should be named in a way
C^^   that suggests their meaning.
C^^ The following example illustrates the use of dummy identifiers (in
C^^   lower case) to indicate that the required size of an array is
C^^   some function of the values of the other arguments.  This may not
C^^   be legal Fortran, but should be easier for a knowledgeable user
C^^   to understand than giving the required size somewhere else.
C^^
C^^      INTEGER  M, N, MDIMA, IERR
C^^      PARAMETER  (nfcns = 6, nwks = 3*nfcns+M+7)
C^^      REAL  X(nmax), A(MDIMA,nmax), FCNS(nfcns), WKS(nwks)
C^^
C^^      CALL NAME (M, N, X, A, MDIMA, FCNS, WKS, IERR)
C^^
C^*Arguments:
C^^ Arguments should be described in exactly the same order as in the
C^^   CALL list.  Include any restrictions, etc.
C^^ The following illustrates the recommended form of argument descrip-
C^^   tions for the example given above.  Note the use of qualifiers.
C^^
C^^   M :IN^    is the number of data points.
C^^
C^^   N :IN^    is the number of unknowns.  (Must have  0.lt.N.le.M .)
C^^
C^^   X :IN^    is a real array containing ...
C^^             (The dimensioned length of X must be at least N.)
C^^
C^^   A :INOUT^ should contain ... on input; will be destroyed on
C^^             return.  (The second dimension of A must be at least N.)
C^^
C^^   MDIMA:IN^ is the first dimension of array A.
C^^             (Must have  M.le.MDIMA .)
C^^
C^^   FCNS:OUT^ will contain the six summary functions based on ...
C^^
C^^   WKS:WORK^ is a real array of working storage.  Its length is a
C^^             function of the length of FCNS and the number of data
C^^             points, as indicated above.
C^^
C^^   IERR:OUT^ is an error flag with the following possible values:
C^^             Normal return:
C^^                IERR = 0  (no errors)
```

```
C^^             Warning error:
C^^                IERR > 0  means what?
C^^             "Recoverable" errors:
C^^                IERR =-1  if M < 1 or N < 1 .
C^^                IERR =-2  if M > MDIMA .
C^^                IERR =-3  means what?
C^^
C^*Function^Return^Values:
C^^ This subsection is present only in a FUNCTION subprogram.
C^^ In case of an integer- or character-valued function with a discrete
C^^   set of values, list all possible return values, with their
C^^   meanings, in the following form.  [The colon is significant.]
C^^      value : meaning
C^^   Otherwise, something of the following sort is acceptable.
C^^      SQRT : the square root of X.
C^^
C^*Description:
C^^ One or more paragraphs describing the intended routine use,
C^^   dependencies on other routines, etc.  Specific algorithm
C^^   descriptions could go here, if appropriate.
C^^ The emphasis should be on information useful to a user (as opposed
C^^   to developer or maintainer) of the routine.
C^^
C^*Examples:
C^^ Detailed examples of usage would go here, if desired.
C^^
C^*Accuracy:
C^^ This optional subsection contains notes on the accuracy or
C^^   precision of the results computed by the routine.
C^^
C^*Cautions:
C^^ List any known problems or potentially hazardous side effects
C^^   that are not otherwise described, such as not being safe for
C^^   multiprocessing or exceptional cases for arguments.
C^^   (Ideally, there should be none in a SLATEC routine!)
C^^
C^*See^Also:
C^^ This subsection would contain notes that refer to other library
C^^   routines that interrelate to this routine in important ways.
C^^   Examples include a solver for a LU factorization routine or an
C^^   evaluator for an interpolation or approximation routine.
C^^ This subsection may amplify information in the C***SEE ALSO
C^^   section, below, which should appear only if the prologue of the
C^^   listed routine(s) contains documentation for this routine.
C^^
C^*Long^Description:
C^^ An optional subsection containing much more detailed information.
C^^
C***SEE^ALSO^^RTN1[,^RTN2[,
C^^^^^^^^^^^^^^RTNn]]
C***REFERENCES^^(NONE)              or
C***REFERENCES^^1. Reference 1 ...
C^^^^^^^^^^^^^^^^^^Continuation of reference 1.
C^^^^^^^^^^^^^^^^2. Reference 2 ...
C^^^^^^^^^^^^^^^^^^Continuation of reference 2.
C***ROUTINES^CALLED^^(NONE)         or
C***ROUTINES^CALLED^^RTN1[,^RTN2[,
C^^^^^^^^^^^^^^^^^^^^RTNn]]
   [Do not include standard Fortran intrinsics or externals.]
C***COMMON^BLOCKS^^^^BLOCK1[,^BLOCK2]
C***REVISION^HISTORY^^(YYMMDD)
```

```
   [ This section should contain a record of the origin and ]
   [ modification history of this routine.                  ]
C^^^871105^^DATE^WRITTEN
C^^^880121^^Various editorial changes.       (Version 6)
C^^^881102^^Converted to new SLATEC format.  (Version 7)
C^^^881128^^Various editorial changes.       (Version 8)
C^
C***END^PROLOGUE^^NAME
C
C*Internal Notes:
C   Implementation notes that explain details of the routine's design
C     or coding, tricky dependencies that might trip up a maintainer
C     later, environmental assumptions made, alternate designs that
C     were considered but not used, etc.
C   Details on contents of common blocks referenced, locks used, etc.,
C     would go here.
C   Emphasis is on INTERNALLY useful information.
C
C**End
C
C  Additional comments that are not appropriate even for an internal
C  document, but which the programmer feels should precede declarations.
C
C  Declare arguments.
C
   < Declarations >
C
C  Declare local variables.
C
   < Declarations >
C
C***FIRST^EXECUTABLE^STATEMENT^^NAME
   < Body of NAME >
^^^^^^END
```

****

ACKNOWLEDGEMENT

The authors wish to acknowledge the assistance provided by Dr. Frederick N. Fritsch of the Computing and Mathematics Research Division, Lawrence Livermore National Laboratory, who wrote Appendix E and made corrections and comments on the manuscript.

****

REFERENCES

[1] W. H. Vandevender and K. H. Haskell, The SLATEC mathematical subroutine library, SIGNUM Newsletter, 17, 3 (September 1982), pp. 16-21.

[2] P. A. Fox, A. D. Hall and N. L. Schryer, The PORT mathematical subroutine library, ACM Transactions on Mathematical Software, 4, 2 (June 1978), pp. 104-126.

[3] P. A. Fox, A. D. Hall and N. L. Schryer, Algorithm 528: framework for a portable library, ACM Transactions on Mathematical Software, 4, 2 (June 1978), pp. 177-188.

[4] R. E. Jones and D. K. Kahaner, XERROR, the SLATEC error-handling package, Software - Practice and Experience, 13, 3 (March 1983), pp. 251-257.

[5] R. F. Boisvert, S. E. Howe and D. K. Kahaner, GAMS: a framework for the management of scientific software, ACM Transactions on Mathematical Software, 11, 4 (December 1985), pp. 313-355.

[6] American National Standard Programming Language FORTRAN, ANSI X3.9-1978, American National Standards Institute, 1430 Broadway, New York, New York 10018, April 1978.

[7] W. S. Brown, A simple but realistic model of floating point computation, ACM Transactions on Mathematical Software, 7, 4 (December 1981), pp. 445-480.

[8] F. N. Fritsch, SLATEC/LDOC prologue: template and conversion program, Report UCID-21357, Rev.1, Lawrence Livermore National Laboratory, Livermore, California, November 1988.