NAG Library

Essential Introduction

Note: *this document is essential reading for any prospective user of the library.*

Contents

1 Summary for All Users

All users both familiar or unfamiliar with this Library who are thinking of using a routine from it, are asked to please follow these instructions:

(a) read the whole of this **Essential Introduction**;

(b) select an appropriate chapter or routine by:

- searching through the **Keyword Index**;
- searching via online search facility;
- consulting the **Library Contents** list; or
- searching through the **GAMS Classification Index**;

(c) read the relevant **Chapter Introduction**;

(d) choose a routine, and read the **routine document**. If the routine does not after all meet your needs, return to step (b);

(e) read the **Users' Note** for your implementation;

(f) consult local documentation, which should be provided by your local support staff, about access to the Library on your computing system;

(g) obtain an online copy of the example program for the particular routine of interest and experiment with it.

You should now be in a position to include a call to the routine in a program, and to attempt to compile and run it. You may of course need to refer back to the relevant documentation in the case of difficulties, for advice on assessment of results, and so on.

As you become familiar with the Library, some of steps (a) to (g) can be omitted, but it is always essential to:

be familiar with this **Essential Introduction**;

be familiar with the **Chapter Introduction**;

read the **routine document**;

be aware of the **Users' Note** for your implementation.

2 The Library and its Documentation

2.1 Structure of the Library

The NAG Library is a comprehensive collection of **routines** for the solution of numerical and statistical problems.

The NAG Library for SMP & Multicore is the same collection of routines as those available in the NAG Library, many of which have been specially tuned to maximize their performance on Symmetric Multiprocessor (SMP) machines. The document entitled 'Introduction to the NAG Library for SMP & Multicore' is ***essential*** reading for NAG Library for SMP & Multicore users.

The Library is divided into **chapters**, each devoted to a branch of numerical analysis or statistics. Each chapter has a three-character name and a title, e.g.,

Chapter D01 – Quadrature

Exceptionally, Chapters H and S have one-character names. The chapters and their names are based on the ACM modified SHARE classification index (see ACM (1960–1976)).

All documented routines in the Library have six-character names, beginning with the characters of the chapter name, e.g.,

D01AJF

Note that the second and third characters are **digits**, not letters; e.g., 0 is the digit zero, not the letter O. The last letter of each routine name almost always appears as 'F' in the documentation. Chapters C05, D03 and E04 have some routines whose last letter is 'A' rather than 'F'. An 'A' version is always paired with an 'F' routine, the 'A' version being safe to use in a multithreaded environment, but otherwise having identical functionality to the 'F' version.

Chapter F06 (Linear Algebra Support Routines) contains all the Basic Linear Algebra Subprograms, BLAS, with NAG-style names as well as with the actual BLAS names, e.g., F06PAF (DGEMV). The names in brackets are the equivalent double precision BLAS names. Chapter F07 (Linear Equations (LAPACK)) and Chapter F08 (Least-squares and Eigenvalue Problems (LAPACK)) contain routines derived from the LAPACK project. Like the BLAS, these routines have NAG-style names as well as LAPACK names, e.g., F07ADF (DGETRF). Details regarding these alternate names can be found in the relevant Chapter Introductions.

In order to take full advantage of machine-specific versions of BLAS and LAPACK routines provided by some computer hardware vendors, you are encouraged to use the BLAS and LAPACK names (e.g., DGEMV and DGETRF) rather than the corresponding NAG-style names (e.g., F06PAF and F07ADF) wherever possible in your programs.

2.2 Structure of the Documentation

The NAG Library Manual is the principal documentation for both the NAG Library and the NAG Library for SMP & Multicore. It has the same chapter structure as the Library: each chapter of routines in the Library has a corresponding chapter (of the same name) in the Manual. The chapters occur in alphanumeric order. General introductory documents appear at the beginning of the Manual.

Each chapter consists of the following documents:

Chapter Contents, e.g., Chapter D01;

Chapter Introduction, e.g., the D01 Chapter Introduction;

Routine Documents, one for each documented routine in the chapter.

A routine document has the same name as the routine which it describes. Within each chapter, routine documents occur in alphanumeric order. For those chapters that have both 'A' and 'F' versions of a routine, the routine descriptions are combined into one routine document.

Documentation is provided in the following formats:

XHTML+MathML, a fully linked version of the manual using XHTML and MathML (recommended for browsing) and providing links to the PDF version of each document (recommended for printing); and

PDF, a full PDF manual browsed using the PDF bookmarks, or via HTML index files.

Advice on viewing and navigating the formats available can be found in the document Online Documentation.

The most up-to-date version of the documentation is accessible via the NAG web site (see Section 5).

2.3 Implementations of the Library

The Library is available on many different computer systems. For each distinct system, an **implementation** of the Library is prepared by NAG, e.g., the Sun Solaris 32-bit implementation. The implementation is distributed to sites as a tested compiled library.

An implementation is usually specific to a range of machines (e.g., the SPARC systems); it may also be specific to a particular Fortran compiler, or compiler option (such as scalar or vector mode or thread safe).

Essentially the same facilities are provided in all implementations of the Library, but, because of differences in arithmetic behaviour and in the compilation system, routines cannot be expected to give identical results on different systems, especially for sensitive numerical problems.

The documentation supports all implementations of the Library, with the help of a few simple conventions, and a small amount of implementation-dependent information, which is published in a separate **Users' Note** for each implementation (see Section 4.4).

2.4 Library Identification

Periodically a new **Mark** of the NAG Library is released: new routines are added, corrections and/or improvements are made to existing routines; and occasionally routines are withdrawn if they have been superseded by improved routines.

You must know **which implementation**, **which precision** and **which mark** of the Library you are using or intend to use. To find out which implementation, precision and mark of the Library is available at your site, you can run a program which calls the NAG Library routine A00AAF.

The program could be:

```
EXTERNAL A00AAF
CALL     A00AAF()
END
```

Alternatively, the example program for A00AAF can be run using the `nagexample` scripts supplied with your implementation (see the Users' Note for details).

An example of the output is:

```
*** Start of NAG Library implementation details ***

Implementation title: IA32, Linux, Intel Fortran
           Precision: FORTRAN double precision
        Product Code: FLLUX22DC
                Mark: 22

*** End of NAG Library implementation details ***
```

2.5 Fortran Language Standards

All routines in the Library conform to the ISO Fortran 95 Standard (ISO (1997)), except for the use of the double precision complex data type COMPLEX*16.

3 Using the Library

3.1 General Advice

A NAG Library routine **cannot** be guaranteed to return meaningful results irrespective of the data supplied to it. Care and thought **must** be exercised in:

(a) formulating the problem;

(b) programming the use of library routines;

(c) assessing the significance of the results.

The Foreword to the Manual provides some further discussion of points (a) and (c); the remainder of Section 3 is concerned with (b).

3.2 Programming Advice

The Library and its documentation are designed with the assumption that you will write a calling program in Fortran (although it may be called from other languages – see Section 3.10).

When programming a call to a routine, read the routine document carefully, especially the description of the **Parameters**. This states clearly which parameters must have values assigned to them on entry to the routine, and which return useful values on exit. See Section 4.3 for further guidance.

The most common types of programming error in using the Library are:

incorrect parameters in a call to a Library routine;

calling the Library from a single precision program.

Therefore if a call to a Library routine results in an unexpected error message from the system (or possibly from within the Library), **check** the following:

Has the NAG routine been called with the correct number of parameters?

Do the parameters all have the correct type?

Have all array parameters been dimensioned correctly?

Does your program pass the correct precision parameters to the NAG Library routines?

Avoid the use of NAG-type names for your own program units or COMMON blocks: in general, do not use names which contain a three-character NAG chapter name embedded in them; they may clash with the names of an auxiliary routine or COMMON block used by the NAG Library.

3.3 Error Handling and the Parameter IFAIL

3.3.1 Errors, Failure and Warning Conditions

The error, failure or warning conditions considered here are those that can be detected by explicit coding in a Library routine. Such conditions must be anticipated by the author of the routine. They should not be confused with run-time errors detected by the compiling system, e.g., detection of overflow or failure to assign an initial value to a variable.

In the rest of this document we use the word 'error' to cover all types of error, failure or warning conditions detected by the routine. They fall roughly into three classes.

(i) On entry to the routine the value of a parameter is out of range. This means that it is not useful, or perhaps even meaningful, to begin computation.

(ii) During computation the routine decides that it cannot yield the desired results, and indicates a failure condition. For example, a matrix inversion routine will indicate a failure condition if it considers that the matrix is singular and so cannot be inverted.

(iii) Although the routine completes the computation and returns results, it cannot guarantee that the results are completely reliable; it therefore returns a warning. For example, an optimization routine may return a warning if it cannot guarantee that it has found a local minimum.

All three classes of errors are handled in the same way by the Library.

Each error which can be detected by a Library routine is associated with a number. These numbers, with explanations of the errors, are listed in Section 6 (Error Indicators and Warnings) in the routine document. Unless the document specifically states to the contrary, you should not assume that the routine necessarily tests for the occurrence of the errors in their order of error number, i.e., the detection of an error does not imply that other errors have or have not been detected.

3.3.2 The IFAIL Parameter

Most of the NAG Library routines which can be called directly by you have a parameter called IFAIL. This parameter is concerned with the NAG Library error trapping mechanism (and, for some routines, with controlling the output of error messages and advisory messages).

IFAIL has **two** purposes:

(i) to allow you to specify what action the Library routine should take if an error is detected;

(ii) to inform you of the outcome of the call of the routine.

For purpose (i), you **must** assign a value to IFAIL before the call to the Library routine. Since IFAIL is reset by the routine for purpose (ii), the parameter must be the name of a variable, **not** a literal or constant.

The value assigned to IFAIL before entry should be either 0 (**hard fail** option), or 1 or – 1 (**soft fail** option). If after completing its computation the routine has not detected an error, IFAIL is reset to 0 to indicate a **successful call.** Control returns to the calling program in the normal way. If the routine does detect an error, its action depends on whether the hard or soft fail option was chosen.

3.3.3 Hard Fail Option

If you set IFAIL to 0 before calling the Library routine, execution of the program will terminate if the routine detects an error. Before the program is stopped, this error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n
** NAG hard failure - execution terminated
```

where `XXXXXX` is the routine name, and `n` is the number associated with the detected error. An explanation of error number `n` is given in Section 6 of the routine document `XXXXXX`.

In addition, most routines output explanatory error messages immediately before the standard termination message shown above.

The hard fail option should be selected if you are in any doubt about continuing the execution of the program after an unsuccessful call to a NAG Library routine. For environments where it might be inappropriate to halt program execution when an error is detected it is recommended that the hard fail option is **not** used.

3.3.4 Soft Fail Option

To select this option, you must set IFAIL to 1 or -1 before calling the Library routine.

If the routine detects an error, IFAIL is reset to the associated error number; further computation within the routine is suspended and control returns to the calling program.

If you set IFAIL to 1, then no error message is output (**silent exit**). If the output of error messages is undesirable, then silent exit is recommended.

If you set IFAIL to -1 (**noisy exit**), then before control is returned to the calling program, the following error message is output:

```
** ABNORMAL EXIT from NAG Library routine XXXXXX: IFAIL = n
** NAG soft failure - control returned
```

In addition, most routines output explanatory error messages immediately before the above standard message.

It is most important to test the value of IFAIL on exit if the soft fail option is selected. A nonzero exit value of IFAIL implies that the call was not successful so it is imperative that your program be coded to take appropriate action. That action may simply be to print IFAIL with an explanatory caption and then terminate the program. Many of the example programs in Section 9 of the routine documents have IFAIL-exit tests of this form. In the more ambitious case, where you wish your program to continue, it is essential that the program can branch to a point at which it is **sensible** to resume computation.

The soft fail option puts the onus on you to handle any errors detected by the Library routine. With the proviso that you are able to implement it **properly**, it is clearly more flexible than the hard fail option since it allows computation to continue in the case of errors. In particular there are at least two cases where its flexibility is useful:

(i) where additional information about the error or the progress of computation is returned via some of the other parameters;

(ii) in some routines, 'partial' success can be achieved, e.g., a probable solution found but not all conditions fully satisfied, so the routine returns a warning. On the basis of the advice in Section 6 and elsewhere in the routine document, you may decide that this partially successful call is adequate for certain purposes.

3.3.5 Historical Note

The error handling mechanism described above was introduced into the NAG Library at Mark 12. It supersedes the earlier mechanism which for most routines allowed IFAIL to be set by you to 0 or 1 only. The new mechanism is compatible with the old except that the details of the messages output on hard failure have changed. The new mechanism also allows you to set IFAIL to -1 (soft failure, noisy exit).

A few routines (introduced mainly at Marks 7 and 8) use IFAIL in a different way to control the output of error messages, and also of advisory messages (see Chapter X04). In those routines IFAIL is regarded as a decimal integer whose least significant digits are denoted ba with the following significance:

$a = 0$: hard failure $\qquad$ $a = 1$: soft failure
$b = 0$: silent exit $\qquad$ $b = 1$: noisy exit

Details are given in the documents of the relevant routines; for those routines this alternative use of IFAIL remains valid.

3.4 Input/output in the Library

Most NAG Library routines perform no output to an external file, except possibly to output an error message. All error messages are written to a logical **error message** unit. This unit number (which is set by default to 6 in most implementations) can be changed by calling the Library routine X04AAF.

Some NAG Library routines may optionally output their final results, or intermediate results to monitor the course of computation. In general, output other than error messages is written to a logical **advisory message** unit. This unit number (which is also set by default to 6 in most implementations) can be changed by calling the Library routine X04ABF. Although it is logically distinct from the error message unit, in practice the two unit numbers may be the same. A few routines in Chapter E04 allow this unit number to be specified directly as an option.

All output from the Library is formatted.

There are only a few Library routines which perform input from an external file. These examples occur in Chapters E04, E05 and H. The unit number of the external file is a parameter to the routine, and all input is formatted.

You must ensure that the relevant Fortran unit numbers are associated with the desired external files, either by an OPEN statement in your calling program, or by operating system commands.

3.5 Auxiliary Routines

In addition to those Library routines which are documented and are intended to be called by you directly, the Library also contains many auxiliary routines.

In general, you need not be concerned with them at all, although you may be made aware of their existence if, for example, you examine a memory map of an executable program which calls NAG routines. The only exception is that when calling some NAG Library routines you may be required or allowed to supply the name of an auxiliary routine from the NAG Library as an external procedure parameter. The routine documents give the necessary details. In such cases, you only need to supply the name of the routine; you **never** need to know details of its parameter list.

NAG auxiliary routines have names which are similar to the name of the documented routine(s) to which they are related, but with last letter 'Z', 'Y', and so on, e.g.,

D01BAZ is an auxiliary routine called by D01BAF.

A few chapters contain auxiliary routines whose names are obtained by adding 50 to the second and third characters of the chapter name. For instance, Chapter E04 has an auxiliary routine with the name E54NFU which is normally used as the actual argument for the QPHESS parameter of E04NFA; the corresponding name to be used with E04NFF is E04NFU.

3.6 Dynamic Memory Allocation

Some NAG Library routines perform dynamic memory allocation to simplify their interfaces. Where possible, the amount of memory allocated by a routine will be given in the routine document (usually as a function of routine parameters). All memory allocated by NAG routines is deallocated before exit.

In the case where a routine detects a failure to dynamically allocate sufficient memory, the routine will set an error condition and return by setting IFAIL $= -999$, and exit with an appropriate error message.

3.7 License Management

If your implementation is license managed then your local site will have details on how the license management is implemented; please contact your site installer for details. To determine whether a valid license is available on your machine run the example program for A00ACF.

Should a valid license not be found when calling license managed routines from the Library then the routine will set an error condition, by setting IFAIL $= -399$, and exit with an appropriate error message. The appropriate environment variables should then be checked (e.g., NAG_KUSARI_FILE) to make sure this points to the licence file containing a valid licence, and the licence file should be checked for any obvious errors (e.g., the licence refers to a different implementation). If everything appears to be correct then please contact NAG (see Section 5 for details).

3.8 Thread Safety

Some implementations of the Library facilitate the use of threads; that is, you can call routines from the Library from within a multithreaded application. Fully thread safe libraries are provided for several platforms — for more information please contact your local Response Centre (see Section 5). See the Thread Safety document for more detailed guidance on using the Library in a multithreaded context. You may also need to refer to the Users' Note for details of whether your implementation of the Library has been compiled in a manner that facilitates the use of threads.

Note that in some implementations, the Library is linked with one or more vendor libraries to provide, for example, efficient BLAS routines. NAG cannot guarantee that any such vendor library is thread safe.

3.9 Performance on SMP systems

The introduction of multicore processors and the availability of more affordable multisocket systems mean that SMP systems are increasingly common. Users of the NAG Fortran Library on these systems may benefit directly from any SMP parallelism present in the underlying vendor library (e.g., in BLAS and LAPACK routines), and indirectly via any Library routine which internally uses these parallelised vendor routines. To benefit from this, you should set the appropriate environment variable (usually OMP_NUM_THREADS) to the desired number of threads. Generally this should not be more than the number of available (idle) cores on your system. You should consult the relevant vendor library documentation for further information and contact NAG (see Section 5 for details) for advice if required.

The NAG SMP Library is designed to give further benefit on SMP systems. Key Library routines have been explicitly parallelised using OpenMP to offer enhanced performance over a wider range of routines compared with the NAG Fortran Library which relies solely on the parallelism in vendor libraries. Further information is given in the Introduction to the NAG Library for SMP & Multicore.

Note that the performance increase achieved, if any, will vary depending upon which routine is called, problem sizes and other parameters, system design and operating system configuration. If you frequently call a routine with similar data sizes and other parameters, it may be worthwhile to experiment with different numbers of threads, to determine the choice that gives optimal performance.

3.10 Calling the Library from Other Languages

In general the NAG Library can be called from other computer languages (such as C and Visual Basic) provided that appropriate mappings exist between their data types.

NAG has produced C Header Files which comprise of a set of header files, indicating the match between C and Fortran data types for various compilers, documentation and examples. The documentation, examples and C Header Files are available from the NAG Web sites (see Section 5).

The Dynamic Link Library (DLL) implementation can be called in a straightforward manner from a number of languages and environments, e.g., Visual Basic, Visual Basic for Applications (Excel), Delphi, C and C++. Guidance on this is provided as part of NAG Library DLLs. Further details can be found on the NAG Web sites.

4 Using the Documentation

4.1 Using the Manual

The Manual is designed to serve the following functions for both the NAG Library and the NAG Library for SMP & Multicore:

to give background information about different areas of numerical and statistical computation;

to advise on the choice of the most suitable NAG Library routine or routines to solve a particular problem;

to give all the information needed to call a NAG Library routine correctly from a Fortran program, and to assess the results.

At the beginning of the Manual are some general introductory documents which provide some background and additional information. There are a small number of documents which are specific to NAG Library or to the NAG Library for SMP & Multicore, you only need to read those specific to the library you are using. All other general introductory documents are relevant to both libraries.

The document entitled 'Introduction to the NAG Library for SMP & Multicore' is ***essential*** reading for NAG Library for SMP & Multicore users.

The documents entitled 'Mark 22 NAG Fortran Library News' and 'Mark 22 NAG Library for SMP & Multicore News' provide details of new routines added, details of routines scheduled for withdrawal and details of routines withdrawn at this mark. The Mark 22 NAG Library for SMP & Multicore News also provides details of routines which have been tuned or enhanced at this mark; a full list of such routines is available in the document 'Tuned and Enhanced Routines in the NAG Library for SMP & Multicore'.

The document entitled 'Library Contents' (a structured list of routines in the Library, by chapter) may help you to find the chapter, and possibly the routine, which you need to solve your problem.

The document entitled 'Routines Withdrawn or Scheduled for Withdrawal' provides full details of all routines withdrawn from the NAG Library and the document entitled 'Advice on Replacement Calls for Withdrawn/Superseded Routines' provides details of new functionality provided by replacement routines.

The online documentation provides you with a fully linked HTML Keyword Index (a keyword index to routines) and GAMS Classification Index (a list of NAG routines classified according to the GAMS scheme).

Having found a likely chapter or routine, you should read the corresponding **Chapter Introduction,** which gives background information about that area of numerical computation, and recommendations on the choice of a routine, including indexes, tables or decision trees.

When you have chosen a routine, you must consult the **routine document**. Each routine document is essentially self-contained (it may, however, contain references to related documents). It includes a description of the method, detailed specifications of each parameter, explanations of each error exit, remarks on accuracy, and (in most cases) an example program to illustrate the use of the routine.

4.2 Structure of Routine Documents

All routine documents have the same structure consisting of nine numbered sections:

1. **Purpose**
2. **Specification**
3. **Description**
4. **References**
5. **Parameters** (see Section 4.3 below)
6. **Error Indicators and Warnings**
7. **Accuracy**
8. **Further Comments**
9. **Example** (see Section 4.5 below)

In a few documents (notably Chapters E04, E05 and H) there are a further three sections:

10. Algorithmic Details
11. Optional Parameters
12. Description of Monitoring Information

4.3 Specification of Parameters

Section 5 of each routine document contains the specification of the parameters, in the order of their appearance in the parameter list.

4.3.1 Classification of parameters

Parameters are classified as follows.

Input: you must assign values to these parameters on or before entry to the routine, and these values are unchanged on exit from the routine.

Output: you need not assign values to these parameters before entry to the routine; the routine may assign values to them.

Input/Output: you must assign values to these parameters before entry to the routine, and the routine may then change these values.

Workspace: array parameters which are used as workspace by the routine. You must supply arrays of the correct type and dimension. In general, you need not be concerned with their contents.

Communication Array: parameters which are used to communicate data from one routine call to another.

External Procedure: a routine which must be supplied (e.g., to evaluate an integrand or to print intermediate output). Usually it must be supplied as part of your calling program, in which case its specification includes full details of its parameter list and specifications of its parameters (all enclosed in a box). Its parameters are classified in the same way as those of the Library routine, but because you must write the procedure rather than call it, the significance of the classification is different.

> *Input*: values may be supplied on entry, which your procedure **must not** change.
>
> *Output*: you may or must assign values to these parameters before exit from your procedure.
>
> *Input/Output*: values may be supplied on entry, and you may or must assign values to them before exit from your procedure.

Occasionally, as mentioned in Section 3.5, the procedure can be supplied from the NAG Library, and then you only need to know its name.

User Workspace: array parameters which are passed by the Library routine to an external procedure parameter. They are not used by the routine, but you may use them to pass information between your calling program and the external procedure.

Dummy: a simple variable which is not used by the routine. A variable or constant of the correct type must be supplied, but its value need not be set. (A dummy parameter is usually a parameter which was required by an earlier version of the routine and is retained in the parameter list for compatibility.)

4.3.2 Constraints and suggested values

The word '*Constraint*:' or '*Constraints*:' in the specification of an *Input* parameter introduces a statement of the range of valid values for that parameter, e.g.,

> *Constraint*: $N > 0$.

If the routine is called with an invalid value for the parameter (e.g., $N = 0$), the routine will usually take an error exit, returning a nonzero value of IFAIL (see Section 3.3).

Constraints on parameters of type CHARACTER only list upper case alphabetic characters, e.g.,

> *Constraint*: CHECK = 'N'.

In practice, all routines with CHARACTER parameters will permit the use of lower case characters.

The phrase '*Suggested Values*:' introduces a suggestion for a reasonable initial setting for an *Input* parameter (e.g., accuracy or maximum number of iterations) in case you are unsure what value to use; you should be prepared to use a different setting if the suggested value turns out to be unsuitable for your problem.

4.3.3 Array parameters

Most array parameters have dimensions which depend on the size of the problem. In Fortran terminology they have 'adjustable dimensions': the dimensions occurring in their declarations are integer variables which are also parameters of the Library routine.

For example, a Library routine might have the specification:

```
SUBROUTINE <name> (M, N, A, B, LDB)
INTEGER            M, N, A(N), B(LDB,N), LDB
```

For a **one-dimensional** array parameter, such as A in this example, the specification would begin

A(N) – INTEGER array

You must ensure that the dimension of the array, as declared in your calling (sub)program, is at least as large as the value you supply for N. It may be larger, but the routine uses only the first N elements.

For a **two-dimensional** array parameter, such as B in the example, the specification might be

B(LDB,N) – INTEGER array

On entry: the m by n matrix B.

and the parameter LDB might be described as follows:

LDB – INTEGER

On entry: the first dimension of the array B as declared in the (sub)program from which <name> is called.

Constraint: LDB $\geq$ M.

You **must** supply the **first** dimension of the array B, as declared in your calling (sub)program, through the parameter LDB, even though the number of rows actually used by the routine is determined by the parameter M. You must ensure that the first dimension of the array is at least as large as the value you supply for M. The extra parameter LDB is needed because Fortran 77 does not allow information about the dimensions of array parameters to be passed automatically to a routine.

You must also ensure that the **second** dimension of the array, as declared in your calling (sub)program, is at least as large as the value you supply for N. It may be larger, but the routine uses only the first N columns.

A program to call the hypothetical routine used as an example in this section might include the statements:

```
INTEGER AA(100), BB(100,50)
LDB = 100
.
.
.
M = 80
N = 20
CALL <name>(M,N,AA,BB,LDB)
```

Many NAG routines contain array parameters declared with the 'assumed size' array dimension, and would be given as

```
INTEGER        A(*), B(LDB,*)
```

However, the original declaration of an array in your calling program must always have constant dimensions, greater than or equal to the minimum value documented.

Consult an expert or a textbook on Fortran if you have difficulty in calling NAG routines with array parameters.

4.4 Implementation-dependent Information

In order to support all implementations of the Library, the Manual has adopted a convention of using ***bold italics*** to distinguish terms which have different interpretations in different implementations.

The most important bold italicised terms and their usual interpretation are as follows:

real	means REAL
double precision	means DOUBLE PRECISION
complex	means COMPLEX
complex*16	means COMPLEX*16 (or equivalent)
basic precision	means double precision
additional precision	means quadruple precision
reduced precision	means single precision

Another important bold italicised term is ***machine precision***, which denotes the relative precision to which ***double precision*** floating-point numbers are stored in the computer, e.g., in an implementation with approximately 16 decimal digits of precision, ***machine precision*** has a value of approximately 10^{-16}.

The precise value of ***machine precision*** is given by the routine X02AJF. Other routines in Chapter X02 return the values of other implementation-dependent constants, such as the overflow threshold, or the largest representable integer. Refer to the X02 Chapter Introduction for more details.

The bold italicised term ***block size*** is used only in Chapters F07 and F08. It denotes the block size used by block algorithms in these chapters. You only need to be aware of its value when it affects the amount of workspace to be supplied – see the parameters WORK and LWORK of the relevant routine documents and the Chapter Introduction.

In Chapters F06, F07 and F08, alternate routine names are available for BLAS and LAPACK derived routines. For details of the alternate routine names please refer to the relevant Chapter Introduction.

For each implementation of the Library, a separate **Users' Note** is published. This is a short document, revised at each mark. At most installations it is available in machine-readable form. It gives any necessary additional information which applies specifically to that implementation, in particular:

the interpretation of bold italicised terms;

the values returned by Chapter X02 routines;

the default unit numbers for output (see Section 3.4).

4.5 Example Programs and Results

The **example program** in Section 9 of most routine documents illustrates a simple call of the routine. The programs are designed so that they can be fairly easily modified, and so serve as the basis for a simple program to solve your problem.

For each implementation of the Library, NAG distributes the example programs in machine-readable form, with all necessary modifications already applied. Many sites make the programs accessible to you in this form. Generic forms of the programs, without implementation-specific modifications, may be obtained directly from the NAG Web site. The Users' Note for your implementation will mention any special changes which need to be made to the example programs.

These example programs contain a number of preprocessor identifiers such as `NAG_CALL` and `NAG_IFMT` to enable cross platform portability. These identifiers are subsequently replaced by appropriate implementation specific tokens via the header files `nag.h` or `nag_types.h`, using the C preprocessor #defines.

Note that the results obtained from running the example programs may not be identical in all implementations, and may not agree exactly with the results in the Manual which were obtained from a double precision implementation (with approximately 16 digits of precision).

For many routine documents, a plot of the example program results is also provided. In some cases the example program has been modified slightly to produce larger sets of results to give a more representative plot of the solution profile produced.

5 Support from NAG

NAG Response Centres

The NAG Response Centres are available for general enquiries from all users and also for technical queries from sites that subscribe to the support service.

The Response Centres are open during office hours, but contact is possible by fax, email and telephone (answering machine) at all times. Please see the Users' Note or the NAG web site for contact details.

When contacting one of the NAG Response Centres, it helps us to deal with your query quickly if you can quote your NAG user reference and NAG product code.

NAG Web Site

The NAG web site is an information service providing items of interest to users and prospective users of NAG products and services. The information is regularly updated and reviewed, and includes implementation availability, descriptions of products, downloadable software, case studies, industry articles and technical reports. The NAG web site can be accessed via:

NAG UK at http://www.nag.co.uk
NAG North America at http://www.nag.com
NAG Japan at http://www.nag-j.co.jp

6 Background to NAG

Various aspects of the design and development of the NAG Library, and NAG's technical policies and organisation are given in Ford (1982), Ford *et al.* (1979), Ford and Pool (1984), and Hague *et al.* (1982).

7 References

ACM (1960–1976) Collected algorithms from ACM index by subject to algorithms

ANSI (1966) USA standard Fortran *Publication X3.9* American National Standards Institute

ANSI (1978) American National Standard Fortran *Publication X3.9* American National Standards Institute

ANSI/IEEE POSIX (1995) POSIX Standard Thread Library ANSI/IEEE POSIX 1003.1c:1995

Ford B (1982) Transportable numerical software *Lecture Notes in Computer Science* **142** 128–140 Springer–Verlag

Ford B, Bentley J, Du Croz J J and Hague S J (1979) The NAG Library 'machine' *Softw. Pract. Exper.* **9 (1)** 65–72

Ford B and Pool J C T (1984) The evolving NAG Library service *Sources and Development of Mathematical Software* (ed W Cowell) 375–397 Prentice–Hall

Hague S J, Nugent S M and Ford B (1982) Computer-based documentation for the NAG Library *Lecture Notes in Computer Science* **142** 91–127 Springer–Verlag

ISO (1997) ISO Fortran 95 programming language (ISO/IEC 1539–1:1997)

ISO/IEC (1990) Information technology – Programming Language C *Current C Language Standard* ISO/IEC 9899:1990

Kernighan B W and Ritchie D M (1988) *The C Programming Language* (2nd Edition) Prentice–Hall

OpenMP *The OpenMP specification for parallel programming*http://www.openmp.org